

- [Table of Contents](#)

Understanding and Deploying LDAP Directory Services, Second Edition

By Timothy A. Howes Ph.D., Mark C. Smith, Gordon S. Good

Publisher: Addison Wesley

Pub Date: May 02, 2003

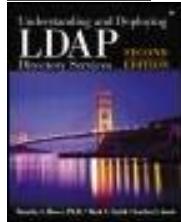
ISBN: 0-672-32316-8

Pages: 936

Slots: 1

Lightweight Directory Access Protocol (LDAP) is the standard for directory information access and is the underlying protocol for a variety of email systems, Web systems, and enterprise applications. LDAP enables central management of users, groups, devices, and other data, thereby simplifying directory management and reducing the total cost of ownership.

Understanding and Deploying LDAP Directory Services, written by the creators of the protocol, is known as the LDAP bible and is the classic text for learning about LDAP and how to utilize it effectively. The Second Edition builds on this success by acting as an exhaustive resource for designing, deploying, and maintaining LDAP directory services. Topics such as implementation pitfalls, establishing and maintaining user access to information, troubleshooting, and real-world scenarios will be thoroughly explored.



- [Table of Contents](#)

Understanding and Deploying LDAP Directory Services, Second Edition

By Timothy A. Howes Ph.D., Mark C. Smith, Gordon S. Good

Publisher: Addison Wesley

Pub Date: May 02, 2003

ISBN: 0-672-32316-8

Pages: 936

Slots: 1

[Copyright](#)

[Preface](#)

[The Book's Organization](#)

[The Book's Audience](#)

[Conventions Used in This Book](#)

[Contacting Us](#)

[Acknowledgments](#)

[About the Authors](#)

[Part I. Introduction to Directory Services and LDAP](#)

[Chapter 1. Directory Services Overview and History](#)

[What a Directory Is](#)

[What a Directory Can Do for You](#)

[What a Directory Is Not](#)

[The History and Origins of LDAP](#)

[Further Reading](#)

[Looking Ahead](#)

[Chapter 2. Introduction to LDAP](#)

[What Is LDAP?](#)

[The LDAP Models](#)

[LDIF](#)

[LDAP Server Software](#)

[LDAP Command-Line Utilities](#)

[LDAP APIs](#)

[LDAP and Internationalization](#)

[LDAP Overview Checklist](#)

[Further Reading](#)

[Looking Ahead](#)

[Chapter 3. LDAPv3 Extensions](#)

[How LDAPv3 Is Extended](#)

[The Root DSE and Extension Discovery](#)

[Selected LDAPv3 Extensions](#)

[Future Directions: Where Is LDAP Headed Next?](#)

[LDAP Extensions and Future Directions Checklists](#)

[Further Reading](#)

[Looking Ahead](#)

[Chapter 4. Overview of Netscape Directory Server](#)

[Basic Installation](#)

[A Brief Hands-on Tour of Netscape Directory Server](#)

[Product Focus and Feature Set](#)

[Extending the Netscape Server: A Simple Plug-in Example](#)

[Further Reading](#)

[Looking Ahead](#)

[Part II. Designing Your Directory Service](#)

[Chapter 5. Directory Design Road Map](#)

[The Directory Life Cycle](#)

[Directory Design Checklist](#)

[Further Reading](#)

[Looking Ahead](#)

[Chapter 6. Defining Your Directory Needs](#)

[Overview of the Directory Needs Definition Process](#)

[Analyzing Your Environment](#)

[Determining and Prioritizing Application Needs](#)

[Determining and Prioritizing Users' Needs and Expectations](#)

[Determining and Prioritizing Deployment Constraints](#)

[Determining and Prioritizing Other Environmental Constraints](#)

[Choosing an Overall Directory Design and Deployment Approach](#)

[Setting Some Goals and Milestones](#)

[Defining Your Directory Needs Checklist](#)

[Further Reading](#)

[Looking Ahead](#)

[Chapter 7. Data Design](#)

[Data Design Overview](#)

[Common Data-Related Problems](#)

[Creating a Data Policy Statement](#)

[Identifying Which Data Elements You Need](#)

[General Characteristics of Data Elements](#)

[Sources of Data](#)

[Maintaining Good Relationships with Other Data Sources](#)

[Data Design Checklist](#)

[Further Reading](#)

[Looking Ahead](#)

[Chapter 8. Schema Design](#)

[The Purpose of a Schema](#)

[Elements of LDAP Schemas](#)

[Directory Schema Formats](#)

[The Schema-Checking Process](#)

[Schema Design Overview](#)

[Sources of Predefined Schemas](#)

[Defining New Schema Elements](#)

[Documenting and Publishing Your Schemas](#)

[Schema Maintenance and Evolution](#)

[Schema Design Checklist](#)

[Further Reading](#)

[Looking Ahead](#)

[Chapter 9. Namespace Design](#)

[The Structure of a Namespace](#)

[The Purposes of a Namespace](#)

[Analyzing Your Namespace Needs](#)

[Examples of Namespaces](#)

[Namespace Design Checklist](#)

[Further Reading](#)

[Looking Ahead](#)

[Chapter 10. Topology Design](#)

[Directory Topology Overview](#)

[Gluing the Directory Together: Knowledge References](#)

[Authentication in a Distributed Directory](#)

[Advantages and Disadvantages of Partitioning](#)

[Designing Your Directory Server Topology](#)

[Topology Design Checklist](#)

[Further Reading](#)

[Looking Ahead](#)

[Chapter 11. Replication Design](#)

[Why Replicate?](#)

[Replication Concepts](#)

[Advanced Replication Features](#)

[Designing Your Directory Replication System](#)

[Replication Design Checklist](#)

[Further Reading](#)

[Looking Ahead](#)

[Chapter 12. Privacy and Security Design](#)

[Security Guidelines](#)

[The Purpose of Security](#)

[Security Threats](#)

[Security Tools](#)

[Analyzing Your Security and Privacy Needs](#)

[Designing for Security](#)

[Privacy and Security Design Checklist](#)

[Further Reading](#)

Looking Ahead

Part III. Deploying Your Directory Service

Chapter 13. Evaluating Directory Products

Making the Right Product Choice

Categories of Directory Software

Evaluation Criteria for Directory Software

Reaching a Decision

Evaluating Directory Products Checklist

Further Reading

Looking Ahead

Chapter 14. Piloting Your Directory Service

A Piloting Road Map

Piloting Your Directory Service Checklist

Looking Ahead

Chapter 15. Analyzing and Reducing Costs

The Politics of Costs

Reducing Costs

Design, Piloting, and Deployment Costs

Ongoing Costs of Providing Your Directory Service

Analyzing and Reducing Costs Checklist

Further Reading

Looking Ahead

Chapter 16. Putting Your Directory Service into Production

Creating a Plan for Putting Your Directory Service into Production

Advice for Putting Your Directory Service into Production

Executing Your Plan

Putting Your Directory Service into Production Checklist

Looking Ahead

Part IV. Maintaining Your Directory Service

Chapter 17. Backups and Disaster Recovery

Backup and Restore Procedures

Disaster Planning and Recovery

Directory-Specific Issues in Disaster Recovery

Backups and Disaster Recovery Checklist

Further Reading

Looking Ahead

Chapter 18. Maintaining Data

The Importance of Data Maintenance

The Data Maintenance Policy

Handling New Data Sources

Handling Exceptions

Checking Data Quality

Maintaining Data Checklist

Further Reading

[Looking Ahead](#)

[Chapter 19. Monitoring](#)

[Introduction to Monitoring](#)

[Selecting and Developing Monitoring Tools](#)

[Notification Techniques](#)

[Taking Action](#)

[A Sample Directory Monitoring Utility](#)

[Performance Analysis](#)

[Monitoring Checklist](#)

[Further Reading](#)

[Looking Ahead](#)

[Chapter 20. Troubleshooting](#)

[Discovering Problems](#)

[Types of Problems](#)

[Troubleshooting and Resolving Problems](#)

[Troubleshooting Checklist](#)

[Further Reading](#)

[Looking Ahead](#)

[Part V. Leveraging Your Directory Service](#)

[Chapter 21. Developing New Applications](#)

[Reasons to Develop Directory-Enabled Applications](#)

[Common Ways That Applications Use Directories](#)

[Tools for Developing LDAP Applications](#)

[Advice for LDAP Application Developers](#)

[Example 1: setpwd, a Password-Resetting Utility](#)

[Example 2: SimpleSite, a Web Site with User Profile Storage](#)

[Developing New Applications Checklist](#)

[Further Reading](#)

[Looking Ahead](#)

[Chapter 22. Directory-Enabling Existing Applications](#)

[Reasons to Directory-Enable Existing Applications](#)

[Advice for Directory-Enabling Existing Applications](#)

[Example 1: A Directory-Enabled finger Service](#)

[Example 2: Adding LDAP Address Lookup to an E-Mail Client](#)

[Directory-Enabling Existing Applications Checklist](#)

[Further Reading](#)

[Looking Ahead](#)

[Chapter 23. Directory Coexistence](#)

[Why Is Coexistence Important?](#)

[Coexistence Techniques](#)

[Privacy and Security Considerations](#)

[Determining Your Coexistence Requirements](#)

[Directory Coexistence Implementation Considerations](#)

[Example: The Idapsync Tool: One-Way Synchronization with Join](#)

[Directory Coexistence Checklist](#)

[Further Reading](#)

[Looking Ahead](#)

Part VI. Case Studies

Chapter 24. Case Study: Netscape Communications Corporation

[Overview of the Organization](#)

[Directory Drivers](#)

[Directory Service Design](#)

[Directory Service Deployment](#)

[Directory Service Maintenance](#)

[Leveraging the Directory Service](#)

[Summary and Lessons Learned](#)

[Further Reading](#)

[Looking Ahead](#)

Chapter 25. Case Study: A Large Multinational Enterprise

[Overview of the Organization](#)

[Directory Drivers](#)

[Directory Service Design](#)

[Directory Service Deployment](#)

[Directory Service Maintenance](#)

[Leveraging the Directory Service](#)

[Summary and Lessons Learned](#)

[Further Reading](#)

[Looking Ahead](#)

Chapter 26. Case Study: An Enterprise with an Extranet

[Overview of the Organization](#)

[Directory Drivers](#)

[Directory Service Design](#)

[Directory Service Deployment](#)

[Directory Service Maintenance](#)

[Leveraging the Directory Service](#)

[Summary and Lessons Learned](#)

[Looking Ahead](#)

Copyright

Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in this book, and Addison-Wesley was aware of a trademark claim, the designations have been printed with initial capital letters or in all capitals.

The authors and publisher have taken care in the preparation of this book, but make no expressed or implied warranty of any kind and assume no responsibility for errors or omissions. No liability is assumed for incidental or consequential damages in connection with or arising out of the use of the information or programs contained herein.

Netscape browser frame © 2002 Netscape Communications Corporation. Netscape Directory Server software © 2002 America Online, Inc. Screenshots used with permission.

The publisher offers discounts on this book when ordered in quantity for bulk purchases and special sales. For more information, please contact:

U.S. Corporate and Government Sales

(800) 382-3419

corpsales@pearsontechgroup.com

For sales outside of the U.S., please contact:

International Sales

(317) 581-3793

international@pearsontechgroup.com

Visit Addison-Wesley on the Web: www.awprofessional.com

Library of Congress Cataloging-in-Publication Data

Howes, Tim.

Understanding and deploying LDAP directory services / Timothy A.

Howes, Mark C. Smith, and Gordon S. Good.-- 2nd ed.

p. cm.

Includes bibliographical references and index.

ISBN 0-672-32316-8 (acid-free paper)

1. Directory services (Computer network technology) 2. Computer network protocols. 3.

LDAP (Computer network protocol) I. Smith, Mark, 1966- II. Good, Gordon S. III. Title.

TK5105.595 .H69 2003

005.7'1369--dc21 2003000318

Copyright © 2003 by Pearson Education, Inc.

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form, or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior consent of the publisher. Printed in the United States of America. Published simultaneously in Canada.

For information on obtaining permission for use of material from this work, please submit a written request to:

Pearson Education, Inc.

Rights and Contracts Department

75 Arlington Street, Suite 300

Boston, MA 02116

Fax: (617) 848-7047

Text printed on recycled paper

1 2 3 4 5 6 7 8 9 10—HT—0706050403

First printing, April 2003

Dedication

For Nancy, whose love and support make it all worthwhile. And for Chewy, too.

—Tim Howes

For my wife, Kathy, who supports me wonderfully in everything I do.

—Mark Smith

For my mother and father, and my brothers, Brian and Kevin.

—Gordon Good

Preface

In the past decade, LDAP directories have risen from a relatively obscure offshoot of an equally obscure field to become one of the linchpins of modern computing. Increasingly, LDAP directories are becoming the nerve center of an organization's computing infrastructure, providing naming, location, management, security, and other services that have traditionally been provided by network operating systems. Design and deployment of a successful LDAP directory service can be complex and challenging, yet little information is available explaining the ins and outs of this important task.

When two of us (Mark and Tim) finished writing a previous book, *LDAP: Programming Directory-Enabled Applications with Lightweight Directory Access Protocol*, in early 1997, we soon realized there was another, much bigger piece of the directory puzzle still to be addressed. The previous book was aimed at directory application programmers, but nothing similar was available to address the needs of directory decision makers, designers, and administrators. This book is aimed at that audience.

Recognizing the size of the task ahead of us and remembering the joys of giving up evenings and weekends for months at a time to meet deadlines for our first book, we quickly decided to expand our team. Just as quickly, we decided there was no one we'd rather share the fun with than our longtime friend and colleague, Gordon Good, at the time a senior directory developer at Netscape. Aside from being the third leg of the LDAP development team at the University of Michigan (U-M), Gordon brought a wealth of system administration experience from his past life as a directory and e-mail administrator and Web master for U-M. With Gordon on board, the three of us set about writing a book that we only half-jokingly referred to as the "LDAP Bible." The first edition of *Understanding and Deploying LDAP Directory Services* was published in 1999.

Two years later, we realized that it was time to update this book and publish a second edition. LDAPv3 work in the IETF was mostly complete. Numerous extensions to the basic LDAP protocol were being developed. LDAP support in commercial and open-source software was widespread. In this edition, we cover these recent directory services developments. In addition, in response to reader suggestions we have streamlined the text, added more hands-on examples, updated the examples to reflect currently available software versions, and updated the case studies to reflect current directory practice. We thank all the readers of the first edition who provided helpful suggestions, and we hope that you find this second edition even more valuable.

The Book's Organization

This book includes 26 chapters in 6 parts. [Part I](#) introduces directories and LDAP. [Parts II](#) through [IV](#) each address a different part of the directory life cycle. [Part V](#) discusses how to leverage your directory service after it's up and running. Finally, [Part VI](#) presents three directory services deployment case studies.

[Part I](#), Introduction to Directory Services and LDAP, provides a comprehensive introduction to directories and LDAP. For readers unfamiliar with the topic, this section should bring them up to speed and provide the background necessary to understand the rest of the book. It also includes a section on the history of directories for readers interested in how all this technology came about.

[Part II](#), Designing Your Directory Service, begins to delve into the directory life cycle by covering the first, and in many ways most important, phase: design. We cover all aspects of directory design, from determining your needs, to designing your data sources, schema, namespace, topology, replication, and finally privacy and security.

[Part III](#), Deploying Your Directory Service, covers the next phase in the directory life cycle: deployment. We cover everything from choosing the right directory products to piloting your service to putting your service into production. We've also included a chapter about analyzing the cost of your service and how to help reduce those costs.

[Part IV](#), Maintaining Your Directory Service, concludes our coverage of the directory life cycle with a look at the maintenance phase. We cover such topics as backups and disaster recovery, maintaining data, monitoring your directory system, and troubleshooting problems when they occur.

[Part V](#), Leveraging Your Directory Service, talks about how to take advantage of the service you have designed and deployed. We discuss how to directory-enable existing applications, how to create new applications that use the directory, and how your directory can coexist with other data sources.

[Part VI](#), Case Studies, closes the book by presenting several directory case studies. Some of the case studies presented are real, and some are fictitious, but all are designed to illustrate the concepts of directory design, deployment, and maintenance in action.

The Book's Audience

This book is intended for primarily three kinds of readers: decision makers, architects, and administrators. In addition, anyone who wants to know more about LDAP or directories in general will find the book useful, as will software engineers who develop directory applications.

Directory decision makers will find this book useful for aiding an understanding of directories and the kinds of business problems they help solve. Decision makers will find [Part I](#) useful for explaining the basics of directories. [Part VI](#) should also prove useful by providing some realistic examples of how directories are used and the benefits they can bring.

Directory architects will find this book useful in defining the design problem and providing a methodology for producing a comprehensive directory design. The design methodology is focused on a practical approach to design based on real-world requirements. We highly recommend that directory architects and designers read the whole book, paying special attention to [Parts II](#), [III](#), and [IV](#). A good directory design results in large part from a clear understanding of the other aspects of the directory life cycle and how the directory will be used.

Directory administrators will find [Part IV](#) especially useful. It focuses on the maintenance phase of the directory life cycle, where administrators spend much of their lives. We also highly recommend that administrators read the rest of the book to get an idea of the directory big picture, as well as to understand some of the directory design decisions that are bound to make their lives either miserable or enjoyable.

Other interested readers can pick and choose from the sections of the book that interest them. We encourage all readers to at least skim [Part I](#), to ensure that they have the background required to benefit from the rest of the book. We've tried to structure the book so that each chapter stands by itself as much as possible. Readers should be able to read the chapters covering topics that interest them, without wading through chapters of less interest.

Finally, we think all readers will find the case studies presented in [Part VI](#) interesting. They give different perspectives on directories designed to illustrate the trade-offs that different directory needs imply.

Conventions Used in This Book

Commands, file names, text output, and code are presented in a constant width typeface. For example:

```
mkdir /export/dsinstall
```

Command and output lines that are too long to fit on one line in the book are broken across two or more lines like this:

```
ldapsearch -b "dc=example,dc=com" -s sub "(&(givenName=Gordon)(sn=Good))" cn ou  
mail
```

➥ postalAddress

Text that serves as a placeholder is shown in italics. For example, in the following command "id" is a placeholder:

```
finger id@example.com
```

Newly introduced terms are also shown in italics. For example:

LDAP is a *message-oriented* protocol.

Keystrokes and textual labels that appear on screen are shown in bold, like this:

Press the **Enter** key or click the **Install** button to continue.

Contacting Us

If you have comments or suggestions about this book, or if you'd like to tell us about an interesting directory deployment or application you've developed, we'd like to hear from you. Feel free to drop us a line at the following addresses:

Tim Howes

howes@opsware.com

Mark Smith

mark@bradesmith.com

Gordon Good

ggood@opsware.com

We'll try our best to get back to you, but keep in mind that we all have day jobs!

Acknowledgments

We'd like to thank our management and the publishing relations department at Netscape, whose support enabled us to write this book. Specifically, our thanks go out to Suzanne Anthony, Ben Horowitz, Claire Hough, William Morris, John Paul, Neel Phadnis, Susan Walton, and David Weiden.

In addition, we'd like to thank the Netscape IS department for its help with the first edition of this book. Bob Ferguson, Gene Irvine, and especially the incredible Leif Hedstrom deserve special thanks.

We'd like to thank the people who reviewed parts of this book, including David Boreham, Kathleen Brade, Nancy Cartwright, Leif Hedstrom, Richard Hesse, Chuck Lever, John Merrells, Rob Powers, Gordon Shephard, Mike SoRelle, and Chuck Woods.

We'd also like to thank the publishers. Thanks to Brett Bartow, Linda Engelman, Jessica Goldstein, and Dayna Isley for their guidance and gentle prodding. Special thanks to Kitty Jarrett for her professionalism in making the process of creating the first edition go so smoothly. Also, thanks to Benjamin Hart, Greg Pearson, Louis Porter, Jr., Mary Ellen Stephenson, Chris Wilcox, and Laura Williams for their work on the first edition. And last but not least, thanks to Geneil Breeze, Audrey Doyle, Elizabeth Finney, Stephanie Hiebert, Heather McNeill, and Elizabeth Ryan for their work on this edition.

About the Authors

Timothy A. Howes is cofounder, CTO, and Executive Vice President of Development at Opsware Inc., the leading provider of data center automation software. Prior to cofounding Opsware, Dr. Howes was Vice President of Technology for America Online, Inc., where he worked on technology strategy, acquisitions, and investments. Before that, he was a Netscape Fellow, Chief Technology Officer of Netscape's Server Products division, and chief architect of several Netscape server products. Dr. Howes coinvented the Lightweight Directory Access Protocol (LDAP), the Internet standard for directories. He has coauthored two books on directories and has written or cowritten more than 20 Internet Requests For Comments (RFCs). Dr. Howes is a former member of the Internet Architecture Board and former cochair of the LDAP Extensions, ASID, and IDS working groups in the Internet Engineering Task Force (IETF). Before joining Netscape, Dr. Howes was a researcher and National Science Foundation project director at the University of Michigan. He holds a Ph.D. in computer science from the University of Michigan.

Mark C. Smith is the chief architect for directory products at Netscape Communications Corporation, an AOL Time Warner company. Mark is responsible for the technical evolution of Netscape Directory Server and several other AOL products and services. He was previously a driving force behind the University of Michigan's LDAP implementation and a key designer of the university's directory service. Mark is coauthor of *LDAP: Programming Directory-Enabled Applications with Lightweight Directory Access Protocol* and has written many RFCs and Internet Drafts. He has a bachelor's degree in computer engineering from the University of Michigan. Mark lives in Saline, Michigan, with his wife, Kathy; his daughter, Christina; and his son, Thomas.

Gordon S. Good is a senior software engineer at Opsware, Inc., a provider of IT automation software. Previously Gordon was a senior developer at Netscape Communications Corporation, where he led the directory server replication development team. Prior to joining Netscape, he was instrumental in the development of the University of Michigan's LDAP implementation and in designing and running the university's Web and e-mail services. Gordon has also written several Internet Drafts and RFCs on directories.

Part I: Introduction to Directory Services and LDAP

- [1. Directory Services Overview and History](#)
- [2. Introduction to LDAP](#)
- [3. LDAPv3 Extensions](#)
- [4. Overview of Netscape Directory Server](#)

Chapter 1. Directory Services Overview and History

- What a Directory Is
- What a Directory Can Do for You
- What a Directory Is Not
- The History and Origins of LDAP
- Further Reading
- Looking Ahead

The fact that you picked up this book and started to read it suggests that you have some idea what a directory service is and what it can do for you. This chapter assumes that you have an everyday understanding of directories and expands on that notion to answer three simple but important questions

1. **What is a directory?** In brief, a directory is a specialized database. In this chapter you'll learn what makes a directory specialized, what separates it from a traditional database, what the defining characteristics of a directory are, and why they are important.
2. **What can a directory do for you?** Directories can do many things, and you probably chose this book with a particular set of problems in mind that you'd like a directory to help you solve. We'll take you through the basic uses of a directory, many of which may have already occurred to you, as well as cover some more advanced uses that may be new to you.
3. **What isn't a directory?** The answer to this question is sometimes even more important when you're defining a successful directory environment than when you're learning what a directory is. In this chapter you'll learn what separates a directory from a file system, a Web server, and other things you have deployed on your network. The distinctions drawn here are crucial to narrowing the task of designing your directory service.

This chapter aims to answer each of these questions in detail, formalizing the answers to give you a common understanding of the task before you design a directory service. You'll learn why directories are important, what the scope of a directory solution is, and what directories can do for you. Near the end of the chapter, we introduce the Lightweight Directory Access Protocol, LDAP, by exploring its history and origins. Armed with this knowledge, you'll be ready to read the rest of the book, which deals with the details of understanding, designing, deploying, maintaining, and using your own directory service.

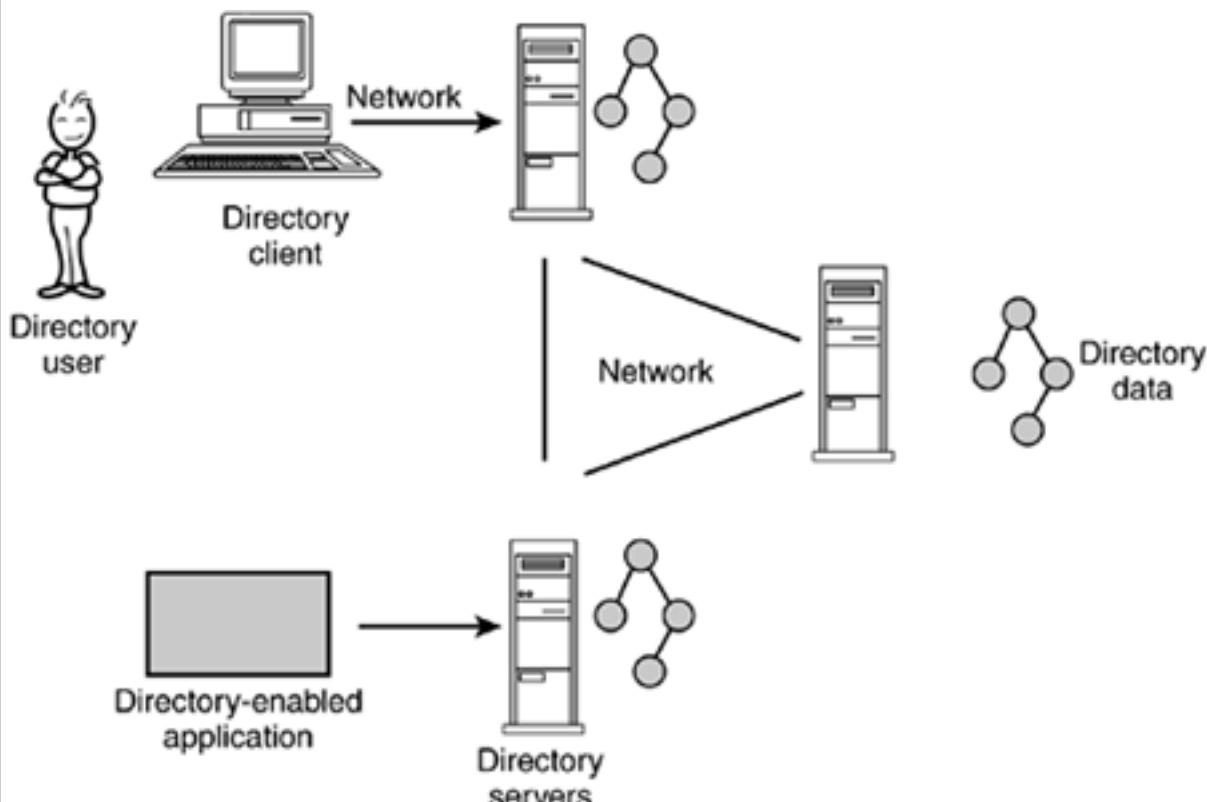
Directory Service Defined

This book uses many terms that may be new to you. A *directory service* is the collection of software, hardware, processes, policies, and administrative procedures involved in making the information in your directory available to the users of your directory. Your directory service includes at least the following components:

- Information contained in the directory
- Software servers holding this information
- Software clients acting on behalf of users or other entities accessing this information
- The hardware on which these clients and servers run
- The supporting software, such as operating systems and device drivers
- The network infrastructure connecting clients to servers and servers to each other
- The policies governing who can access and update the directory, what can be stored in it, and so on
- The procedures by which the directory service is maintained and monitored
- The software used to maintain and monitor the directory service

As you can see, it's quite a list! Some of these components are depicted in [Figure 1.1](#). Generally, we will use the term *directory* as a synonym for *directory service*. It's important to keep in mind that your directory is a sophisticated system of components that work together to provide a service. Concentrating exclusively on one set of components without thinking about the others is sure to lead to trouble.

Figure 1.1. Directory System Components



What a Directory Is

Most people are familiar with various kinds of directories, whether they realize it or not. Directories are part of our everyday lives. Everyday examples of directories we encounter include phone books (white and yellow pages), *TV Guide*, shopping catalogs, library card catalogs, and others. We refer to these directories as *everyday directories*, or sometimes *offline directories*.

With these examples as a guide, it's clear that directories help people find things by describing and organizing the items to be found. Information in such directories ranges from phone numbers to television shows, from consumer goods to reference material, and more.

Directories in the computer and networking world are similar in many ways, but with some important differences. We call these directories *online directories*. Online directories differ from offline directories in the following ways:

- Online directories are *dynamic*.
- Online directories are *flexible*.
- Online directories can be made *secure*.
- Online directories can be *personalized*.

These differences are explored in the following sections.

It's also important to understand the different kinds of directories. We divide directories into the following categories:

- **Application-specific directories.** These come bundled with or embedded into an application. In many cases, you may not think of them as directories at all because their function is so tightly integrated into the application of which they are part. Examples include the IBM/Lotus Notes Name and Address book, the Microsoft Exchange directory, and the `aliases` file used by the sendmail message transfer agent (MTA).
- **Network operating system (NOS)-based directories.** Directories such as Novell's eDirectory (originally named Netware Directory Services, or NDS), Microsoft's Active Directory, Sun Microsystems' Network Information Service (NIS), and Banyan's StreetTalk directory were designed to meet the needs of a NOS.
- **Purpose-specific directories.** These are not tied to an application but are designed for a narrowly defined purpose and are not extensible. Examples are the Internet's Domain Name System (DNS) and centralized Internet phone directories such as Switchboard directory (<http://www.switchboard.com>).
- **General-purpose, standards-based directories.** These are developed to serve the needs of a wide variety of applications. Examples include the LDAP directories we focus on in this book and X.500-based directories.

This chapter makes reference to all four types of directories. Our focus, however, is squarely on the general-purpose type of directory.

Directories Are Dynamic

The everyday directories you are familiar with are relatively static; that is, they do not change often. For example, the phone book is reissued only once a year; you have to call a public directory assistance service such as 411 to get more up-to-date information. A new *TV Guide* is produced every week, but still your favorite show is preempted without notice more often than you'd like. The shopping catalogs you receive in the mail are updated only several times a year, at most; also, they do not contain such useful information as which items are in stock and in which colors and sizes. Why? Because that information changes so often that by the time the catalog got to you, it would be out-of-date.

By contrast, online directories can be kept much more up-to-date. This feature is not always used, of course. Directories are usually only as up-to-date as their administrators choose to keep them. Sometimes administrative procedures are put in place to update the directory automatically. Often online directories are much better if they are their own ultimate authority for the information they hold. As soon as information changes, it can be updated in the directory and made available to users.

It's easy to see how this online update capability can be used to make directories more accurate, resulting in a more useful directory. This kind of improvement is incremental. But online updates have the potential to produce more revolutionary improvements too. These improvements open the door to brand-new directory applications that have no offline analogy.

For example, consider a directory that contains up-to-date information on who's employed at your organization. Such a directory could be consulted by an automated card reader to authorize access to buildings and rooms at your company. In this case, you could revoke access easily and instantly simply by making a change to the directory.

As another example, consider a directory containing location information that is updated as you move from office to office, from hotel room to hotel room, and to other locations. This directory could be consulted to route your phone calls, faxes, and messages to you wherever you are. Traditional paper directories could never be used for such a purpose. The very nature of this application requires frequent updates of the information.

The superior update capacity of online directories not only tends to keep information more up-to-date; it also can be used to distribute the update responsibility. The closer information is to its source, the more accurate and timely the information is likely to be, for at least three reasons:

1. The source of the information is, by definition, the most accurate.
2. Extra delay and opportunity for error between the source and the directory are eliminated if the source makes the update itself.
3. Depending on the information and the application, the source is likely to be the party most motivated to maintain the information correctly.

To illustrate, consider the location directory example described previously. The source is the user (you), and the information is your current location. Who knows better than you where you are? Which path is more accurate for receiving updates: the path directly from you or the one from your administrative assistant (your typing skills notwithstanding)? Suppose that the update came from a directory administrator typing in information reported by your assistant relayed from you? At each step, opportunity for error is introduced, accuracy decreases, and the cost increases as more people are involved. Finally, who is most motivated to have accurate information about you in the directory? Again, it is likely to be you, the source, because you do not get your phone calls, faxes, and mail unless the information is accurate. Of course, this example assumes that you are responsible enough to

want the information to be accurate and that you have the tools and expertise to make it happen.

Directories Are Flexible

Another important difference between static, everyday directories and online directories is that online directories offer far greater flexibility in two areas:

1. The types of information they can store
2. The ways in which that information can be organized and searched

Flexible Content

Offline directories are static in terms of their content. By that we mean that offline directories contain a restricted and seldom extended set of information. For example, if you want to know something beyond the phone number, address, and name information provided by your phone book, you are probably out of luck. But there is a whole host of other useful information that you might like to have. Fax number, mobile phone number, pager number, e-mail address, even a picture or short biographical sketch, to name a few, are all items in the same category as the traditional phone book information. But these items are seldom, if ever, included.

By contrast, online directories can easily be extended with new types of information. The cost of additions like these is huge with printed directories but relatively small with online directories. A printed directory would need to be redesigned, reprinted, and redistributed, at enormous cost. The cost of printing the previous directory cannot be leveraged much at all.

Online directories, however, are typically designed to be extended without a redesign. There is no need for reprinting because changes are reflected automatically and immediately. Nor is there a need to redistribute the directory because clients access the directory online and do not keep their own copy. Some clients may cache or replicate portions of the data, but these copies can be updated automatically.

For simple economic and practical reasons, a printed directory is usually not extended in this way unless a large majority of the directory's users are clamoring for the information. First, as a producer of a printed directory, you could not afford to double or triple the size of your directory to include more information without a compelling reason; doing so could double or triple your cost in producing the directory. Second, from a practical standpoint the directory itself could become unwieldy and inconvenient for the very customers you are trying to serve.

An online directory, on the other hand, can be extended without such costs. Adding a new data item used by only a small proportion of your users suddenly becomes cost-effective. The cost is incremental to the cost of providing the basic service. It may involve only adding some more disk space to your system and marginally increasing backup time, management, and support costs. No inconvenience is experienced by users of your service, however, because they need not even see the additional information. Customers who want the new information can easily get it. An economic incentive exists as well: You could charge extra for these premium directory services.

Flexible Organization

The second way online directories provide more flexibility is in how they let you organize

data. Let's continue with the phone book example. The phone book contains name, phone number, and address information, organized to facilitate searching by name. If you wanted to search by phone number or by address, you would find it difficult, to say the least.

Other specialized directories that are organized to facilitate these kinds of searches may exist, but there is no guarantee of consistency with differently organized directories. Your phone book organized by name might be more or less up-to-date than a special phone book organized by phone number. Such directories contain duplicate information, often leading to inconsistencies and out-of-date information. Also, such directories are usually not readily available, and they are usually expensive. The types of data organization that can be supported are limited. They are also limited by the nature of the medium on which the directories are distributed (for example, paper) and by the capabilities of their end users (people without specialized training, perhaps).

By contrast, online directories can support several kinds of data organization simultaneously. The online analogy to your printed phone book can easily let you search by name, phone number, address, or other information. Furthermore, online directories can provide more advanced types of searches that would be difficult or impossible to provide in printed form.

For example, if you are not sure of the spelling of a name, an online directory can let you search for names that sound like the one you provide. It can also provide searches based on common misspellings, substrings of names, and other variations. These different kinds of searches can be performed simultaneously or in a defined order (for example, an exact search first, then a sounds-like search, then a substring search, and so on) until a match is found. This kind of power in searching is key to providing users with the kind of "do what I mean" behavior they often desire.

Directories Can Be Secure

Offline directories offer little, if any, security. The phone book, for example, is public. Your company's printed internal phone book may have "do not distribute outside the company" stamped on it in big red letters, but this kind of security is advisory at best. Such a lack of security reduces the number of applications that can be served by an offline directory. It also forces users to make difficult choices, if any choice is available to them at all. Most people are familiar with unlisted phone numbers, a service most phone companies offer for a premium fee. Opting out of the directory makes your number less available to telemarketers and other annoying callers, but it also makes your number unavailable to people you probably want to have it.

The root of this problem is the lack of any security in an offline directory. Either its information is accessible to anybody with access to the directory, or information can be left out of the directory and accessible to nobody. This limitation is a natural consequence of the methods used to distribute and access offline directories. Distribution is often wide, and everybody gets a personal copy. The access method consists of flipping through pages or calling a public directory assistance service, such as 411. None of these methods provide any way of determining who is accessing the directory and, therefore, what information they should be able to access.

Online directories can solve these problems. Online directories centralize information, allowing access to that information to be controlled. Clients accessing the directory can be identified through a process called *authentication*. Simply put, authentication is the process by which a directory client establishes an identity with a directory service, typically by providing some credentials, such as a password, that prove the client is who it says it is. In conjunction with access control lists (ACLs) and other information, such as time of day or the IP address of the client, the directory can use the identity established during authentication to make *authorization* decisions about which clients have access to what information in the directory.

Returning to our phone book analogy, consider how security features such as ACLs would change the situation. You could be listed in the directory, but your information would be accessible to only a subset of directory clients. You might be able to specify this subset as a list of friends. You might be able to specify according to certain criteria, such as "anyone who lives on my block." You could allow your information to be available to everyone *except* a list of people you specify. The possibilities go on, and the results are powerful.

It's important to realize that even this level of powerful and flexible security is not a panacea. For example, ACLs can be effectively, if somewhat awkwardly, defeated by a trusted employee who copies confidential information off his or her screen and distributes it outside the company. Still, online directories have security capabilities that are far more advanced than those of offline directories.

Directories Can Be Personalized

Another difference between printed directories and online directories is the degree to which each can be personalized. There are two aspects to this personalization:

1. Personalized delivery of service to users of the directory
2. Personalized treatment of information contained in the directory

TV Guide and the phone book are personalized on a regional basis. But everyone accesses the same card catalog at the library, and everyone probably gets the same L.L.Bean catalog. Furthermore, everyone within the same region gets the same phone book or *TV Guide*. It would be nice to get catalogs tailored to your specific interests, a phone book organized to do searches in the way you prefer, or a card catalog that remembers the kinds of books you like. This is the first aspect of personalization: the capability to deliver information tailored to your needs as an information consumer.

The second aspect of personalization concerns your ability to determine who has access to information about you and other things. This is your ability to tailor the directory to your needs as an information provider. In offline directories, as we saw previously, you have only two broad choices about the accessibility of directory information about yourself: You can either be included in the directory or not—with no in-between. Furthermore, many directories do not even provide this choice. Trying to get yourself unlisted can be frustrating and time-consuming.

Online directories offer both of these features. The mechanism for doing so is rooted in the directory security capabilities described previously. By identifying users who access the directory and storing profile information about them, an online directory can easily provide personalized views of the directory to different users. For example, an online product catalog can show you the types of products that are most likely to interest you. This personalized service could be based on interests that you have explicitly declared. It could also be based on your previous interactions with the service.

From a user's perspective, personalization of this kind is great because it provides a more desirable service. The user does not need to wade through information that is of less interest just to get to the desired information. From a service provider's perspective, personalization of this kind is great because it provides a more desirable service to the provider's users. It also allows the service provider to better target all kinds of special services. For example, the service provider can provide information about promotions and sales, new product offerings, and advertisements, all tailored to a user's preferences. Of course, some users will voice privacy concerns related to what information is collected about them and how it may be used, so savvy providers always provide flexible privacy controls as

part of their directory service.

Directory Described

So far we've been relying primarily on a commonsense understanding of the word *directory* in our discussion. We've used familiar, everyday printed directories to explain what online directories are and how they differ from offline directories. Now it's time to glean from our previous discussion the defining characteristics of online directories. The definition we will give is not formal or mathematical. Instead, we will expound on a list of characteristics that online directories share.

Design Center Defined

The term *design center* refers to the defining set of assumptions, constraints, or criteria driving the design or implementation of a system. When designing or implementing a system, you have to make a series of decisions about what's important, what's not, what the system must do well, and what it can afford to do less well. A system's design center is an expression of the focus the designer or implementer had when making these decisions. Design center is a concept that applies to software and other systems and products as well.

For example, suppose you were going to design and implement a vehicle for yourself. Aside from needing a few common characteristics that essentially boil down to a wheeled, motorized conveyance, you have a lot of flexibility. A designer who has a large family might design a station wagon or van. His design center might be focused on large passenger capacity. Another designer with a lot of stuff to haul around might design a truck. Her design center might be focused on cargo capacity. A designer who is a driving enthusiast might focus on performance and handling.

Software and service design centers work in similar ways, and often the designer considers a whole series of questions to determine the appropriate design center—for example, Does the software system or service need to serve a large community or a small one? Is the community technically sophisticated or inexperienced? Is performance a critical feature of the system? Is security? The answers to these questions and others drive the focus of the design and implementation efforts and ultimately determine the character of the system.

A directory can be thought of as a specialized database. It is interesting to compare databases and directories because the differences have more to do with environment and design center than with anything fundamental. The comparison is also interesting because most people generally have a better understanding of what a database is and does than of what a directory is and does. The differences between a general-purpose database and a general-purpose directory fall into the following broad categories:

- **Read-to-write ratio.** Directories typically have a higher read-to-write ratio than databases.
- **Extensibility.** Directories are typically more easily extended than databases.
- **Distribution scale.** Directory data is usually more widely distributed than data held in databases.

- **Replication scale.** Directories are often replicated on a larger scale than databases.
- **Performance.** Directories have different performance characteristics than databases.
- **Standards.** Support for standards is more important in directories than in databases.
- **Transactions and join.** Directories usually do not support transactions or relational operations such as join.

Each of these points is explained in the remainder of this section.

Read-to-Write Ratio

One defining characteristic of a directory is that it is typically read or searched far more often than it is written or updated. This is often not true for a database. A database might be used to record auditing data that is read only under exceptional, or at least infrequent, conditions. For example, such data might be written thousands of times each day (one record for each database transaction) but read only once a month to produce a summary report, or once a year when an internal audit is conducted.

Information in a directory, on the other hand, is usually read many more times than it is written. In fact, it is not unusual for a piece of directory information to be read 1,000 to 10,000 times more often than it is written. If you think about the types of information usually stored in a directory, this makes sense. Information about people, for example, changes relatively infrequently, especially compared to the number of times the information needs to be accessed. How often do you change phone numbers compared with the number of times somebody calls you? How often do you change addresses compared with the number of times you receive mail?

Data with this "often read, seldom written" characteristic is not restricted to information about people. Catalog data, most location information, configuration information, network routing information, reference information, and many other types of information are all read far more often than they are written. The domain of applications that can be served by a directory is large. For some applications, the information is never updated online; instead, it is updated only periodically via a batch process initiated by an administrator.

Why is this characteristic important? It sets a design center for directory implementations. Implementers can make important, simplifying design decisions based on this characteristic. Directory implementations can be highly optimized for the types of operations that will be performed most often. If directory designers know that read and search operations are performed 10,000 times more often than update operations, they can spend more effort to make those operations perform quickly. In contrast, databases are often designed to support write and read operations equally well. The fact that directories are usually optimized for read-intensive applications has implications for other directory features, such as replication, which we will discuss later in this section.

Information Extensibility

Another important, defining characteristic of a directory is that it supports information extensibility. The term *directory schema* refers to the types of information that can be stored, the rules that information must obey, and the way that information behaves. Schema design for LDAP directories is discussed in detail in [Chapter 8](#), Schema Design.

Directories are not limited to a fixed set schema that can be stored and retrieved. The

schema can be extended in response to new needs and new applications. A directory usually comes with a useful set of predefined types of information that can be stored, but many installations have special requirements that dictate the extension of this predefined set. Your organization may have special fields (attributes, in directory parlance) that you want to store, including, for example, employee status for people or the building location code for a printer. Most directories allow new attributes such as these to be added to existing directory objects without affecting the information that is already present. Although databases are used to store many kinds of information organized in all kinds of ways, they are usually constrained in the types of information that can be stored, and some databases make it difficult to add fields to existing records. It is rare to find a database that allows you to introduce a new, primitive data type with new semantics. Some directories, however, do support adding new primitive data types.

Data Distribution

Distribution of data is another area in which directories differ from databases. Data distribution refers to the placement of information in servers throughout your network. Data can be centralized in a single server, as shown in [Figure 1.2](#), or it can be distributed among several servers, as shown in [Figure 1.3](#).

Figure 1.2. Centralized Directory Data Held in a Single Server

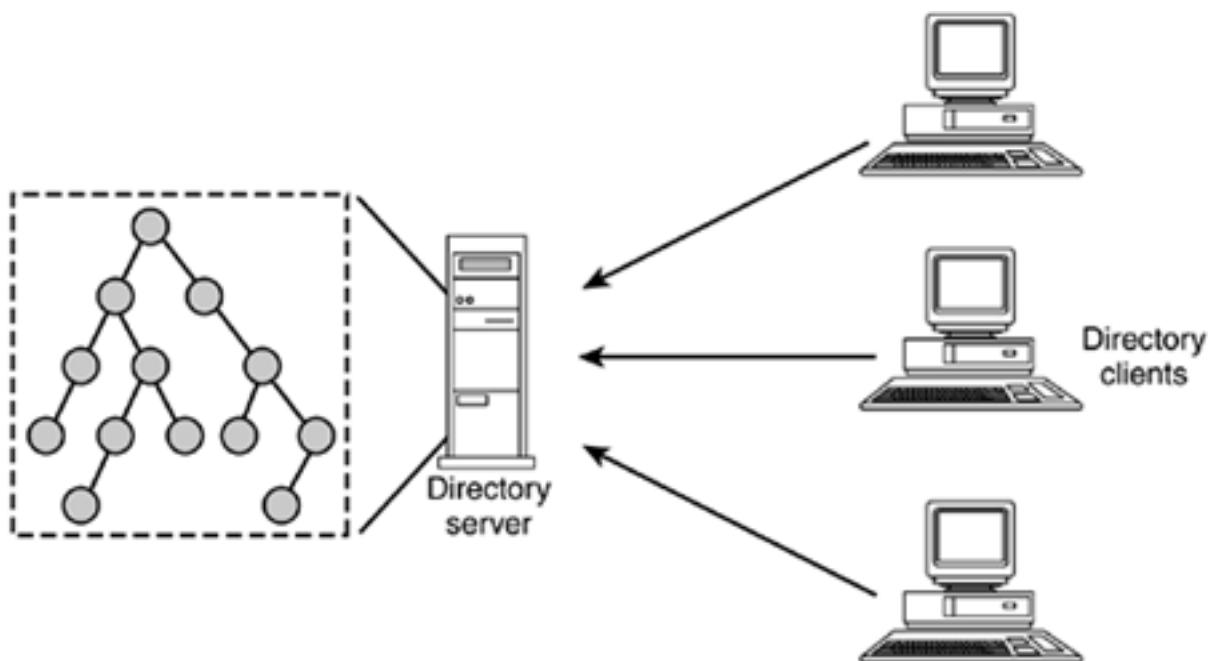
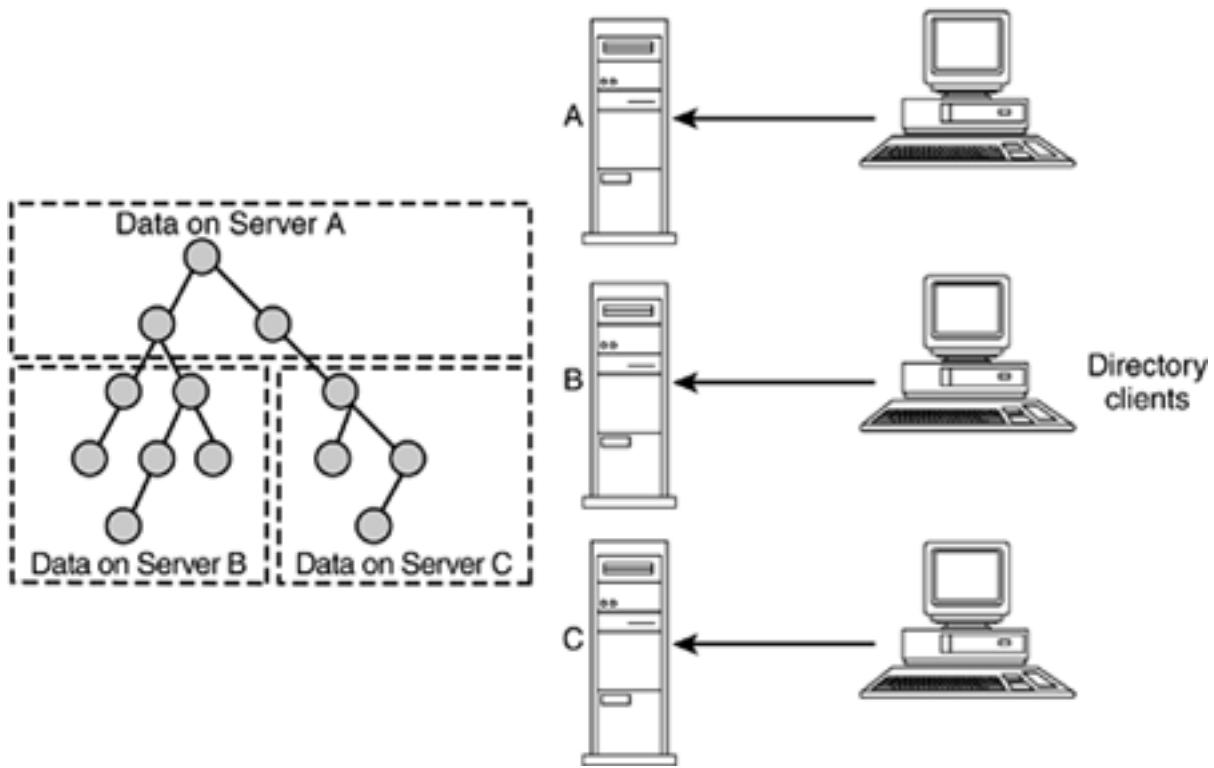


Figure 1.3. Distributed Directory Data Held in Three Servers



Although you can find databases that allow limited distribution of data, the scale of the distribution is different. A typical relational database management system allows you to store one table over here and another table over there. This distribution is usually limited to a few sites. The ability to make queries that involve both sites exists, but performance is often a problem, which causes the distribution features to be used only rarely.

Data distribution is a fundamental factor in the design of directories. Part of the directory's purpose is to allow data to be distributed across different parts of a network. This capability is aimed at addressing environments where authority and administration must be distributed. An example of an organization needing this kind of distribution is one with offices in several countries around the world. Each office wants to have authority over its own directory, but the organization wants to present a single, logical directory to the outside world.

Another example in which data distribution is important is in support of large-scale directories. As your directory data grows, at some point the tactic of buying a bigger server with more disk, memory, and CPU horsepower produces diminishing returns.

A better approach may be to construct your directory from a set of smaller machines that work together to provide the overall service. This solution is cheaper in many cases. It has the advantage of harnessing the parallel processing power of all the machines holding the directory, which can improve both read and write performance. It also has the advantage that failure of one machine does not bring down the entire service. In addition, it has certain attractive practical implications for the performance of some system administration functions, such as performing backups, recovering from disasters, and so on. Consider a directory distributed across ten small machines: Backing up or recovering one of the small machines is easier than backing up or recovering a single large machine.

Distribution of data allows information to be stored near the applications and people that need to use it. For example, consider three applications that need to use directory information: an employee lookup tool (online phone book), a private branch exchange (PBX) that stores the configuration of the phone system in a directory, and a network operating system such as Microsoft Windows NT that stores user profile information in a directory. Through distribution, the data specific to each of these three applications can be stored in a directory server close to the application, thereby improving efficiency and avoiding

unnecessary duplication of data that is private to each application.

Data Replication

Closely related to data distribution is the topic of replication. *Replication* is the process of maintaining multiple copies of directory data at different locations. There are several reasons to do this:

- **Reliability.** If one copy of the directory is down because of a hardware or software failure, other copies can still be accessed.
- **Availability.** Clients are more likely to find an available replica, even if part of the network has failed.
- **Locality.** Latency and variation in performance are reduced if clients are located closer to a directory replica.
- **Performance.** More queries can be handled as additional replicas are added, thereby improving the overall throughput of the directory service.

More detailed information on replication can be found in [Chapter 11](#), Replication Design.

Databases sometimes support replication, but typically they do so for only a small number of replicas, whereas directories typically support dozens of replicas. Historically, performance has been a big problem with database replication implementations, partly because database replication is almost always strongly consistent; that is, all copies of the data must be in sync at all times. Typically, a distributed cross-network locking mechanism must be employed, and a two-phase commit algorithm must be used to achieve strong consistency for database updates.

Directory replication, on the other hand, is almost always loosely consistent. This means that temporary inconsistencies in the data contained in different replicas are acceptable. This characteristic has important implications for the number of replicas that directories can support and the physical distribution of those replicas across the network.

As you will learn later in this section, performance is an important directory characteristic. One good way of helping to ensure great performance is to make sure that each user of the directory has a copy of it close by. There are two reasons to do this:

1. Moving directory data close to the clients accessing it cuts down on the network latency of directory requests, which typically increases overall throughput and improves the consistency of performance for each directory operation.
2. The total number of directory queries processed by the system as a whole can be increased by the addition of replicas. If one directory server can handle 1 million queries per day, adding another server could increase the capacity of the system to 2 million queries per day. The technique of adding more replicas to handle more load is often referred to as *horizontal scalability*.

Availability of the directory is also a key factor. Directories tend to be used by many different applications for such fundamental purposes as authentication, access control, and configuration management. The directory must always be available to these applications if they are to function at all.

Note that availability is not the same thing as reliability. A reliable directory may have redundant hardware and an uninterruptible power source. Such a directory may almost never go down, but that does not mean that it is always available to the clients that need to access it. For example, network segments that connect clients and servers might go down. From the client's perspective, this causes the same problem as the directory hardware or software going down.

You could try to solve this problem by building into your network the same kind of hardware reliability that is available for servers. Redundancy, uninterruptible power, and other techniques are all valuable, although not always practical. The other approach is to replicate your directory data to bring the data closer to the clients needing access to it. This approach helps mitigate network problems that might otherwise prevent clients from accessing the directory. [Figure 1.4](#) shows a sample unreplicated scenario, and [Figure 1.5](#) shows a sample replicated scenario.

Figure 1.4. An Unreplicated Directory Service with Data Held by Only One Server

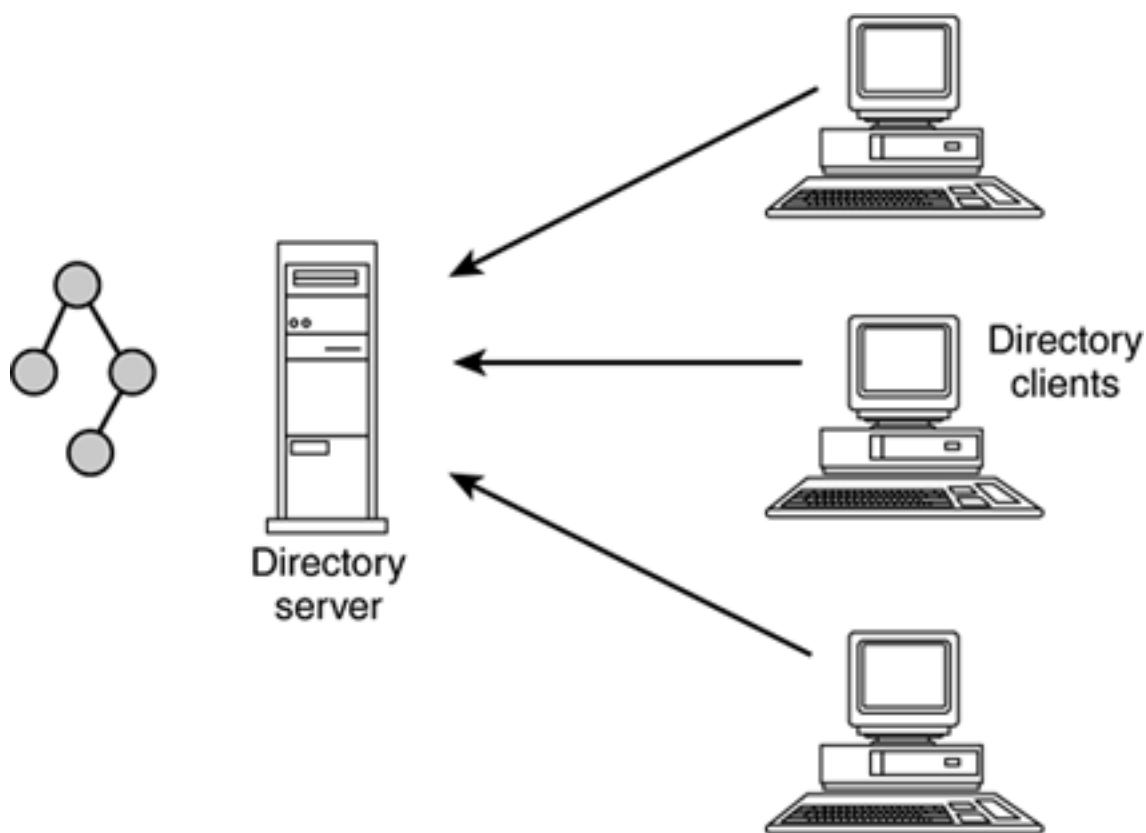
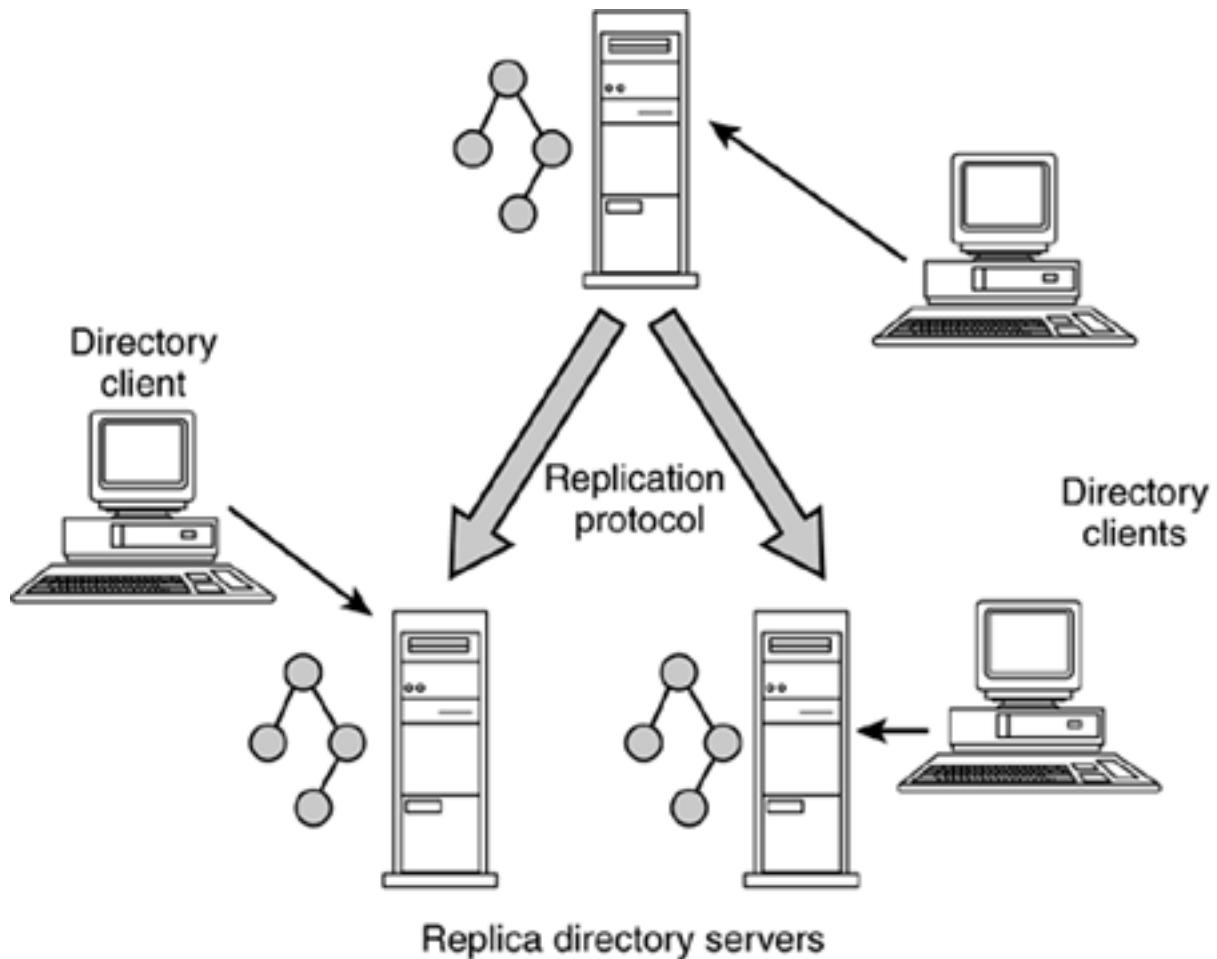


Figure 1.5. A Replicated Directory Service with Data Held by Three Servers



These facts have several implications for directory replication. Directories are replicated on a far greater scale than databases. It is not unusual for a directory replica to be maintained on each subnet in a network to minimize latency and increase availability. In some cases, a replica might be maintained on each machine, potentially leading to literally hundreds or thousands of replicas. These replicas may be many network hops away from the central directory. They may even be connected over links that are up only intermittently. These kinds of replication requirements set directories apart from databases.

Performance

As mentioned previously, high performance is another characteristic that differentiates directories from databases. Database performance is typically measured in terms of the number of transactions that can be handled per second. This is also an important measure of directory performance, but the requirements on a directory service are different from those on most database systems.

A typical large database system might handle hundreds of transactions every second. The aggregate directory performance required by a typical large directory system may be thousands or tens of thousands of queries per second. These queries are usually simpler than the complex transactions handled by databases. As described earlier, the read-to-write ratio is typically much higher on a directory than on a database. Therefore, update performance is not as critical for directories as for databases. As you will learn later in this section, though, it is important nonetheless.

Some of the directory's increased query performance requirements are caused by the wide variety of applications that use the directory. Whereas a database may be designed and deployed with a single or a small set of driving applications in mind, directories are often deployed as an infrastructure component that will be used by an unknown but continually increasing number of applications developed across your company, and even across the

Internet at large. Access to the directory is distributed, as is the development of the applications causing this access. This means that you, as the directory administrator, often do not have control over the kinds of queries your directory must answer. Therefore, it is important that your directory be flexible and capable of good performance regardless of the types of queries it must respond to.

Also driving directory performance requirements are the types of applications that typically access the directory. Applications access the directory for many different purposes. If your directory is used by your e-mail software to route e-mail, for example, one or more directory lookups are required for each piece of mail. Depending on the volume of mail that your site processes, this can be a significant load on the directory.

Many more applications require high performance. If your directory is used by Web application software as an authentication database, it is accessed each time a user launches a new application. If your directory is used by these applications to store user preference and other information needed to provide location independence, even more directory accesses are called for. If your directory is used to store configuration and access control information for your Web, mail, and other servers, the directory must potentially be accessed each time those services are accessed by clients. With a large user population, this quickly adds up to a lot of traffic. In these environments, using directory locality to minimize network latency is critical to providing adequate performance.

As you can see, directories are at the center of a lot of things that quickly increase performance requirements. Of course, client-side caching can and should be used to minimize the number of times the directory itself is accessed, but even these techniques can only slow the flow of directory queries. High performance is still one of the most important characteristics of a directory.

Earlier we stated that the read-to-write ratio for directories is high. The natural conclusion you could draw from this is that write performance is not nearly as important as read and search performance. Although this is true in a way, the scale of data handled by some directories makes write performance an important factor as well. And, as we described earlier, the capacity for online updating is one of the key enablers of some exciting new online directory applications. Clearly, the ability to update is important, and it must function at a certain level of performance.

For example, consider a directory with 1 million entries. This may seem like a lot, but it is not unreasonable for an e-commerce site or for a large corporation (after you're finished adding entries for all users, groups, network devices, external partners, customers, and other things). If each entry changes only once each month on average, that is 1 million updates per month, 250,000 updates per week, almost 36,000 updates per day, or about 1,500 updates per hour. That's quite a few updates! And the peak number of updates that must be handled within a given hour is much higher because user-initiated changes are usually made during business hours. Administrator-initiated changes may need to be saved up and applied in a batch during limited off-peak hours, further affecting performance requirements.

Standards and Interoperability

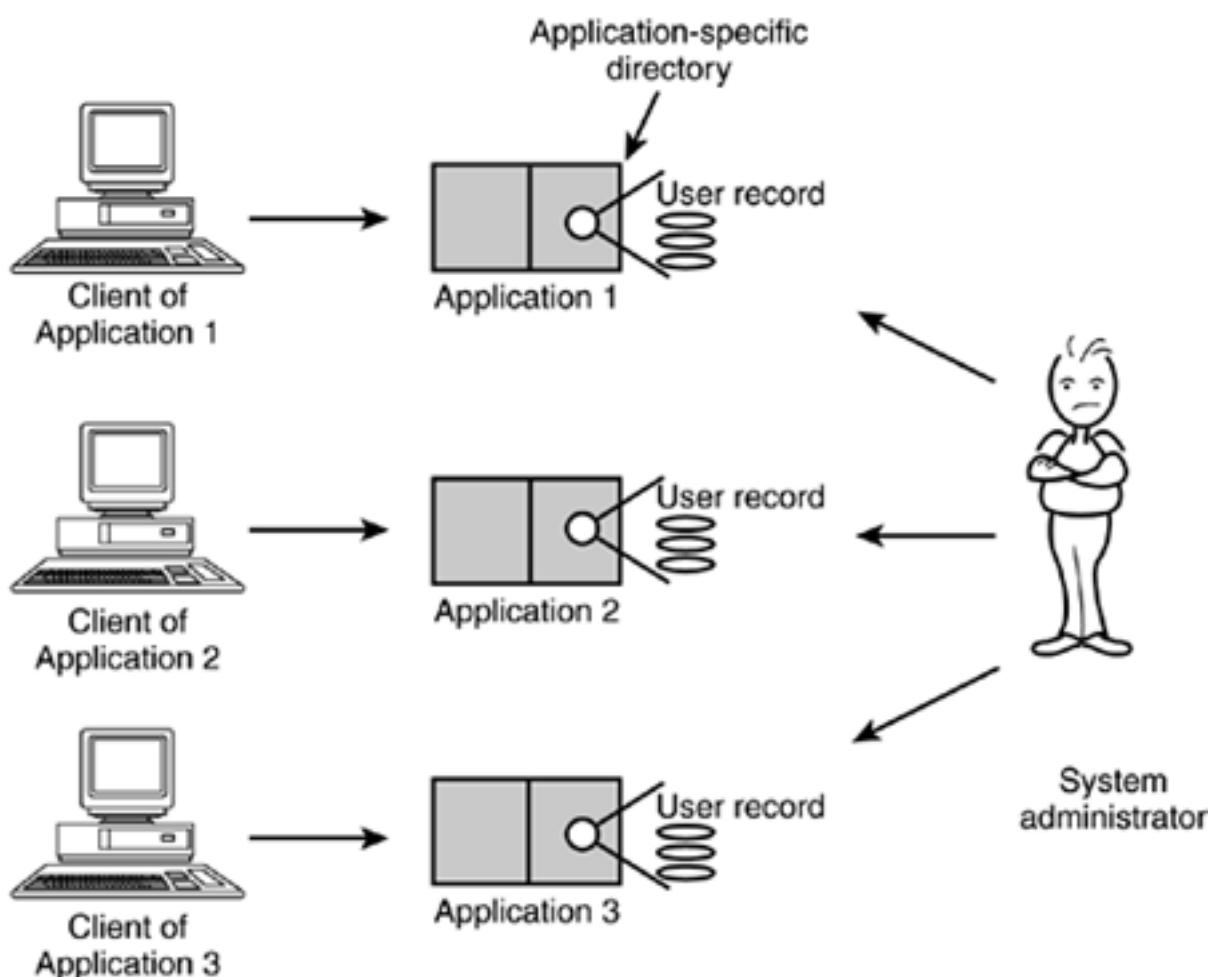
Another important factor that sets general-purpose directories apart from general-purpose databases is standards. The database world has various pseudostandards, from the relational model itself to Structured Query Language (SQL). These pseudostandards make it easier to migrate from one database system to another. They also make it so that when you've learned the concepts behind one vendor's system, you can easily apply that knowledge to come up to speed on another's quickly. However, these standards do not provide real interoperability. In the directory world, because applications from any vendor must be able to use the directory, real interoperability standards are critical.

This is where LDAP, the Lightweight Directory Access Protocol, comes in. LDAP provides the standard models and protocols used in today's modern directories. LDAP makes it possible for a client developed by Microsoft to work with a server developed by Netscape, and vice versa. LDAP also makes it possible for you to develop applications that can be used with any directory. In the database world, an Oracle application cannot be used with an Informix database, and an Informix application cannot be used with a Sybase database. This kind of interoperability, which is lacking in databases, is important to directories for two reasons:

1. It allows the decoupling of directory clients from directory servers.
2. It allows the decoupling of the application development process from a decision about a particular directory vendor.

Before LDAP came along, each application that needed a directory usually came with its own directory built right in. This may seem like a convenient solution at first glance, but the unpleasantness of the situation becomes clear after you've installed your twenty-fourth application, and therefore your twenty-fourth directory. Each user in your organization who requires access to these applications needs an entry in each directory—a lot of duplicate information to maintain. All this duplication is a primary source of headaches for system administrators and increased costs for information technology (IT) organizations, as [Figure 1.6](#) illustrates.

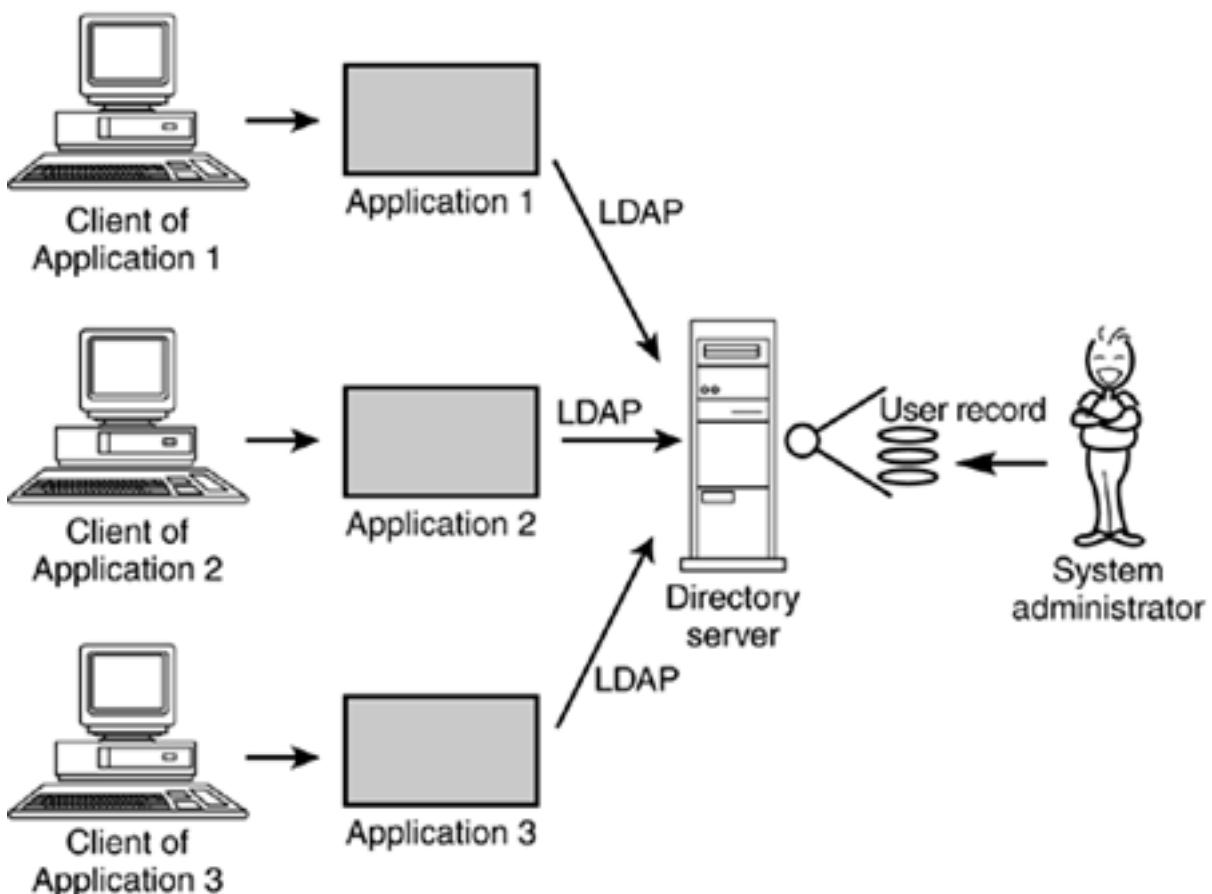
Figure 1.6. Application-Specific Directories Cause Duplicate Information and System Administration Headaches



Application developers everywhere can write applications using the standard directory tools of their choice. These applications will run with any LDAP-compliant directory, which

essentially turns the directory into a piece of network infrastructure. The result is a dramatic increase in the number of applications that can and will be written to take advantage of the directory. In addition, you are freed from having to rely on a single vendor for your directory solution. The same advantages are what drove the success of other Internet protocols, such as HTTP (Hypertext Transfer Protocol) for the Web, IMAP (Internet Message Access Protocol) for accessing e-mail, and even TCP/IP (Transmission Control Protocol/Internet Protocol) itself. [Figure 1.7](#) illustrates a standards-based directory infrastructure.

Figure 1.7. A Standards-Based, General-Purpose Application Directory Eliminates Information Duplication



Transactions and Join

Directories support a relatively simple transactional model. Directory transactions involve only a single operation and a single directory entry. Databases, on the other hand, are typically designed to handle large and diverse transactions that span multiple records and encompass a series of operations. Databases support a feature called *rollback* through which a set of operations that is part of an incomplete transaction can be completely undone, or "rolled back," to restore the original state of the data. Rollback is useful when an error occurs late in the series of operations that make up a transaction. For some kinds of applications, the comprehensive transaction support offered by databases is important. Note that some directory software supports database-style transactions through proprietary extensions to the base LDAP standard.

In addition, most databases in use today are relational databases that support data joins. A *join* is a type of query that brings together related data from multiple sources (data tables) into one result set by leveraging a common key. Most directories do not support joins at all. As an example where a join operation is useful, suppose that you want to search for all printer objects managed by people who work within the Product Development department. Suppose further that each person object has an employee ID field and each printer object has an owner field that also holds an employee ID, and that each person object has a

department field. Then the desired result set can easily be produced by use of a join operation to match the printer objects to the Product Development department via the people entries. In SQL, such a query might look like this:

```
Select *
  From PrinterTable A, EmployeeTable B
 Where A.OwnerID = B.EmployeeID AND Department =
   "Product Development";
```

To produce the same result set using a directory would require several queries. One approach would be to find all people who work within Product Development (one query) and then, for each person found, search for printers that have an owner ID that matches the person's employee ID (many queries). As you can see, the RDBMS (relational database management system) join operation is powerful. Not all applications need it, though.

Directory Description Summary

In concise terms, a directory is a specialized database that is read or searched far more often than it is written to. A directory usually supports the storage of a wide variety of information and provides a mechanism to extend the types of information that can be stored. Directories can be centralized or distributed. They are often distributed in large scale, in terms of both how and where information is distributed. Directories are usually replicated so that they are highly available to the clients accessing them. The scale of directory replication may involve hundreds or thousands of replicas. Replication also helps increase directory performance, which is important to providing applications with a fast, reliable infrastructure component that can be used with confidence. Finally, with LDAP, directories have become standardized. This standardization allows applications and servers from different vendors to be developed, sold, and deployed independently. Directories do not support the sophisticated transactions and the join operations that databases do.

What a Directory Can Do for You

We've already talked about some applications of offline directory services that you deal with every day. We've also talked about how these same applications can be improved, and sometimes revolutionized, by the timely update capabilities, extensible nature, and personalization possibilities of online directories. Now it's time to turn our attention fully to this exciting aspect of online directories: the types of new applications that can be developed to take advantage of all the capabilities of the online directory.

Finding Things

As mentioned at the beginning of this chapter, most everyday directories are aimed at the problem of finding things—for example, the right book in the library or the telephone number of your friend or a business you want to contact; the right style, size, and color of shirt you want to order from a catalog; or the time and channel of the television show you want to watch tonight. Traditional types of directories organize information so that it is easy to find what you are looking for. Finding things is also an important application for online directories. And as you will see, online directories have capabilities that allow far more advanced ways of finding things.

Perhaps the most basic and best-understood application of an online directory is the online analogy to a phone book. Directories in this category start on one end with the large Internetwide or countrywide directories provided by Internet directory service providers, such as Switchboard directory (switchboard.com), Yahoo's People directory (people.yahoo.com), Bigfoot (bigfoot.com), Lycos's WhoWhere? (whowhere.lycos.com), and AT&T's AnyWho (anywho.com). On the other end of the spectrum, you can apply directory technology to create a local phone book for your organization. The service provided by these directories is remarkably similar to the analogous paper directories, with some important differences.

First, online directories let you greatly increase the scale and coverage of the directory. The Bigfoot directory, for example, contains essentially every phone book in North America. Try keeping that many phone books in your house for quick and easy reference! More importantly, online directories allow you to organize information in new and exciting ways, even in an application as simple as a phone book. For example, you can use your directory to search by name, phone number, address, and other information not even contained in a traditional phone book. You can also perform new kinds of searches if you don't know the correct spelling of the name you're looking for, if you think the phone number you have might be off by a digit, or if you have only part of a name. In addition, you can browse the contents of the directory in different ways.

Searching versus Browsing

Directories usually support both searching and browsing, but you must take into account their differences.

In thinking about the offline analogies again, it becomes clear that both searching and browsing are needed. When you're looking for the phone number of Pete's pizza delivery service, you know what you want and you want to go right to the answer. Phone books provide alphabetized listings to make this easy. This is *searching*.

On the other hand, when you're in the mood to buy some new clothes, seldom do you know exactly what you want. You probably have some general characteristics in mind (you know what you like and are in the mood for). It is unlikely that you know the make and model of the shirt you want to buy. You want to flip through the pages of a catalog, looking at pictures of clothes until you come to what you want. This is *browsing*.

Browsing and searching are not mutually exclusive. Often you perform some kind of search to narrow down the choices, which you then browse. Sometimes the reverse is true: You browse some possibilities to find the place where you want to search. The important point to remember is that browsing and searching are complementary. Some applications require one, some the other. And some applications require both, used either together or at different times.

This flexibility of searching and browsing methods is a direct result of the directory's capability to organize information in different ways simultaneously. This kind of flexibility is important in a wide variety of applications. Consider, for example, a network printing application. This application allows users to choose the desired attributes of a printer. The attributes might be location, color or black-and-white, print quality, size or type of paper, single- or double-sided capability, and others. Typical interaction with the user might involve the user's selecting the most important attributes (for example, color and print quality). The user can then choose from a list of printers, perhaps selecting one based on location.

As a final example of finding things, consider a client in search of a service on the network. This service might be a database service, an employee benefit enrollment service, a stock reporting or trading service, an online dating service, a discipline-specific research service, or just about anything else. As long as these services register themselves with the directory, users or other application programs can later query the directory to locate the services. The directory's online update capability allows users to find the services they need even when those services change locations. The service simply updates the directory with its new location.

Managing Things

One application topic that sets online directories apart from their offline counterparts is information management. Online directories provide a big piece of the management puzzle in a distributed environment, providing a central repository for objects that need to be managed. The example cited most often in this category is the expensive task of user and group management—one of the most important management services a directory can provide. A centralized directory can help reduce costs of this service substantially.

Until recently, just about every application you deployed came with its own incompatible directory. Mail applications such as cc:Mail, groupware applications such as Lotus Notes and

Microsoft Exchange, and even earlier versions of Netscape's Web server all came with their own directory. Unix, too, was not immune, with applications such as sendmail using the `/etc/aliases` database. Although this situation is convenient if you're installing only one application, it quickly becomes unmanageable as the number of directories in your environment increases.

Imagine that you have a user needing access to all the applications you have installed on your network. The user must be added to all the application-specific directories. If any information about the user changes, it must be changed in all these directories. This is known as the *N+1 directory problem*; that is, each new application adds one more directory to manage. This is a troublesome and costly problem for administrators everywhere (refer to [Figure 1.6](#)). End users are also inconvenienced; for example, suppose that there are 24 directories and each one stores a copy of a user's password. In the absence of some kind of synchronization solution, changing one password will not affect the other 23.

Consider how this situation changes when you install a centralized directory. In this case you need to enter or change user information in only one place. As new applications are added to the network, they use the centralized directory instead of their own. Passwords are stored and changed in one place. Of course, the flaw in this beautiful plan is clear: All applications must be rewritten to use a centralized directory, at least as an option, rather than their own proprietary directories. These changes are taking place, although perhaps not as fast as you'd like. The improved situation is depicted in [Figure 1.7](#).

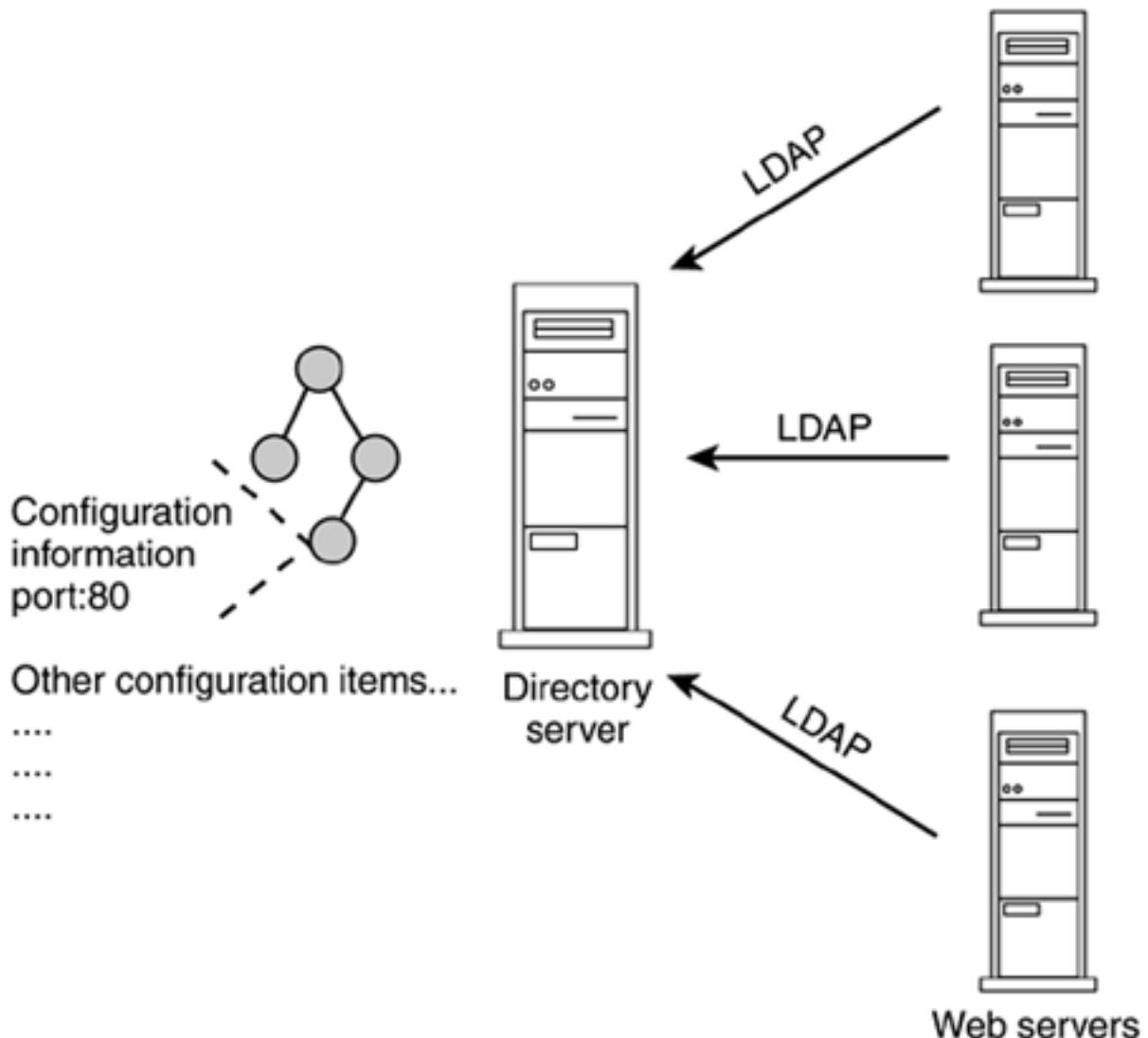
Application developers also benefit from storing user and group information in a centralized directory. Instead of having to develop a proprietary directory service integrated with each application, application developers can use what's already provided and focus on making a better application.

Another important area where directories can help solve the management problem is configuration management. Historically, the configuration for an application has been kept in a set of files or an operating system–specific repository such as the Microsoft Windows Registry. This solution is simple to implement and works just fine for many applications. But network applications often have different needs, especially when deployed in large numbers.

Consider, for example, a network of Web servers. A large corporation may have hundreds of them. If they share one or more configuration items, imagine the onerous task of changing the configuration in all of them. In the configuration file approach, you would need to visit each Web server and edit its configuration file. With hundreds of servers, this task would indeed be tedious. You could develop an ad hoc configuration management system based on tools, such as `rdist` and `rsync`, but the system itself needs to be managed.

Now consider the implications of these servers storing their configuration information in the centralized directory. First, remote management of the server becomes possible; the server's configuration can be accessed from anywhere on the network. Second, the configuration can be shared among servers, making cluster management of servers possible. In our earlier example, the hundreds of Web servers could have their configuration changed quickly and easily from a central location. An example of this scenario is shown in [Figure 1.8](#).

Figure 1.8. Using a Directory for Centralized Configuration Management

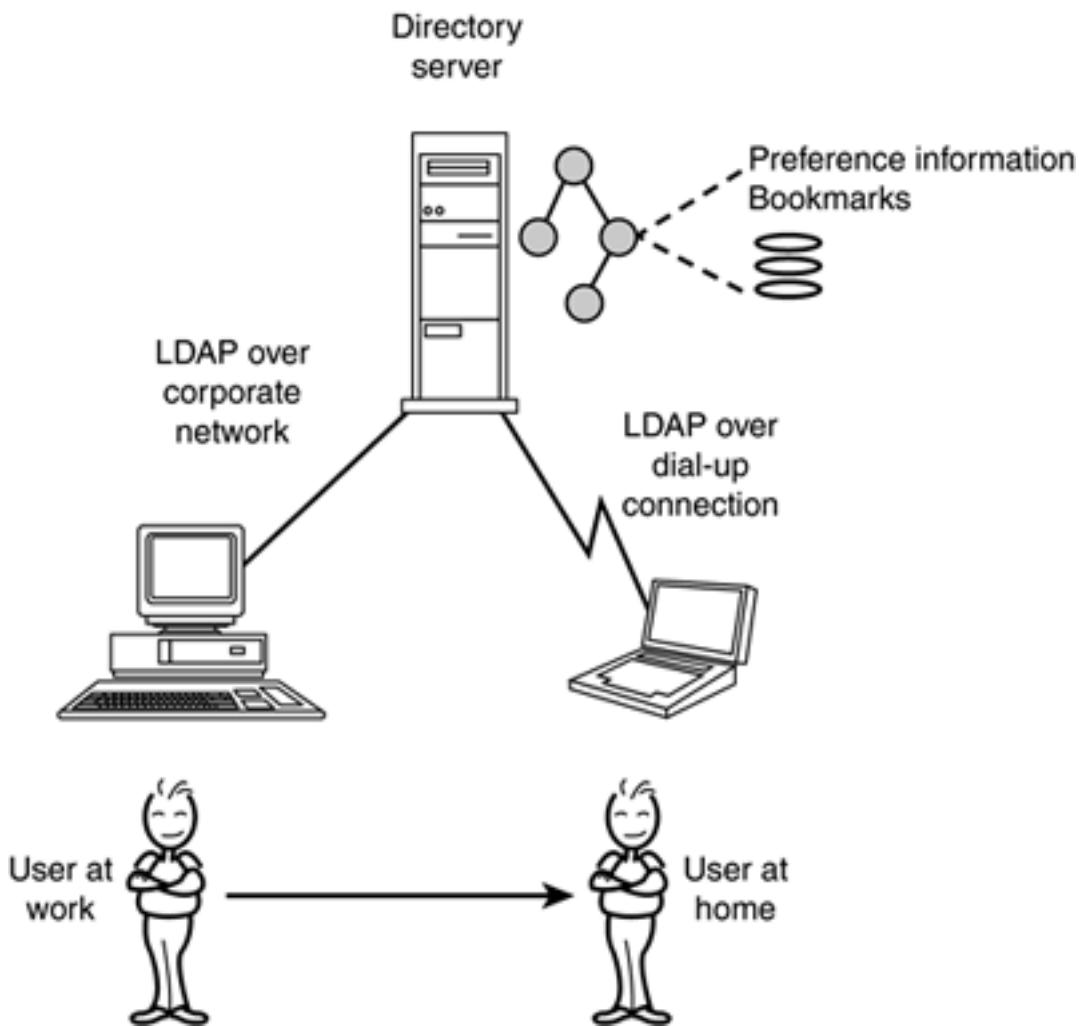


A similar and potentially more powerful application of directory-based management is in the area of user configuration and preferences. As with application configuration, user preferences have historically been maintained in configuration files or local databases, such as the Microsoft Windows Registry, dot files in home directories on Unix, and Macintosh preference files. In an environment where centralized user configuration management is important (and there are many such environments), this proves to be a rather inconvenient solution. However, if applications store and read this information from the directory, thousands of user configurations can be changed at one time. Imagine being able to change a configuration setting one night in one place, and the next day when your users arrive and start their applications, the configuration is seamlessly changed.

When this approach is combined with the storage of user-specific state information in the directory, location independence can be achieved. This is an important feature in many environments. Consider a user who accesses applications from both a machine at home and one at work. The user probably has her preferences set up just the way she prefers. Perhaps she has a personal e-mail address book she needs to access from each location, or personal bookmarks in her Web browser. Storing these things in a directory and retrieving them from the network allows a user to have the same environment both at home and at work.

Similar requirements exist in shop floor situations, in hospitals, or on university campuses, where kiosk access is provided to users who do not have dedicated machines of their own. We use the term *roaming* to describe this type of user; the emerging practice of not providing fixed office space for workers is sometimes called *hoteling*. IS professionals and others who find themselves in front of many different machines in the course of a normal day can also benefit from location independence enabled by a directory. [Figure 1.9](#) illustrates this application.

Figure 1.9. Using a Directory to Provide Location Independence



Lightweight Database Applications

In the first section of this chapter we described a directory by comparing it to a database. In this section we go into more detail about the differences between the two.

Directories are good at storing and retrieving relatively small pieces of information. The fact that LDAP directories provide a standard protocol and a standard API to access this information is also attractive. Prior to LDAP's arrival, application developers who needed to maintain a simple database for their application had just a couple of choices:

- **Build a proprietary special-purpose database.** This approach means extra work for the application developer and the application administrator who must install and maintain the database. It also means a lot of wasted time solving problems that have already been solved.
- **Embed a commercial database package.** This approach is more attractive because it generally involves less work for the application developer and avoids duplication of existing efforts. However, it can still be inconvenient for users and administrators who have to install and maintain the database. Embedding a full-fledged database such as Oracle is overkill and comes with a high price tag. Other simpler, embedded databases exist, but they are usually proprietary and difficult to integrate.

Embedding a simple directory server can often solve the same problems with some distinct advantages. Directories are generally smaller, less complex applications than full-fledged databases. LDAP directories provide a standard access protocol, meaning that your application can work with other directories, and other applications can work with your directory. This approach helps to avoid the N+1 directory problem described earlier.

LDAP directories also provide de facto (soon-to-be-official) standard APIs you can use to access the directory, making your application more portable. The result is a reduction in the amount of work required for application programmers to embed directory services in an application.

One example of an embedded database like this can be found in Netscape's Web browser and mail client. The client maintains several local databases containing user bookmarks, local address book information, security information, and other information. All of these bits of data used to be stored in separate proprietary databases, meaning that the information could not be shared and could not be accessed from anywhere but the local machine. Introducing a directory server to store the information solves both of these problems.

Another example of an embedded database is found in the Netscape Certificate Management System. This server maintains a database of certificates it has issued to users. Version 1.0 of the product stored these certificates in an Informix database. This approach proved to be unwieldy, difficult to manage and install, and generally a poor choice. Now the certificate server uses an embedded LDAP directory server to manage this database of information. This directory-based solution is much easier to install and manage, easier to develop with, and much more flexible.

Security Applications

Another interesting directory application is security. Of particular interest are public key-based security systems. [Chapter 12](#), Privacy and Security Design, covers these systems in more detail. For now, it's enough to know that the public key infrastructure (PKI) required to provide security like this is difficult to manage. PKI requires a way to distribute security objects such as certificates to clients and servers, to keep those security objects up-to-date, to revoke them when they are compromised, and to handle other functions. Without a directory, these tasks are difficult.

Directories help solve two problems that certificates and PKI in general introduce:

1. **The PKI life cycle management problem.** This problem has to do with how certificates are created, maintained, and destroyed. Without a directory, this process is up to you to manage. If you're lucky, you might have some proprietary software acting on your behalf that helps manage the process for you.
2. **The certificate location problem.** This problem has to do with how you find the certificates of the people with whom you want to communicate securely, and how people who want to communicate with you find your certificate. Without a directory, you and your friends might have to resort to calling each other on the phone and reading off strings of hexadecimal (base 16) digits to each other.

Introducing a directory helps solve these two problems. The directory acts as a central point of administration throughout the PKI life cycle. Your various security objects can even live in the directory, where you or an administrator you trust can maintain them. When you need a new certificate, one can easily be issued to you through the directory. When your certificate needs to be revoked, again the directory provides the means to do so quickly and efficiently, along with the means by which other parties can be notified of the change.

The directory also helps locate the certificates of others. In this application, the certificate or other security information is just another attribute of a user's directory entry. A certificate can be retrieved from the directory as conveniently and efficiently as a name, phone number, or e-mail address. A more detailed and formal description of these problems and processes is presented in [Chapter 12](#).

What a Directory Is Not

In our experience, people sometimes have one of two extreme reactions upon learning about directories. One reaction is negative: "What good is this directory really going to do? Why can't I just use X to fill that same need? I don't see how this directory is going to be as reliable and perform as well as you say, and I am hesitant to make my application depend on it!" This reaction can usually be overcome by additional education, helping the skeptic see a directory in action, and better explaining all the great things a directory can do.

The other reaction is equally misguided, but it can be more difficult to overcome: "This directory is great! I bet there's nothing it can't do. I can now use the directory instead of my file system, Web server, FTP server, and a host of other things. Finally, a handy, all-in-one tool that chops, slices, dices, makes homemade desserts, but best of all, cleans itself!" The enthusiasm of this reaction is to be admired, but there is danger here too. An old saying states that if the only tool you have is a hammer, every problem begins to look like a nail. Needless to say, not all problems are nails, and trying to treat them as such leads to poor carpentry.

A directory is no different; like any other technology, it is best suited to solving one set of problems. A directory can be usefully applied to many other problems as well, though with somewhat less satisfaction. You can abuse a directory by trying to make it solve problems that it was not intended to solve.

In this section we examine this area more closely, explaining some of the many applications for which directories are *not* well suited. We will explain why directories are different from the following network services that may look similar on the surface:

- Databases
- File systems
- Web servers
- FTP servers
- DNS servers

We will explain how your directory can complement all these services and why each one fills a valuable niche in your computing infrastructure. We will conclude this section with a summary of when to use a directory to store information and when to use something else.

Directories versus Databases

In the first section of this chapter we described what a directory is, and we compared directories to databases. We will not repeat that comparison here, but we will highlight a few important points. First, recall that directories are typically read-focused rather than write-focused. So if your application is writing large volumes of data—say, for recording merchandise transactions—you should choose a database over a directory.

Second, directories support a relatively simple transaction model. Directory transactions involve only a single operation and a single directory entry. Databases, on the other hand, are designed to handle large and diverse transactions, spanning multiple data items and many operations. Databases also support join operations that allow complex result sets to be produced efficiently. If your application requires this kind of support, again a database is a better answer than a directory. On the other hand, if you have an application that does not have these requirements but instead wants to read and occasionally write information to a network-accessible database in simple transactions, a directory server may well be a good, cost-effective, much simpler choice than something like an Oracle relational database.

Directories versus File Systems

A directory makes a poor file system. Files have characteristics different from directory information. Files are often large, containing many megabytes or even gigabytes of information, whereas directories are optimized for storing and retrieving relatively small pieces of information.

Although there is often no restriction on the size of a directory entry or attribute, the question is one of design center. Some files are written far more often than they are read, such as log files and files used to hold a database. Directories, as you'll recall, are optimized for read over write. Some files, of course, are read far more often than they are written. Application binaries are a good example of files in this category, but the size of these files is often larger than should be stored in a directory.

Applications often access a file in chunks, especially if the file is large. File system APIs provide functions for this very purpose, such as `seek()`, `read()`, and `write()`, which can be used to access only a portion of a larger file. Directories do not provide support for this kind of random access. Instead, a directory entry is split up into data items called *attributes*—for example, telephone number and name. You can retrieve each attribute separately, but you usually have to retrieve all of any attribute you ask for. Unlike a file system, there is no way to retrieve only part of an attribute, starting at a particular byte offset. File systems, on the other hand, are not good at storing attribute-based information, and they do not typically have the general-purpose search capabilities that directories have.

Directories versus Web Servers

Since the World Wide Web burst onto the computing scene in the mid-1990s, Web servers have become ubiquitous. Chances are good that your organization, depending on its size, runs anywhere from one to several hundred Web servers. Web servers have certain similarities to file systems. They are made to serve clients accessing documents or files that can vary greatly in size. Although Web server documents usually share the typical "read many, write few" characteristic of directory information, directories are not well suited to the task of delivering to clients multimegabyte JPEG images or Java applications.

Web servers also often serve as a springboard to a development platform for Web applications. These platforms range from simple CGI (Common Gateway Interface) to more complex platforms, such as the one found in BEA's WebLogic Application Server or Sun's ONE Application Server. Directories typically do not provide this kind of flexibility in application development, although some directory implementations, such as the Netscape Directory Server, do provide a platform and a set of services that can be used for directory application development.

Directories are optimized for providing sophisticated searching of the data they hold. Web servers can be used to develop similar search applications, but such applications are not based on standards. Web servers are tuned to providing GUI-style interfaces on applications; they are not tuned to providing generic application access to directory data. If you have a specific database of information you want to make available to users, a Web server might be a good choice. If you have information you want to make available to a wide variety of applications, a directory server is a better choice.

Directories versus FTP Servers

An argument similar to the previous one for Web servers applies to File Transfer Protocol (FTP) servers. One could argue that the days of FTP servers are numbered now that Web

servers are so ubiquitous, but they are not threatened by the arrival of directory services. Again, the main differentiating factor is the size of the data and the type of client that needs access to it. Another important point is that FTP is a simple protocol, tuned to do one thing and do it well. If all you want to do is create a means to transfer files from one place to another, the extra directory infrastructure needed to perform replication, searching, updating, and so on is overkill.

On the other hand, if your application involves more than simple retrieval and storage of information, a directory is a more appropriate choice. Unlike directories, FTP provides no search capability, no attribute-based information model, and no incremental update capability.

Directories versus DNS Servers

DNS, the Internet's Domain Name System, translates host names, such as [home.netscape.com](#), into IP addresses. Host names are good for users who want to remember how to connect to their favorite Internet service. IP addresses are required by the Internet networking infrastructure that is responsible for making the user's connection happen. The DNS and a typical directory have certain similarities, such as providing access to a hierarchical distributed database. But there are some important differences that set the two apart.

The DNS is highly optimized for its main purpose; most directories are meant to be more general-purpose. The DNS has a specialized, fixed set of schemas; directories allow schemas to be extended. DNS servers typically do not allow update of their information; directories do. The DNS can be accessed over a connectionless transport; directories are usually accessed through connection-based transport.

Note

The Internet Engineering Task Force (IETF) has developed standards to add update capabilities to the DNS, and systems such as Microsoft Windows 2000/XP and recent versions of the Berkeley Internet Name Domain (BIND) support dynamic DNS updates. Proposed IETF work on connectionless LDAP is aimed at providing LDAP access over User Datagram Protocol (UDP) in addition to TCP. These efforts and others may eventually bring DNS and LDAP closer together. But for now, the best argument for not trying to replace the DNS with LDAP is that the DNS is working just fine, and it would be disruptive and expensive to rewrite all Internet applications to use anything other than DNS for mapping from host name to IP address.

The Complementary Directory

Directories share certain similarities with many of the services just listed. Directories tend to complement most of these services. We will now take a moment to explore this notion of a complementary directory, what it means, and how it can create synergy among all the services in your enterprise.

A good example of the complementary directory is how it relates to a Web server. Here the directory has some important supporting roles to play.

First, the directory can serve as an authentication database. When clients authenticate to

the Web server, their credentials (for example, a user ID and password or a certificate) are checked against the directory. When the Web server needs to make an access control decision, the directory can again be consulted to determine group membership and other information pertinent to the decision. The value of the directory in this role is especially apparent when you consider an environment that is running many Web servers, all of which need access to the same authentication database. By sharing a directory, the Web servers reduce the user management problem to a manageable level. Other services can benefit from this type of use as well.

Second, the directory can serve as a network-accessible storage device for information about configuration, access control, user preferences and profiles, and other things. The value of the directory in this role is twofold:

1. If any of this information is to be shared across servers, the directory can act as a central repository, eliminating redundant administration much as it does for user and group information. As mentioned earlier in this chapter, it is much more convenient to change a shared configuration item on 100 Web servers via a single update to the directory. The alternatives are to visit each Web server and make the change redundantly or to maintain a separate, ad hoc system to manage configuration.
2. The directory can provide a standard, network-accessible way to administer all this information. This opens up a whole new set of possibilities for standardized management tools and common administration frameworks.

As a third example of its complementary nature, the directory can be used to help organize and access information contained in the Web server itself. Today, Web search engines exist to catalog and organize Web-based content. But, as anyone who has spent much time using these services can attest, you often spend more time wading through irrelevant matches to your query than reading the actual information desired. The problem is simply that Web content lacks structure and therefore is difficult to organize and search in an automated way. With the advent of XML (eXtensible Markup Language), this lack of structure is beginning to change. This is where directories come in.

Directories are great at organizing and providing subsequent access to information. Imagine today's Web search engines driven by directories: If you used the directory query language and typed information structure to specify the information you were looking for more accurately, the directory could return a much more focused set of results. Keep in mind that in this scenario the content itself is still stored in a Web server and that free-text searches are still conducted as they are today. The directory's value is to provide some structure on top of this arrangement and also to provide a precise, yet flexible, query mechanism.

Another good example of a directory complementing existing services is found in the examination of file systems and FTP servers. In this case a directory can be used not to hold the contents of files, but rather to hold metainformation about those files, their locations, who owns them, and other things that might be useful in locating them. Most importantly, the directory can hold the information that a file system or FTP client needs to access files contained in that service. An FTP server could also use a directory server for authentication purposes.

This idea of using directories to organize and search for information that you want to access is a common theme. Directories often don't hold the content you seek, but they can hold the location of that information along with other attributes that can help you find what you're looking for.

The dividing line between what should be held in a directory and what should be held elsewhere is not always clear. General guidelines are that the larger a piece of information is, the less likely it should be put in a directory. The more frequently the information

changes, the less likely it should be maintained in a directory. The less structured a piece of information is, the less benefit you will likely derive from placing it in a directory. However, the more often a piece of information is shared, the more benefit you will likely derive from placing it in a directory.

Here's a brief summary of things to think about when deciding whether to use a directory or another piece of technology for storing information:

- **Size of the information.** Directories are best at storing relatively small pieces of information, not multimegabyte files. Directories are good at storing pointers to large things, but not the large things themselves.
- **Character of the information.** Directories typically have an attribute-based information model where information is broken up into a set of name–value pairs (see [Chapter 2](#), Introduction to LDAP, for a detailed discussion of the LDAP information model). If you can express your information naturally in this form, a directory might be a good choice. If you can't, consider using a database, file system, or other approach.
- **Read-to-write ratio.** Directories are best for information that is read far more often than it is written. If the information is to be written more frequently, a database or file system might be a more appropriate choice.
- **Search capability.** Directories are made to search the information they contain. If your application has this requirement, a directory might be a good choice.
- **Standards-based access.** If you need standards-based access to your information, a directory is a good choice.

Keeping these principles in mind when you're deciding what storage mechanism to use for your application will keep you out of trouble. By this time, you should have a good understanding of what a directory is, what a directory is not, and how a directory relates to other services in your network. Hopefully this knowledge will help you avoid the "hammer syndrome," in which every problem looks like a nail that you can solve using your directory hammer.

The History and Origins of LDAP

In the first part of this chapter we introduced the notion of online directories and made comparisons to a variety of other network-based services and protocols. Now that we have discussed the capabilities of directories in depth, it is time to discuss LDAP and the origin of general-purpose, standards-based online directories.

The Dawn of Standard Directories: X.500

It is impossible to discuss the origins of LDAP without first talking about X.500. In the mid-1980s, two separate standards bodies independently started work on directory services that could work across system, corporate, and international boundaries. The first body was the International Telegraph and Telephone Consultative Committee (CCITT), which later changed its name to the International Telecommunication Union (ITU). CCITT member organizations (mainly telephone companies) wanted to create a white pages directory that could be used to look up telephone numbers and electronic mail addresses.

The other standards body was the International Organization for Standardization (ISO), which wanted a directory to serve as a name service for Open Systems Interconnection (OSI) networks and applications. (A *name service* provides information about network objects; the DNS is a familiar example.)

Eventually, the two independent directory specification efforts merged into one effort, and X.500 was born. The first X.500 standard was approved in late 1988 and published in early 1990 by CCITT; it was subsequently updated in 1993, 1997, and 2001 (work continues today). The specification itself references other ISO Standards and consists of a series of recommendations, including the following:

- **X.501: The Models.** Describes the concepts and models that underlie an X.500 directory service.
- **X.509: Authentication Framework.** Describes in detail how the authentication of directory clients and servers is handled in X.500.
- **X.511: Abstract Service Definition.** Describes in detail the functional services provided by X.500 directories (for example, search operations, modify operations, and so on).
- **X.518: Procedures for Distributed Operation.** Describes how directory operations that span multiple servers are handled, among other details.
- **X.519: Protocol Specifications.** Describes all the X.500 protocols, including Directory Access Protocol (DAP), Directory System Protocol (DSP), Directory Operational Binding Protocol (DOP), and Directory Information Shadowing Protocol (DISP).
- **X.520: Selected Attribute Types.** Defines attribute types used by X.500 itself, and some that are generally useful as well (such as the `telephoneNumber` attribute).
- **X.521: Selected Object Classes.** Defines object classes used by X.500 itself, and some that are generally useful as well (such as the `person` object class).

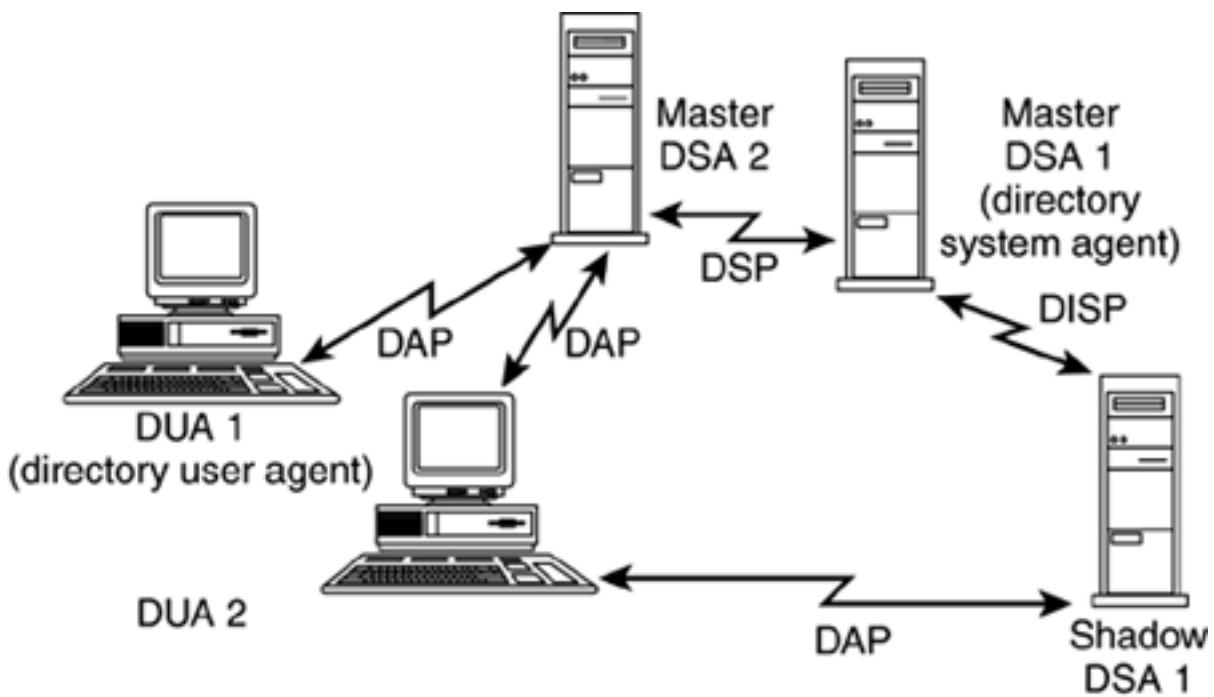
- **X.525: Replication.** Describes how directory content is replicated among X.500 servers.
- **X.530: Systems Management.** Describes how an X.500 directory service can be managed through the use of OSI systems management services and protocols, directory services and protocols, and local means.

As you can see, X.500 developers must absorb quite a few documents before they can begin to develop X.500 directory services software! The X.500 documents are published both online and in paper form by the ITU, which currently charges money to access them.

X.500 Innovations

At the time it was originally defined, X.500 had many novel qualities. First, it was one of the first truly general-purpose directory systems, and it was designed from the beginning to be extensible to serve the needs of a great variety of applications. Second, X.500 provided a rich search operation that supported many different kinds of queries. Third, X.500 was designed to be a highly distributed system in which the servers, data, and administrators could span the globe. Finally, X.500 was an open standard not controlled by any one vendor or tied to any specific operating system, networking technology, or application. The dream of the X.500 designers was that there would eventually be one fully interconnected global directory service: *the Directory*. [Figure 1.10](#) shows the major components of an X.500 directory service.

Figure 1.10. Components of an X.500 Directory Service



X.500 directories rely on a large suite of protocols, including DAP, DSP, DOP, and DISP. Some of X.500's strengths are its information model (which is flexible and complete), its versatility, and its openness. As you will see, the LDAP designers adopted many of the best ideas from X.500 while removing unneeded complexity.

X.500 Flaws

In practice, X.500 directories suffered from some significant flaws, especially in their first

decade of existence. The early implementations were buggy and did not perform or scale well. Those who tried to deploy distributed X.500 directories discovered some significant barriers to adoption, many of which were not specific to X.500 (such as difficulty obtaining and maintaining good data in a directory service). In addition, because of the complexity of the standards themselves and the associated difficulty of implementation, it has taken a long time for interoperability between different X.500 implementations to become possible.

Furthermore, X.500 was based on the OSI network protocols. The dream of the OSI designers was that OSI would eventually replace TCP/IP as the networking protocol suite of the future. Unfortunately for them, this never happened. The simplicity, speed, and low cost of TCP/IP proved to be an unbeatable combination. Recently, in fact, the proponents of X.500 have defined a mapping that allows X.500 to run directly over TCP/IP.

Finally, the origins of X.500 itself have to some extent prevented it from succeeding. The Internet grew rapidly in a bottom-up fashion as independent organizations deployed hosts and services at their own pace and without a lot of reliance on others. In contrast, X.500 was designed with the large public service providers in mind, implying a top-down deployment. Top-down deployment has proved to be impossible to achieve—and it is certainly at odds with the prevailing Internet culture.

Early X.500 Implementations and Pilots

One of the best-known early X.500 implementations, called Quipu, was developed at University College, London. Its name refers to the complicated system of knot-tying that the Incan culture used for communication. Part of the reason for Quipu's early popularity was that it was freely available in source code form. Quipu is built on top of an OSI networking package called the ISO Development Environment (ISODE), which allows OSI protocols to run on top of TCP/IP and, thus, the Internet.

The availability of Quipu encouraged many organizations to experiment with X.500 directories. The Quipu implementation served as the basis for several European and United States-based pilot projects for X.500 white pages applications. Unfortunately, these worldwide X.500 directory pilots didn't grow as fast as their sponsors hoped, and interest has waned.

X.500 found its greatest success in large organizations that could afford the time and effort needed to deploy a large, complex directory service. Some early deployers of X.500 included the United Kingdom academic community, Boeing Corporation, the National Aeronautics and Space Administration (NASA), the University of Texas, and the University of Michigan. X.500 products are still available today, although all of them are accessible through the use of LDAP in addition to the X.500 protocols. Typically, an X.500 directory service is deployed in partnership with the X.500 software vendor. Most vendors provide extensive consulting services to help organizations deploy and integrate an X.500 directory into their existing environment.

The Creation and Rise of LDAP

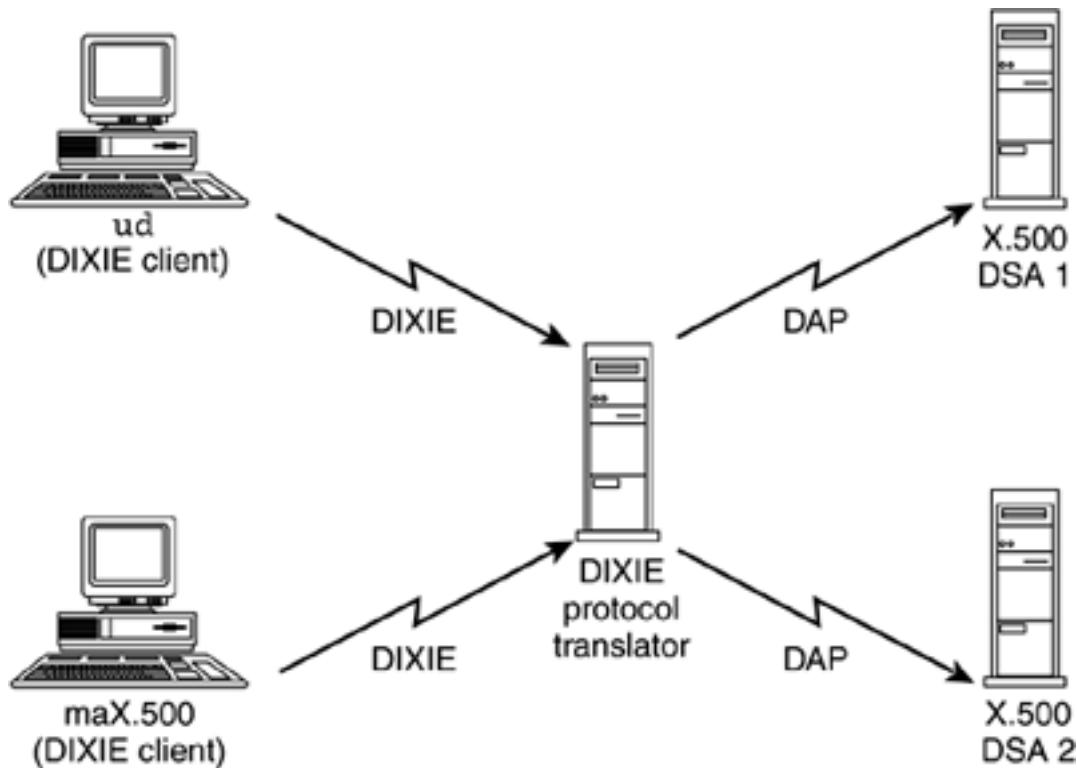
The early adopters of X.500 found that DAP (X.500's directory client access protocol) was fairly complex and not well suited for or available on the desktop computers of the day (PCs and Macintoshes). DAP is large, complex, and difficult to implement, and most implementations perform poorly. Because most potential directory users had ordinary machines on their desks, the people deploying X.500 began to look for an approach that avoided the heavyweight DAP.

Forerunners of LDAP: DIXIE and DAS

In about 1990, two independent groups devised similar protocols that, compared to DAP, were simpler and easier for desktop computers to implement. Desktop clients used one of these new protocols to communicate with an intermediate server directly over TCP/IP, and the intermediate server in turn implemented X.500 DAP. Both groups brought their work to the IETF.

The two lightweight protocols for desktop computers were called Directory Assistance Service (DAS), defined in RFC 1202; and Directory Interface to X.500 Implemented Efficiently (DIXIE), defined in RFC 1249. [Figure 1.11](#) shows the architecture of a directory system that uses a DIXIE client/client/server architecture.

Figure 1.11. DIXIE Provides a Front End to an X.500 Directory



Both protocols were successful; DIXIE, though, was quickly adopted as the preferred method to access X.500 directory systems. However, one shortcoming of the DAS and DIXIE protocols is that both were closely tied to a single X.500 implementation (Quipu).

The Creation of LDAP

After DIXIE and DAS showed the utility of a lighter-weight access protocol for X.500, the members of the OSI-DS Working Group within the IETF decided to join forces and produce a full-featured, lightweight directory access protocol for X.500 directories. Thus, LDAP was born.

The developers of the first LDAP specification were Wengyik Yeong, Steve Kille, Colin Robbins, and Tim Howes, who is one of the authors of this book. Note also that contributions were made by many people from the Internet and X.500 communities. The first LDAP specification was published as RFC 1487 in July 1993. The first version of LDAP to see widespread use was version 2 (LDAPv2); the final specification for LDAPv2 was published as RFC 1777.

LDAP Innovations

The LDAP developers simplified the heavyweight X.500 DAP protocol in four important areas:

1. **Functionality.** LDAP provides most of DAP's functionality at a much lower cost. Redundant operations and rarely used features of DAP were eliminated, simplifying the implementation of LDAP clients and servers.

The IETF

The Internet Engineering Task Force (IETF) is a large, open, international group of network researchers, designers, and operators who work on evolving the Internet architecture and enhancing its capabilities. The IETF was formally chartered in 1986, although it existed informally for some time before that. The IETF is focused primarily on protocol development and engineering for the Internet. The group is open to any interested individual; there are no membership requirements, and only a small fee is collected to cover the cost of meetings. The IETF holds face-to-face meetings three times each year.

The technical work is done primarily within IETF working groups (although individual or vendor contributions are also accepted). The working groups have one or more chairpersons, and the group members meet three times each year at the regularly scheduled IETF meetings. However, most of the work happens outside the meetings on electronic mailing lists that each working group maintains. The working groups are organized into several large areas (routing, security, transport, applications, and so on). An early directory working group was the Open System Interconnection-Directory Services (OSI-DS) Working Group, which initially focused on the use of X.500-based directories on the Internet.

Specifications produced by IETF members are first published in draft documents called, logically enough, *Internet Drafts*. When consensus is reached in a working group and a series of technical and administrative requirements are adequately met, documents may be published as Requests for Comments (RFCs). All IETF documents are freely available at no cost on the Internet. Note that there are several categories of RFCs, ranging from Informational to Experimental to Standards Track, and that few RFCs are intended to become official Internet Standards. As of early 2003, more than 3,400 RFCs had been published, but fewer than 70 had reached full Internet Standard status.

The first IETF meeting, held in San Diego in January 1986, was attended by 21 network engineers. IETF meetings now have upwards of 2,000 attendees. As of early 2003, more than 140 active working groups were organized into 8 areas within the IETF. More information about the IETF can be found on its World Wide Web site at www.ietf.org.

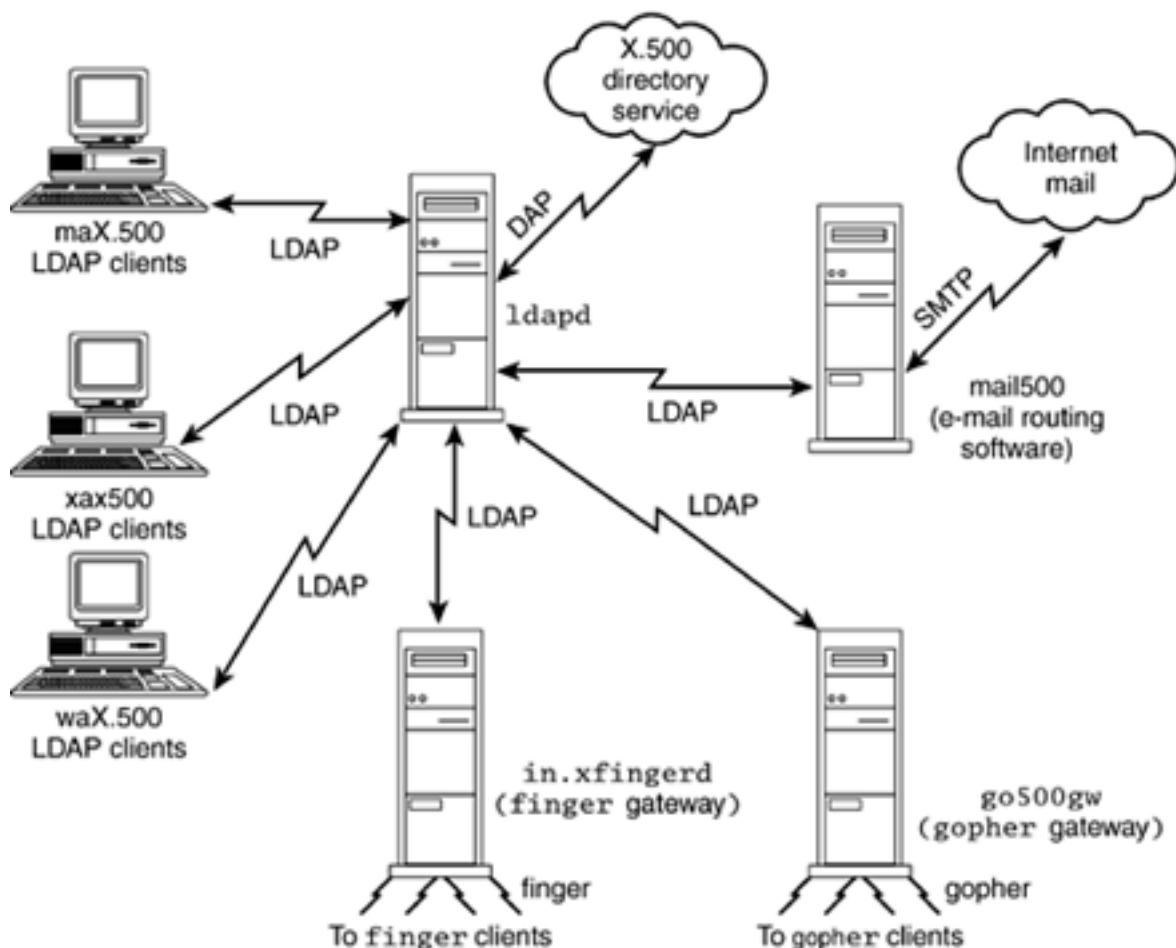
2. **Data representation.** In LDAP, most data elements are carried as simple text strings, thereby simplifying implementations and increasing performance (but for efficiency the text strings are wrapped inside binary-encoded messages).
3. **Encoding.** A subset of the X.500 encoding rules is used to encode LDAP messages, thus simplifying implementation.

4. **Transport.** LDAP runs directly over TCP instead of requiring the unwieldy, multilayer OSI networking stack. Implementation is simplified, performance is increased, and the need for OSI is eliminated, thus easing the deployment of LDAP directories.

Early LDAP Implementations

At first, LDAP was used exclusively to provide a front end for X.500-based directory services. The first LDAP implementation was produced by the authors while at the University of Michigan. The U-M LDAP implementation, as it came to be known, was small and fast, and it ran on a wide variety of popular computing platforms. Most importantly, it included a simple, well-specified C language API that implemented all of the client side of LDAP and could be used to develop any kind of LDAP client application. The LDAPv2 client API eventually became a de facto standard and was published in RFC 1823. [Figure 1.12](#) depicts the components of the early U-M LDAP software releases. The key server-side component is ldapd, which is an LDAP-to-X.500 DAP protocol translator.

Figure 1.12. Major Components of the Early U-M LDAP Releases



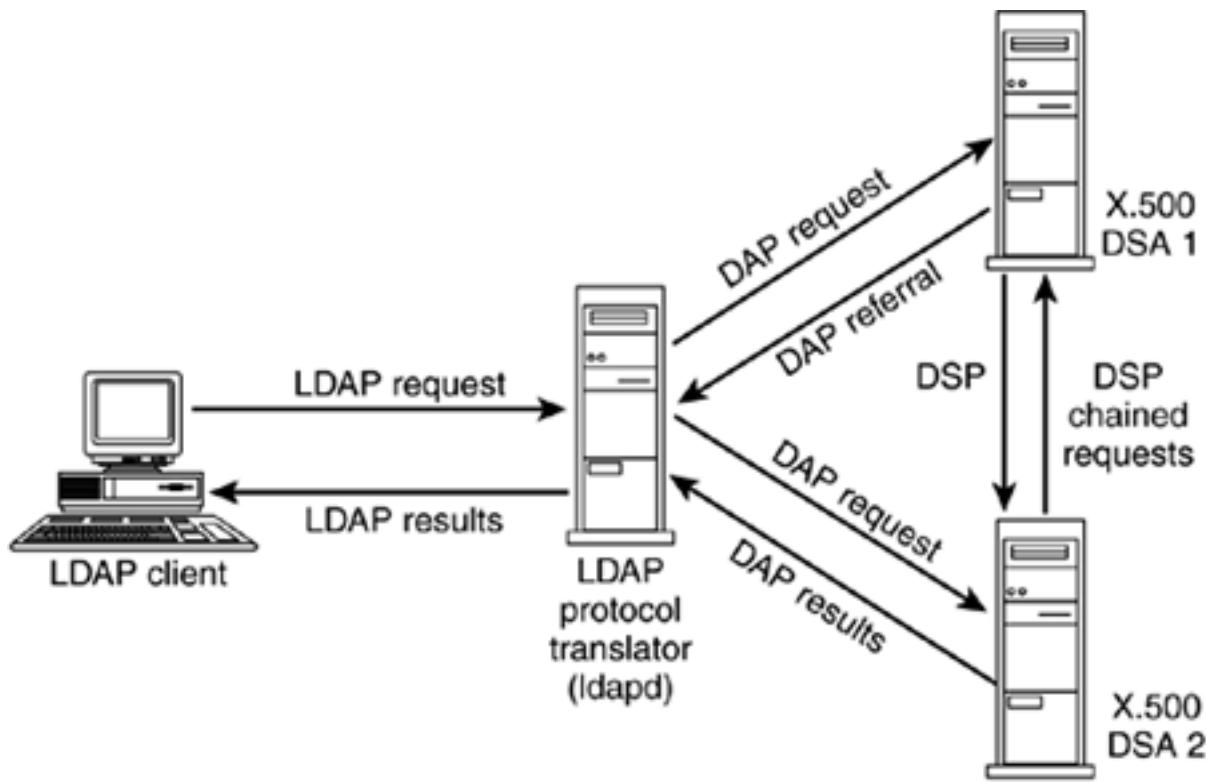
The first version of the U-M LDAP implementation was released publicly on the Internet in 1992 with a copyright notice that allowed unrestricted use of the software. A wide range of freely available and commercial LDAP client software soon followed, much of which was based on the University of Michigan implementation.

LDAP as a Standalone Directory Service

In early 1995, the LDAP group at U-M found itself on the brink of another revelation. When examining the directory access statistics for the U-M service, the group noticed that more

than 99 percent of the X.500 directory access came through LDAP. This was true of most X.500 directories. [Figure 1.13](#) shows the architecture prevalent at the time.

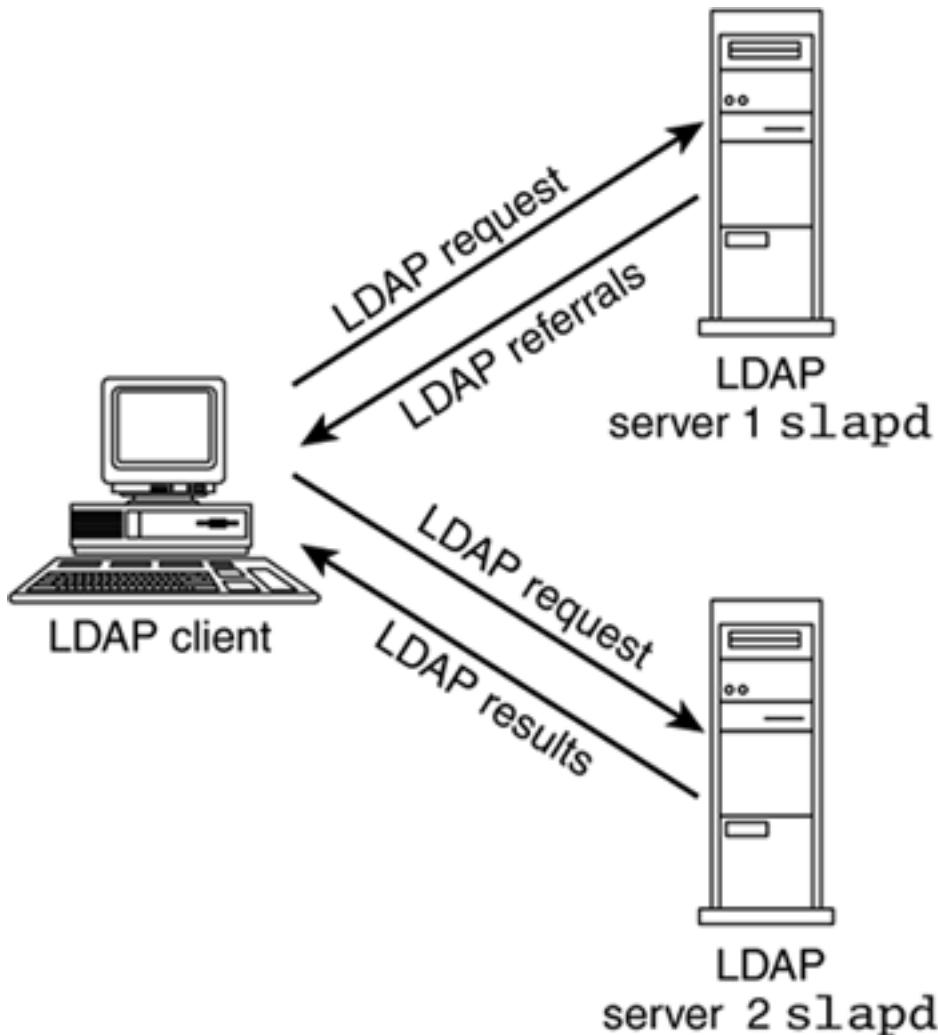
Figure 1.13. LDAP Directory System Architecture, circa 1995



Although LDAP had solved the directory client access problem, the server implementations were still large, complex, and difficult to deploy. The U-M LDAP group realized that by eliminating the intermediate `ldapd` server, they could greatly reduce the overall complexity of the system and greatly increase the performance. After all, `ldapd` spent all its time translating LDAP requests into DAP requests and recasting the DAP results returned from the X.500 servers as LDAP results.

In addition, the promise of a global, interconnected X.500 directory didn't seem to be achievable, so the need for an X.500 directory server at all was questioned. The concept of a standalone, or native, LDAP server was born. The new server was dubbed *slapd* (for *standalone LDAP daemon*), and with the aid of a grant from the National Science Foundation, it was quickly implemented. [Figure 1.14](#) shows the revised LDAP system architecture.

Figure 1.14. LDAP System Architecture after the Introduction of slapd



While keeping the best pieces of the X.500 models (see [Chapter 2](#) for more information), LDAP had now broken free of the last bit of X.500 and could proudly stand on its own. At the same time, referrals were added to the LDAPv2 protocol as an experimental extension to support an interconnected mesh of slapd directory servers. These changes moved LDAP out of its role as a simple, useful protocol for accessing X.500 directories to a much broader role as the foundation for a complete directory service. The first release of the U-M LDAP software to include slapd was the U-M LDAP 3.2 release in December 1995.

LDAP Momentum

LDAP's commercial success was assured in April 1996 when Netscape headed a coalition of more than 40 prominent software companies that endorsed LDAP as the Internet directory service protocol of choice. Today, companies such as Netscape, Sun Microsystems, and Novell produce a complete set of enterprise software centered on an LDAP-based directory service, and all major network application software vendors support LDAP in their client and server products.

LDAP Version 3 Developed

In 1997, LDAP entered an even more mature phase in its evolution with the release of version 3 of the protocol (LDAPv3). Mark Wahl, now with Sun Microsystems, along with some of the original LDAP authors and a cast of hundreds from across the Internet, contributed to the LDAPv3 specification. LDAPv3 improves on LDAPv2 in the following important areas:

- **Internationalization.** The UTF-8 (UCS Transformation Format 8) character set is

used everywhere it is appropriate. Because UTF-8 is an encoding of the universal Unicode character set, all the characters used in every language in the world can be stored and manipulated by LDAPv3 directory servers and clients.

- **Referrals.** A standard mechanism for returning referrals to other servers was added, which allows LDAP servers to be deployed in a bottom-up fashion, just as World Wide Web servers are.
- **Security.** Support for Simple Authentication and Security Layer (SASL) and Transport Layer Security (TLS) was added to increase LDAP's security and allow a richer set of applications to be supported.
- **Extensibility.** LDAPv3 can be extended to support new operations, and existing operations can be extended through the use of controls, allowing LDAP to be enhanced incrementally to meet future needs.
- **Feature and schema discovery.** All LDAPv3 servers publish the versions of the LDAP protocol they support, along with other useful information, in a special directory entry called the *root DSE* (DSA-Specific, or Directory Server-Specific, Entry). In addition, LDAPv3 servers publish their supported directory schemas as a set of LDAP attributes. These features help LDAP clients and servers work together more intelligently.

At the time of this writing, the LDAPv3 specification encompasses the following RFCs:

- **RFC 2251: *Lightweight Directory Access Protocol (v3)*.** Describes the LDAP protocol itself.
- **RFC 2252: *Lightweight Directory Access Protocol (v3): Attribute Syntax Definitions*.** Defines attribute syntaxes used by LDAP and its applications, including how these values are represented as simple strings.
- **RFC 2253: *Lightweight Directory Access Protocol (v3): UTF-8 String Representation of Distinguished Names*.** Describes how distinguished names are represented as simple strings with the UTF-8 character set.
- **RFC 2254: *The String Representation of LDAP Search Filters*.** Defines a scheme for representing LDAP search filters (which may be complex Boolean expressions involving multiple terms) as simple strings for use in LDAP URLs and APIs.
- **RFC 2255: *The LDAP URL Format*.** Defines a Uniform Resource Locator (URL) scheme to represent LDAP search requests and referrals.
- **RFC 2256: *A Summary of the X.500(96) User Schema for Use with LDAPv3*.** Provides a list of useful directory schemas (attributes and object classes) that are pulled from the X.500 specifications.
- **RFC 2829: *Authentication Methods for LDAP*.** Describes the required and optional authentication mechanisms that LDAPv3 servers and clients must support.
- **RFC 2830: *Lightweight Directory Access Protocol (v3): Extension for Transport Layer Security*.** Describes how to use Transport Layer Security (TLS) with LDAPv3 for authentication and privacy.

- **RFC 3377: Lightweight Directory Access Protocol (v3): Technical Specification**. Specifies the set of RFCs that define LDAP version 3 and addresses the "IESG Note" regarding the lack of satisfactory authentication mechanisms that is attached to RFCs 2251 through 2256.

Like their X.500 counterparts, developers of LDAP directories must absorb quite a few documents before they can successfully develop LDAP software. Fortunately, all RFCs are freely available on the Internet, and most of the RFCs related to LDAP are fairly short and easy to read. Unfortunately, some of the LDAP documents assume that the reader has knowledge that can be obtained only by consultation of the X.500 standards. For example, to understand all the details of the LDAP naming model (which specifies how to refer to and arrange directory information), you would need to read the X.500 documents.

In late 1997, LDAPv3 was approved as a proposed Internet Standard (the first step on its way to becoming a full Internet Standard), and in January 1998, Netscape shipped the first commercial LDAPv3 server, Netscape Directory Server 3.0. All serious directory services software vendors support LDAPv3 in their products today.

Some of the available LDAP directory services products are modeled after the original University of Michigan slapd concept and are built around a standalone LDAP server that includes a high-performance embedded database (many such products use some of the U-M code). In other cases, vendors of X.500 or other directory services strongly embraced LDAP and made it their directory access and operational protocol of choice. Some examples of general-purpose LDAP directory services available at the time of this writing include

- Netscape Directory Server
- Sun ONE Directory Server

IETF Directory Services Working Groups

Over the years, several different IETF working groups have dealt with LDAP. All these groups have been chartered within the Applications or Operations areas of the IETF.

The first group to deal with LDAP was OSI-DS. Some work that focused on adapting X.500 for the Internet, along with all the early DAS, DIXIE, and LDAP work, was done within the confines of OSI-DS.

In 1993 the directory services efforts in the IETF were split into several groups. These groups included the Access, Searching, and Indexing of Directories (ASID) group and the Integrated Directory Services (IDS) group. The core LDAPv3 work was done within the ASID group; IDS focused mainly on directory services issues that were not specific to LDAP, such as security and privacy.

Eventually, the ASID and IDS groups were disbanded, and a series of new working groups was chartered to continue LDAP's evolution. The LDAPEXT Working Group focuses on standardizing important extensions to LDAPv3 itself, such as authentication methods and server-side sorting, and it is likely to complete its work soon. The LDAP Duplication/Replication/Update Protocol (LDUP) group is working on server-to-server and server-to-client replication standards for LDAP directory services. The LDAPv3 Revision (LDAPBIS) Working Group is focused on revising the core LDAPv3 RFCs to advance them from Proposed to Draft Internet Standard status (one step closer to full Internet Standard). Thanks to the simple but flexible extension mechanisms included in LDAPv3, no one expects an LDAP version 4 to be needed anytime soon.

More information about these IETF working groups can be found on the IETF's World Wide Web site at <http://www.ietf.org./html.charters/wg-dir.html>.

- Microsoft Active Directory
- IBM Directory Server
- Novell eDirectory
- Oracle Internet Directory
- OpenLDAP slapd server

These directory products are generally designed to support a variety of Internet, intranet, and extranet applications. They usually provide good integration with other services that are based on Internet protocols, such as SMTP-based messaging servers, HTTP-based Web servers, and the like. They provide good administrative tools, are relatively inexpensive, and aim to be easy to deploy and maintain. Note that because these products do not all share a common design center, some are more suitable to certain kinds of directory deployments than others. For example, whereas Netscape Directory Server is designed to support mainly large intranet and e-commerce applications, Active Directory is designed to support mainly the administration of large Windows networks.

LDAP: Protocol or Directory Service?

Some people object to use of the term LDAP directory service. Why? Because, the argument goes, LDAP is merely a directory access protocol and therefore we should talk about only LDAP-enabled directory services—for example, an X.500 directory can be LDAP-enabled, and Novell eDirectory is LDAP-enabled. So why do we use the term *LDAP directory service* throughout this book?

Since the introduction of the University of Michigan standalone LDAP server (slapd), and with the specification of LDAPv3 by the IETF, directory servers that speak LDAP can stand on their own and need not be dependent on a full-blown implementation of the X.500 standards or any other directory specification other than the LDAP RFCs and the parts of X.500 that LDAP depends on. Also, the term *LDAP* has become commonly used to refer to servers; for example, people often say "LDAP server" just like they say "FTP server." Therefore, the natural label for a directory service based on a collection of LDAP servers is *LDAP directory service*. In fact, we can't think of a better term to describe a pure LDAP solution such as was pioneered by the University of Michigan and Netscape!

The Key Advantages of LDAP

Today it is clear that LDAP has moved to the front and center of the online directory services space, and much excitement and energy are being put into developing and deploying LDAP directories. LDAP emerged from the rest of the pack to dominate the directory services space and capture the interest of information technology professionals because of the following factors:

- **LDAP is simple, yet versatile.** LDAP supports a wide variety of directory-enabled applications that have widely varying needs.
- **LDAP is ubiquitous.** A good implementation of LDAP was developed and distributed freely on the Internet by researchers at the University of Michigan, thus providing

much of LDAP's early momentum. Today, LDAP implementations are available for every major and most minor computing platforms in use.

- **LDAP directories are inexpensive and easy to understand.** Organizations that choose LDAP directories find them to be relatively inexpensive to deploy and maintain. LDAP directory systems are simple enough that an army of consultants is not needed to understand and deploy them.
- **LDAP directory services simply work better.** The high reliability, performance, and scalability of LDAP directory products, combined with their general-purpose design, allow them to meet the most important directory services needs.
- **LDAP has a lot of "mindshare."** Shortly after Netscape and dozens of other software companies listened to requests from their customers and put their support behind LDAP, interest in LDAP exploded. The hunger for a single directory standard is being filled by LDAP, and momentum continues.

In short, LDAP is making the dream of a universal, general-purpose directory service a reality on the Internet and within organizations.

Further Reading

Development of the Domain Name System. P. Mockapetris and K. J. Dunlap, ACM SIGCOMM Symposium, August 1988.

Grapevine: An Exercise in Distributed Computing. A. D. Birrell, R. Levin, R. M. Needham, and M. D. Schroeder, *Communications of the ACM*, 25, April 1982, pp. 260–273.

Introduction to White Pages Services Based on X.500 (RFC 1684). P. Jurg, 1994. Available on the World Wide Web at <http://www.ietf.org/rfc/rfc1684.txt>.

The Lightweight Directory Access Protocol: X.500 Lite. T. Howes, CITI Technical Report 95-8, July 1995. Available on the World Wide Web at <http://www.citi.umich.edu/techreports/reports/citi-tr-95-8.pdf>.

Overview of the IETF. Available on the World Wide Web at <http://www.ietf.org/overview.html>.

Programming Directory-Enabled Applications with Lightweight Directory Access Protocol. T. Howes and M. Smith, Macmillan Technical Publishing, 1997.

A Scalable, Deployable, Directory Service Framework for the Internet. T. Howes and M. Smith, INET '95. Available on the World Wide Web at <http://info.isoc.org/HMP/PAPER/173/abst.html>.

A Survey of Advanced Usages of X.500 (RFC 1491). C. Weider and R. Wright, 1993. Available on the World Wide Web at <http://www.ietf.org/rfc/rfc1491.txt>.

Understanding X.500: The Directory. D. Chadwick, International Thomson Computer Press, 1996. Now out of print; selected portions available on the World Wide Web at <http://www.salford.ac.uk/its024/X500.htm>.

X.500 Directory Services: Technology and Deployment. S. Radicati, Van Nostrand Reinhold, 1994.

Looking Ahead

This chapter has provided an overview of directories and directory services. Starting from a layman's understanding of everyday, offline directories, we've explained analogous online directories and the features that enable them to be used for new types of applications. By now, you should have a good idea of what a directory is and what it can do for you. Equally importantly, you should have a good idea of what a directory is not, and what you should not try to use it for. We also introduced LDAP by discussing its origins and some of its key features. [Chapter 2](#), Introduction to LDAP, will describe LDAP thoroughly, in preparation for in-depth coverage of directory design.

Chapter 2. Introduction to LDAP

- What Is LDAP?
- The LDAP Models
- LDIF
- LDAP Server Software
- LDAP Command-Line Utilities
- LDAP APIs
- LDAP and Internationalization
- LDAP Overview Checklist
- Further Reading
- Looking Ahead

The second part of [Chapter 1](#), Directory Services Overview and History, discussed the origins of general-purpose, standards-based directories and how LDAP was born. This chapter takes a much closer look at LDAP.

What Is LDAP?

Today, the term *LDAP* refers to the following things:

- The Lightweight Directory Access Protocol, a standard, extensible Internet protocol used to access directory services
- A set of four models that guide you in your use of the directory: an information model that describes what you can put in the directory, a naming model that describes how you arrange and refer to directory data, a functional model that describes what you can do with directory data, and a security model that describes how directory data can be protected from unauthorized access
- The LDAP Data Interchange Format (LDIF), a standard text format for exchanging directory data
- LDAP server software, including commercial and open-source implementations
- A set of command-line utilities commonly bundled with LDAP servers and LDAP-based applications
- The LDAP application programming interfaces (APIs), used to develop LDAP client applications

All of these are described in further detail in the next several sections.

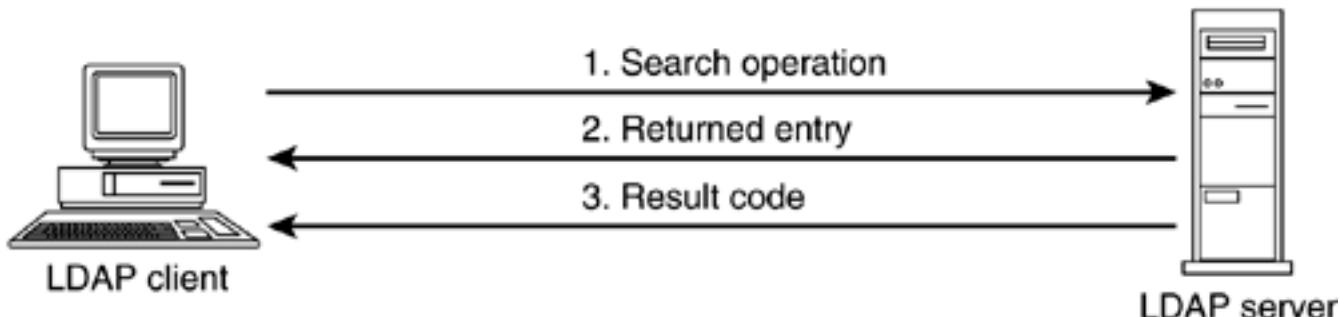
The LDAP Protocol

In this section we'll discuss the individual LDAP protocol operations and show how clients use them to perform useful tasks. We'll also show how LDAP works "on the wire" by discussing the wire protocol.

It's important to understand that the LDAP protocol is a *message-oriented* protocol. The client constructs an LDAP message containing a request and sends it to the server. The server processes the request and sends the result(s) back to the client as a series of one or more LDAP messages.

For example, when an LDAP client searches the directory for a specific entry, it sends an LDAP search request message to the server. This message contains a unique message ID, generated by the client. The server retrieves the entry from its database and sends it to the client in an LDAP message. It also returns a result code to the client in a separate LDAP message. All the responses from the server to the client are identified with the message ID provided in the original client request. [Figure 2.1](#) shows this interaction.

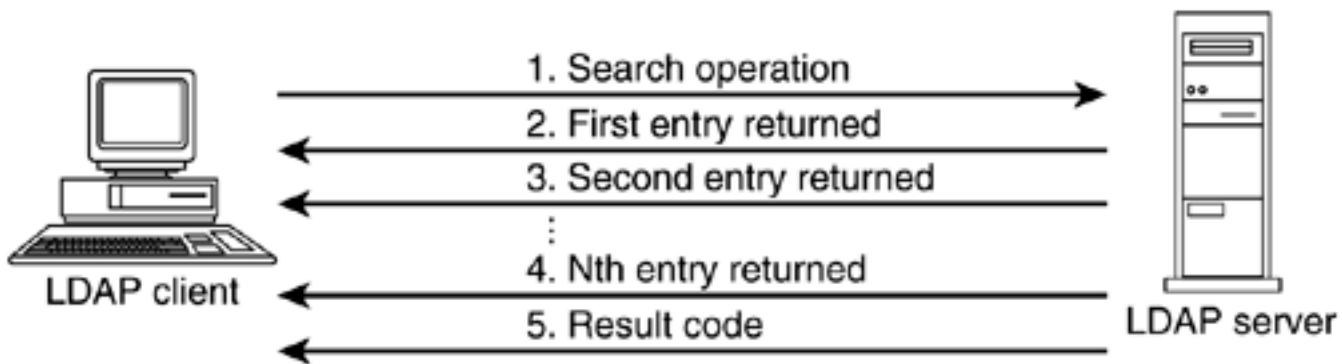
Figure 2.1. A Client Retrieves a Single Entry from the Directory



If the client searches the directory and finds multiple entries, those entries are sent to the client in a series of LDAP messages, one for each entry. Each entry has a unique name

called a *distinguished name (DN)*, which is carried in the LDAP messages as a text string. The entries that constitute the search results are terminated with a result message, which contains an overall result for the search operation, as shown in [Figure 2.2](#).

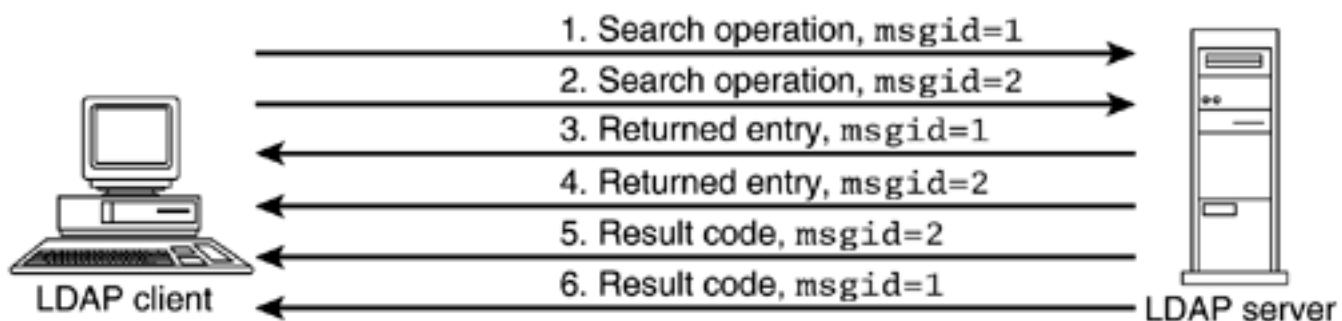
Figure 2.2. A Client Retrieves Multiple Entries from the Directory



Because the LDAP protocol is message-based, it also allows the client to issue multiple requests at once. For example, a client might issue two search requests simultaneously. The client generates a unique message ID for each request; returned results for a specific request are tagged with the request's message ID, allowing the client to sort out multiple responses to different requests arriving out of order or at the same time.

In [Figure 2.3](#), the client issued two search requests simultaneously. The server processes both operations and returns the results to the client. Notice that the server sends the final result code of message 2, identified in the figure as `msgid=2`, to the client before it sends the final result code from message 1. This is perfectly acceptable. These details are hidden from the programmer by an LDAP SDK (software development kit). Programmers writing an LDAP application don't need to be concerned with sorting out these results; the SDKs take care of this automatically.

Figure 2.3. A Client Issues Multiple LDAP Search Requests Simultaneously



Allowing multiple concurrent requests "in flight" enables LDAP to be more flexible and efficient than protocols that operate in a "lockstep" fashion (for example, Hypertext Transfer Protocol, or HTTP). With a lockstep protocol, each client request must be answered by the server before another may be sent. For example, an HTTP client program—such as a Web browser that wants to download multiple files concurrently—must open one connection for each file. LDAP, on the other hand, can manage multiple operations on a single connection, reducing the maximum number of concurrent connections a server must be prepared to handle.

LDAP has nine basic protocol operations, which can be divided into three categories:

1. **Interrogation operations: search, compare.** These two operations allow you to ask questions of the directory.
2. **Update operations: add, delete, modify, modify DN (rename).** These operations allow you to update information in the directory.
3. **Authentication and control operations: bind, unbind, abandon.** The bind operation allows a client to identify itself to the directory by providing an identity and authentication credentials; the unbind operation allows the client to terminate a session; and the abandon operation allows a client to indicate that it is no longer interested in the results of an operation it had previously submitted.

We will discuss each protocol operation when we describe the LDAP functional model later in this chapter.

A typical complete LDAP client/server exchange might proceed as depicted in [Figure 2.4](#), which shows an LDAP client and server performing the following steps:

Step 1. The client opens a TCP connection to an LDAP server and submits a bind operation. This bind operation includes the name of the directory entry the client wants to authenticate as, along with the credentials to be used for authenticating. Credentials are often simple passwords, but they might also be digital certificates used to authenticate the client.

Step 2. After the directory has verified the bind credentials by checking that the password or digital certificate is correct, it returns a success result to the client.

Step 3. The client issues a search request.

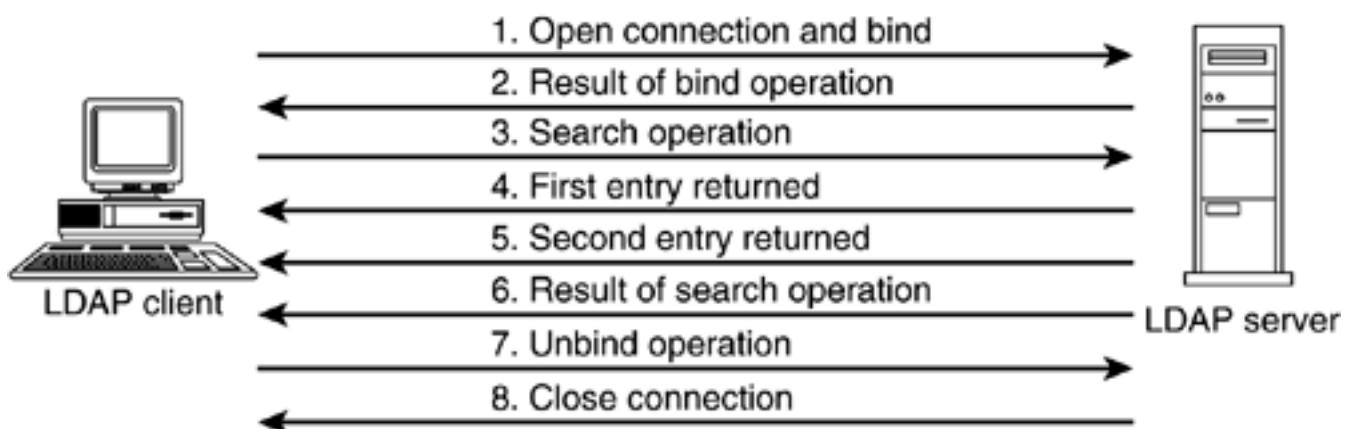
Step 4 and 5. The server processes this request, which results in two matching entries.

Step 6. The server sends a result message.

Step 7. The client then issues an unbind request, which indicates to the server that the client wants to disconnect.

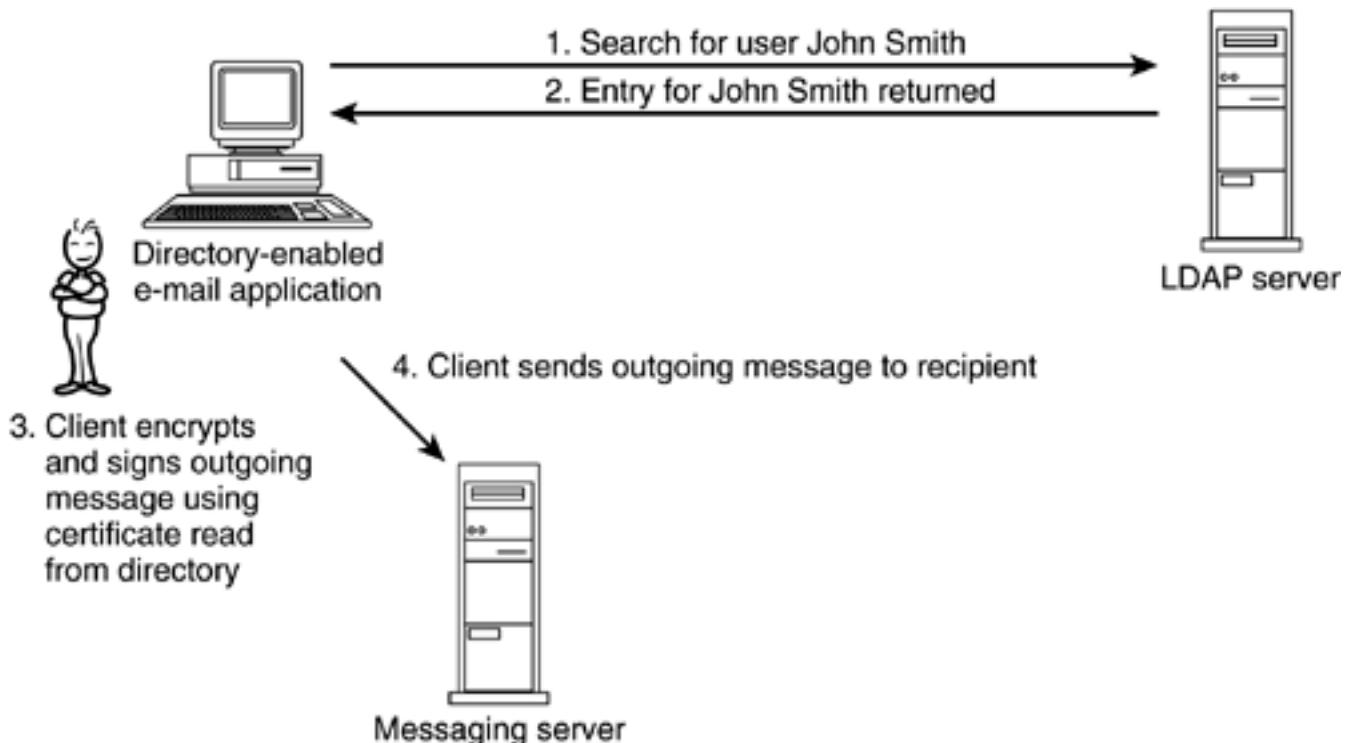
Step 8. The server obliges by closing the connection.

Figure 2.4. A Typical LDAP Exchange



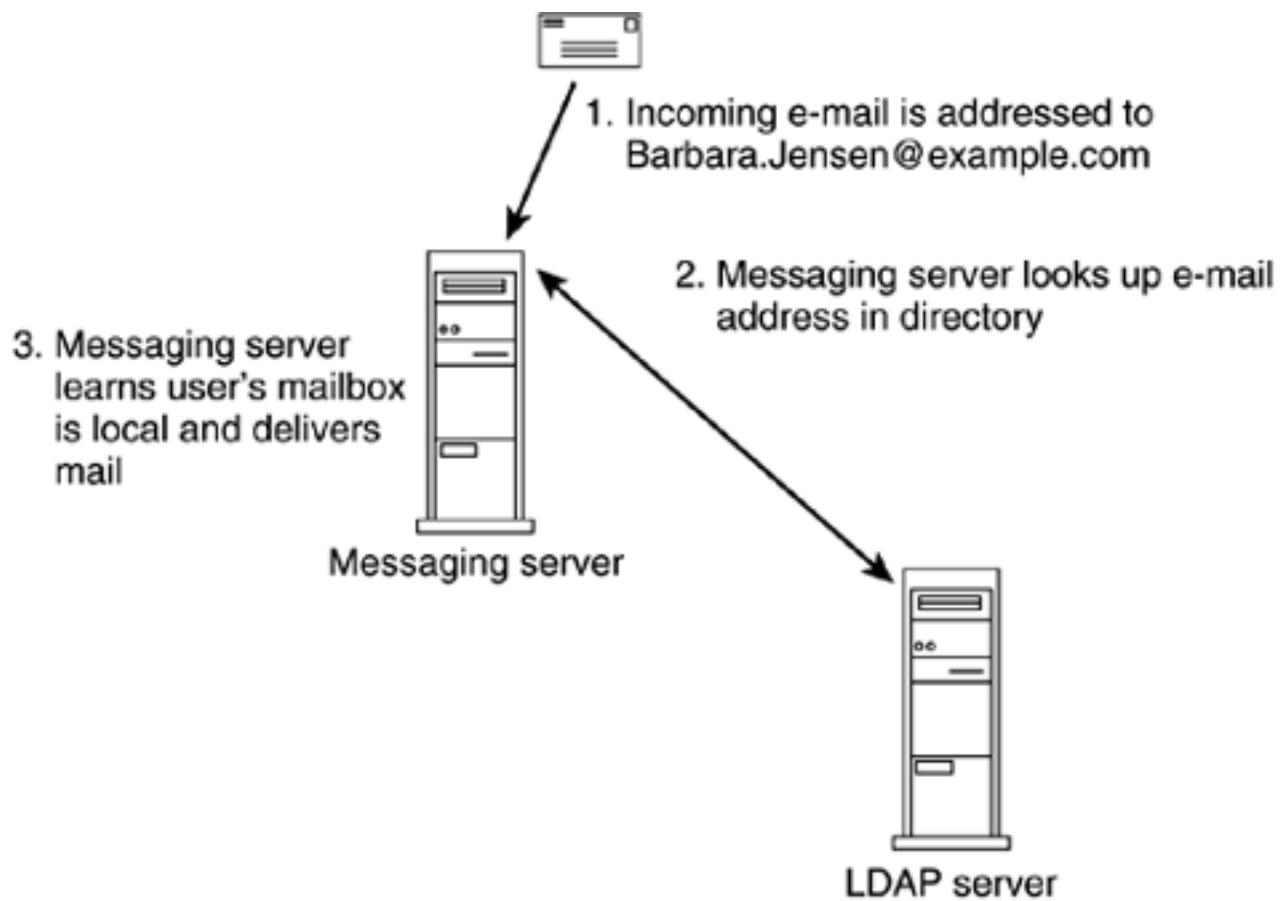
By combining several of these simple LDAP operations, directory-enabled clients can perform complex tasks that are useful to their users. For example, as shown in [Figure 2.5](#), an electronic mail client such as Netscape Communicator can look up mail recipients in a directory, helping a user to address an e-mail message. It can also use a digital certificate stored in the directory to digitally sign and encrypt an outgoing message. Behind the scenes, the user's e-mail program performs several directory operations that allow the mail to be addressed, signed, and encrypted. But from the user's point of view, it is all taken care of automatically.

Figure 2.5. A Directory-Enabled Application Performing a Complex Task



End-user applications are not the only kind of directory-enabled applications. Server-based applications often benefit from being directory-enabled too. For example, Sun ONE Messaging Server can use an LDAP directory when routing incoming electronic mail, as shown in [Figure 2.6](#).

Figure 2.6. A Directory-Enabled Server Application



LDAP Extensibility

In addition to providing the nine basic protocol operations, LDAPv3 is designed to be extensible via three methods:

1. **LDAP extended operations.** Entirely new protocol operations, which can be defined by means of LDAPv3 extended operations. If the need arises for a new operation, it can be defined and made standard without changes having to be made to the core LDAP protocol. An example of an extended operation is StartTLS, which indicates to the server that the client wants to begin using Transport Layer Security (TLS) to encrypt and optionally authenticate the connection.
2. **LDAP controls.** Extra pieces of information that piggyback on existing LDAP operations, altering the behavior of the operation. For example, the `ManageDSAIT` control is sent along with a modify operation when the client wants to manipulate certain types of metainformation stored in the directory (this metainformation is normally hidden from users of the directory). In the future, additional controls may be defined that alter the behavior of existing LDAP operations in useful ways.
3. **Simple Authentication and Security Layer (SASL).** A framework for supporting multiple authentication methods. If the SASL framework is used to implement authentication, LDAP can easily be adapted to support new, stronger authentication methods. SASL also supports a framework for clients and servers to negotiate lower-layer security mechanisms, such as encryption of all client/server traffic. SASL is not specific to LDAP, though; its general framework can be adapted to a wide range of Internet protocols.

We'll discuss LDAPv3 extensions in detail in [Chapter 3](#), LDAPv3 Extensions.

The LDAP Protocol on the Wire

What information is transmitted back and forth between LDAP clients and servers? We won't go into a great deal of detail here because this book isn't about protocol design, but there are a few things you should know about the LDAP wire protocol.

LDAP uses a simplified version of the *Basic Encoding Rules (BER)*, a set of rules for encoding various data types, such as integers and strings, in a system-independent and compact fashion. BER also defines ways of combining these primitive data types into useful structures such as sets and sequences. The simplified BER that LDAP uses is often referred to as *Lightweight BER (LBER)*. LBER does away with many of the more esoteric data types that BER can represent, simplifying implementation.

Because LDAP uses LBER, it is not a simple text-based protocol like HTTP, and you can't simply telnet to the LDAP port on your server and start typing commands. Also, if you want to use a network sniffer program to observe LDAP traffic between a client and server, the sniffer program needs to understand the LDAP protocol to be useful (most state-of-the-art sniffer software does—for example, there is an LDAP module for Ethereal, which is a well-known open-source sniffer). If you are familiar with text-based Internet protocols such as Post Office Protocol (POP), Internet Message Access Protocol (IMAP), and Simple Mail Transfer Protocol (SMTP), this may seem like an unfortunate limitation. On the other hand, the Domain Name System (DNS), a successful distributed system, uses a protocol that has nontextual protocol primitives. The presence of universal implementations of client libraries for both DNS and LDAP makes this limitation less problematic.

The LDAP Models

LDAP defines four basic models that fully describe how it operates, what data can be stored in LDAP directories, and what can be done with that data.

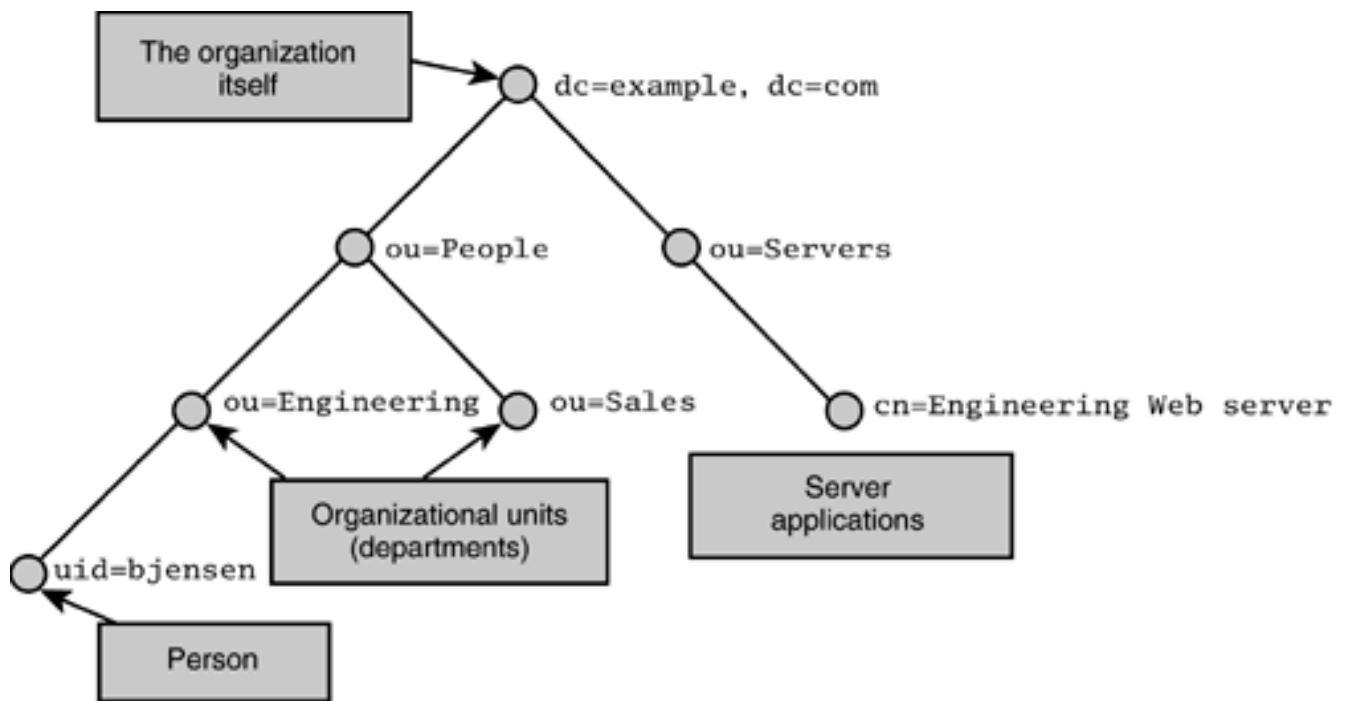
The LDAP Information Model

The LDAP information model defines the types of data and basic units of information you can store in your directory. In other words, the LDAP information model describes the building blocks you can use to create your directory.

Entries, Attributes, and Values

The basic unit of information in the directory is the *entry*, a collection of information about an object. Often the information in an entry describes a real-world object such as a person, but the model does not require this. In a typical directory you'll find thousands of entries that correspond to people, departments, servers, printers, and other real-world objects in the organization served by the directory. [Figure 2.7](#) shows a portion of a typical directory, with objects corresponding to some of the real-world objects in the organization.

Figure 2.7. Part of a Typical Directory



Each directory entry has a distinguished name (DN); for example, the organization shown in [Figure 2.7](#) has the DN `dc=example,dc=com`. We will discuss DNs in detail later in this chapter when we dive into the LDAP naming model. An entry is composed of a set of *attributes*, each of which describes one particular trait of the object. Each attribute has a *type* and one or more *values*. The type describes the kind of information contained in the attribute, and the value contains the actual data. For example, [Figure 2.8](#) zooms in on an entry describing a person, with attributes for the person's full name, surname (last name), telephone number, and e-mail address.

Figure 2.8. A Directory Entry Showing Attribute Types and Values

Attribute type	Attribute values
cn:	Barbara Jensen Babs Jensen
sn:	Jensen
telephoneNumber:	+1 408 555 1212
mail:	babs@example.com

Note

Throughout this book you'll see directory entries shown in the LDIF (LDAP Data Interchange Format) text format. This is a standard way of representing directory data in a textual format, and it is used when data is being exported from and imported into a directory server. We'll describe LDIF in detail later in this chapter.

Attribute types also have an associated *syntax* and a set of *matching rules*. The syntax of an attribute specifies the form of the data that may be present in an attribute of that type. For example, the `INTEGER` syntax allows only digits to be present in a value. Matching rules specify the following things:

- **The rules used for comparing values for equivalence.** For example, the `caseIgnoreMatch` rule specifies that case is not significant when values are being searched or replaced. In comparisons of the values `Smith` and `smith`, they are considered equivalent if the `caseIgnoreMatch` matching rule is used.
- **The rules used for sorting values.** For example, the `caseIgnoreMatch` rule specifies that values are ordered lexicographically, but the `integerMatch` rule specifies that values are ordered according to their numerical values.

All standards-compliant LDAP server software supports a set of well-known required syntaxes and matching rules; see RFC 2252 (<http://www.ietf.org/rfc/rfc2252.txt>) for a list. Some packages, such as Netscape Directory Server, provide a plug-in interface or another mechanism that allows additional syntaxes or matching rules to be added.

Attributes are also classified broadly into two categories: user and operational. *User attributes*, the "normal" attributes of an entry, may be modified by the users of the directory (with appropriate permissions). *Operational attributes* are special attributes that either modify the operation of the directory server or reflect the operational status of the directory. An example of an operational attribute is the `modifyTimeStamp` attribute, which reflects the time that the entry was last modified and is automatically maintained by the directory. When an entry is sent to a client, operational attributes are not included unless the client requests them by name.

Attribute values can also have additional constraints placed on them. Some server software allows the administrator to declare whether a given attribute type may hold multiple values or whether only a single attribute value may be stored. For example, the `givenName` attribute is typically multivalued so that a person can include more than one given name (for example, `Jim` and `James`). On the other hand, an attribute holding an employee ID number is likely to be single-valued. Some server software allows the administrator to set the maximum length of an attribute's values. This feature can be used to prevent directory users from using unreasonable amounts of storage.

Maintaining Order: Directory Schemas

Any entry in the directory has a set of required attribute types and a set of allowed attribute types. For example, an entry describing a person is required to have a `cn` (common name) attribute and an `sn` (surname) attribute. Other attributes are allowed, but not required, for person entries. Any attribute type not explicitly required or allowed is prohibited. The collections of all information about required and allowed attributes are called the *directory schemas*.

It's important to understand that directory schemas do not have any bearing on the arrangement of entries into the LDAP directory tree. This may seem odd if you are familiar with relational database technology, where *schema* is a more inclusive term describing the layout of database tables and their relationship to one another. The LDAP schema is a simpler concept, and therefore it determines only what types of data appear in an individual entry.

Directory schemas, which are discussed in detail in [Chapter 8](#), Schema Design, allow you to retain control and maintain order over the types of information stored in your directory.

In summary, the LDAP information model describes *entries*, which are the basic building blocks of your directory. Entries are composed of attributes, which are composed of an attribute type and one or more values. Attributes may have constraints that limit the type and length of data placed in attribute values. The directory schemas place restrictions on the attribute types that must be or are allowed to be contained in an entry.

The LDAP Naming Model

The LDAP naming model defines how you organize and refer to your data. In other words, it describes the types of structures you can build out of your individual building blocks, which are the directory entries. After you've arranged your entries into a logical structure, the naming model also tells you how to refer to any particular directory entry within that structure.

The flexibility afforded by the LDAP naming model allows you to place your data in the directory in a way that is easy for you to manage. For example, you might choose to create one container to hold all the entries describing people in your organization, and another container to hold all your groups. Alternatively, you might choose to arrange your directory in a way that reflects the geographical placement of your organization's offices. [Chapter 9](#), Namespace Design, guides you in making good choices when you design your directory hierarchy or namespace.

The LDAP naming model specifies that entries should be arranged in an inverted tree structure, as shown in [Figure 2.9](#). Readers familiar with the hierarchical file system used by Unix systems will note its similarities to this directory structure. Such a file system consists of a set of directories and files; each directory may have zero or more files or directories beneath it. [Figure 2.10](#) shows part of a typical Unix file system.

Figure 2.9. A Directory Tree

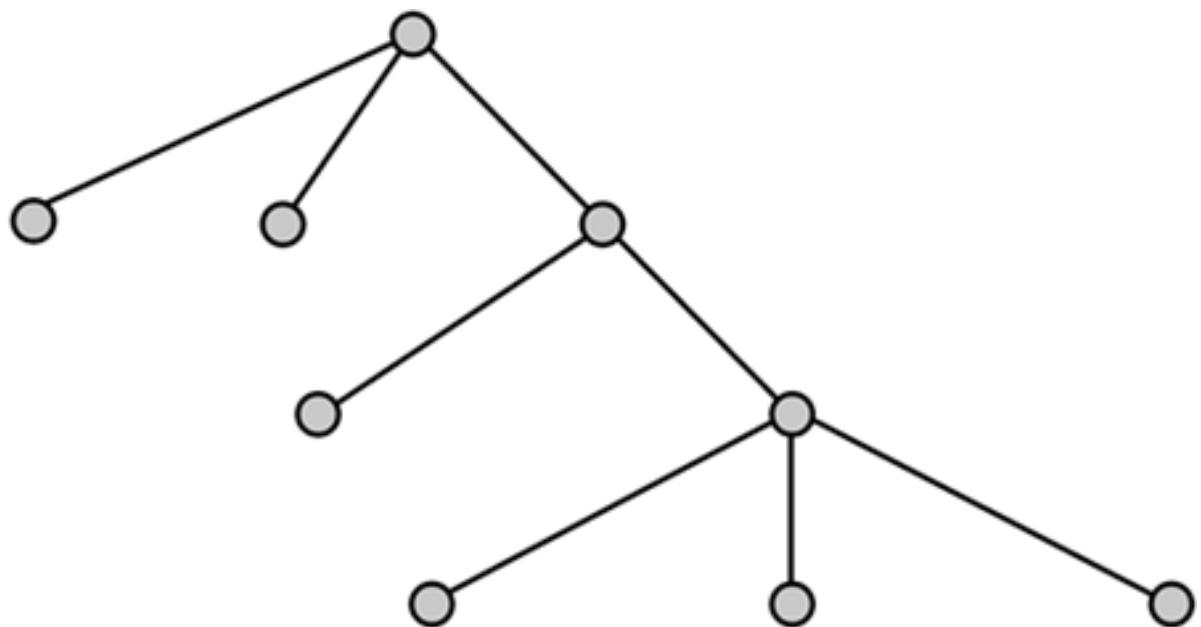
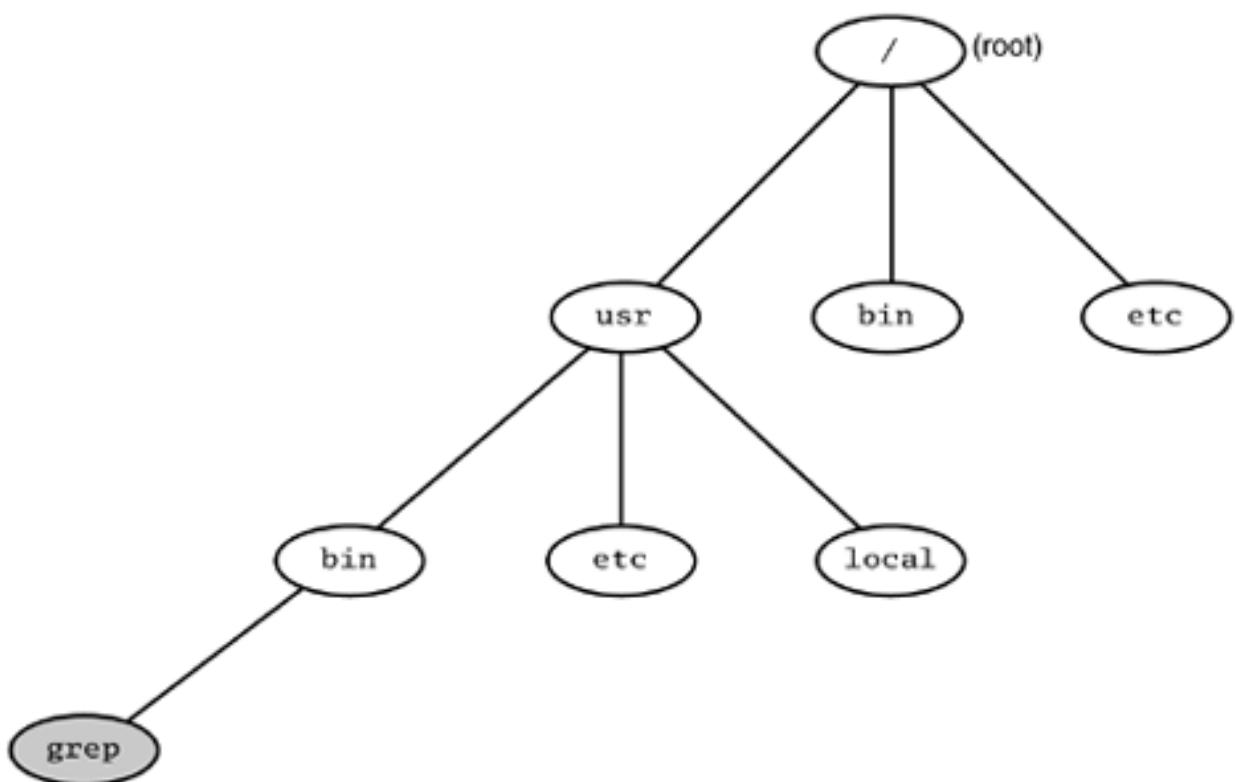


Figure 2.10. Part of a Typical Unix File System

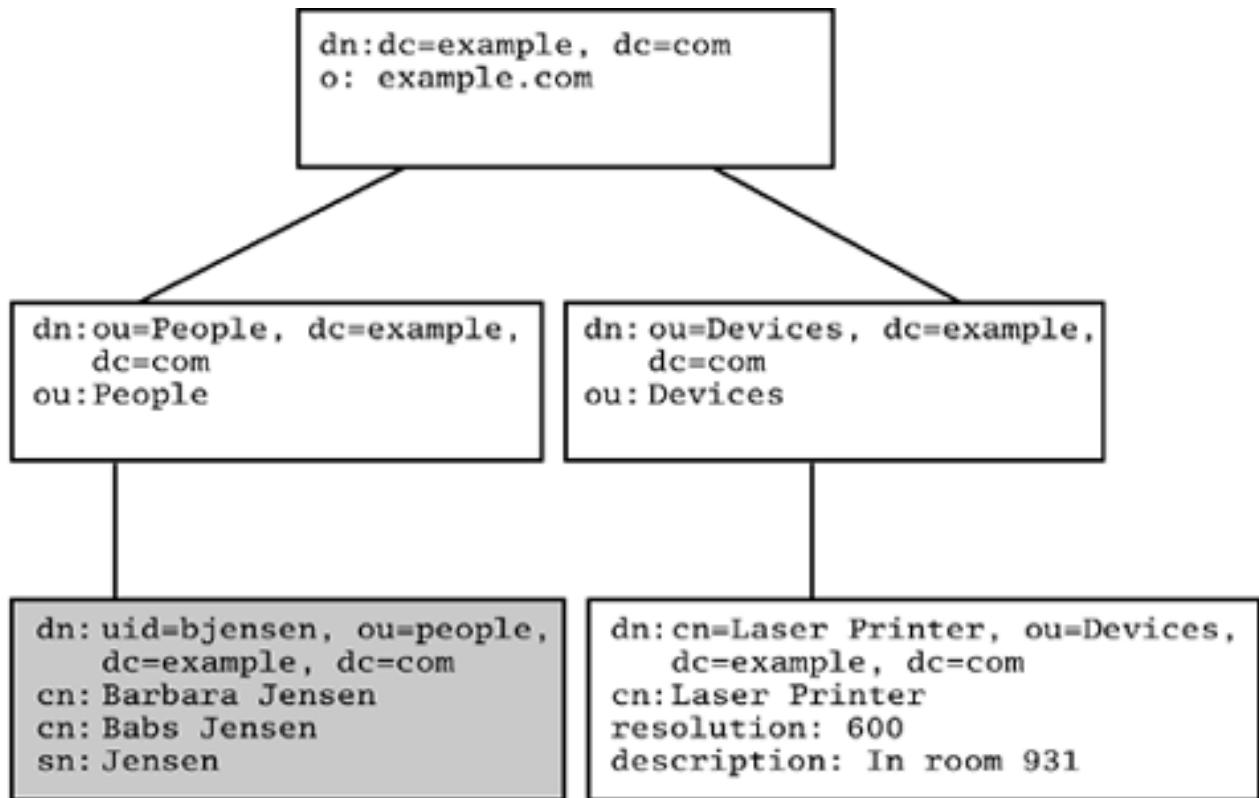


There are three significant differences between the Unix file system hierarchy and the LDAP directory hierarchy, however. The first major difference between the two models is that there isn't really a root entry in the LDAP model. A file system, of course, has a root directory, which is the common ancestor of all files or directories in the file system hierarchy. In an LDAP directory hierarchy, on the other hand, the root entry is a special entry that contains configuration information about the directory server. It is not normally used to store information.

The second major difference is that in an LDAP directory, every node contains data, and any node can be a container. This means that any LDAP entry may have child nodes underneath it. In contrast, in a file system a given node is either a file or a directory, but not both. In the

file system, only directories may have children, and only files may contain data. Another way of thinking of this is that an entry in a directory may be both a file and a directory simultaneously. The directory tree shown in [Figure 2.11](#) illustrates this concept. Notice how the entries `dc=example`, `dc=com`, `ou=People`, and `ou=Devices` all contain data (attributes) but are also containers with child nodes beneath them.

Figure 2.11. Part of a Typical LDAP Directory



The third and final difference between the file system hierarchy and the LDAP hierarchy is how individual nodes in the tree are named. LDAP names are backward relative to file system names. As illustration, consider the names of the shaded nodes in [Figures 2.10](#) and [2.11](#). In [Figure 2.10](#), the shaded node is a file with a complete filename of `/usr/bin/grep`. Notice that if you read the filename from left to right, you move from the top of the tree (`/`) down to the specific file being named.

Contrast this name with the name of the shaded directory entry in [Figure 2.11](#): `uid=bjensen, ou=people,dc=example,dc=com`. Notice that, if you read from left to right, you move from the specific entry being named back up toward the top of the tree. We'll discuss directory entry names in detail in [Chapter 9](#), Namespace Design.

Although LDAP supports a hierarchical arrangement of directory entries, it does not mandate any particular type of hierarchy. Just as you're free to arrange your file system in a way that makes sense to you and is easy for you to manage, you're free to construct any type of directory hierarchy you want. Of course, some directory structures are better than others, depending on your particular situation; we'll cover the topic of designing your directory namespace in [Chapter 9](#).

The one exception to this freedom is if your LDAP directory service is actually a front end to an X.500 service. The X.500 naming model is much more restrictive than the LDAP naming model. In the X.500 1993 standard, directory structure rules limit the types of hierarchies you can create. The standard accomplishes this restriction by specifying what types of "object classes" may be direct children of an entry. In the X.500 model, for example, only entries

representing countries, localities, or organizations may be placed at the root of the directory tree. The LDAP naming model, on the other hand, does not limit the tree structure in any way; any type of entry may be placed anywhere in the tree.

In addition to specifying how to arrange directory entries into hierarchical structures, the LDAP naming model describes how to refer to individual entries in the directory. We mentioned this briefly when we were discussing the similarities and differences between file system hierarchy and LDAP directory hierarchy. Now let's go into more detail about naming.

Why Is Naming Important?

A naming model is needed so that you can give a unique name to any entry in the directory, allowing you to refer to any entry unambiguously. In LDAP, distinguished names (DNs) are how you refer to entries.

Like file system pathnames, we form the name of an LDAP entry by connecting in a series all the individual names of the parent entries back to the root. For example, look back at the directory tree shown in [Figure 2.11](#). The shaded entry's name is `uid=bjensen,ou=people,dc=example,dc=com`. Reading this name from left to right, you can trace the path from the entry itself back to the root of the directory tree. The individual components of the name are separated by commas. Spaces after the commas are optional, so the following two distinguished names are equivalent:

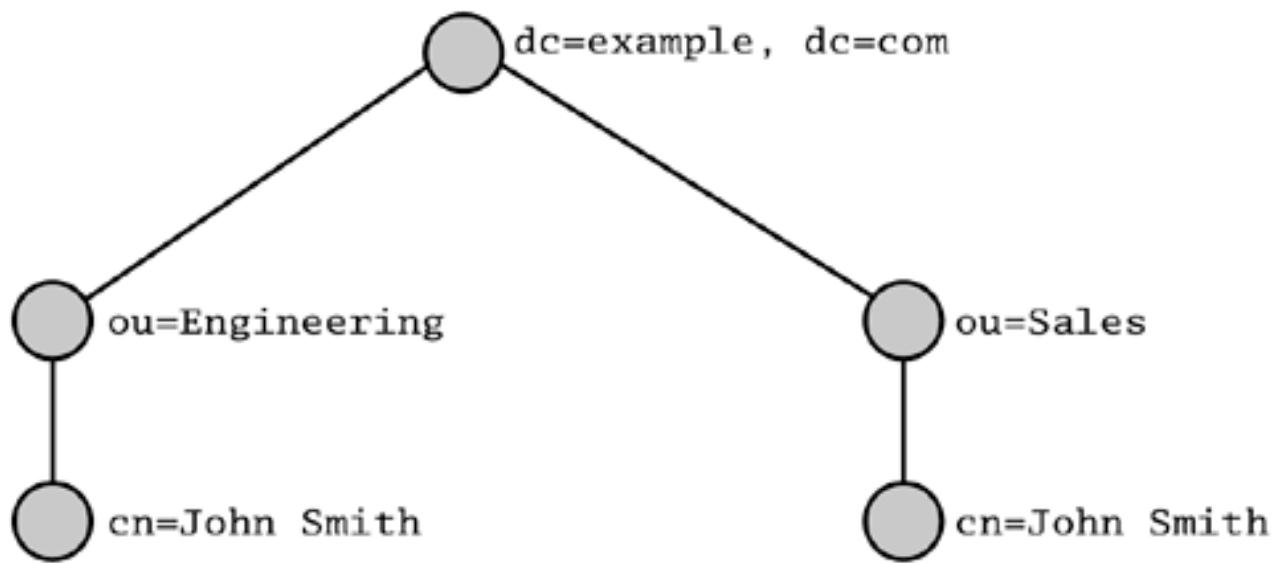
`uid=bjensen, ou=people, dc=example, dc=com`

`uid=bjensen,ou=people,dc=example,dc=com`

In any entry's DN, the leftmost component is called the *relative distinguished name (RDN)*. Among a set of peer entries (those that share a common immediate parent), each RDN must be unique. This rule, when applied recursively to the entire directory tree, ensures that no two entries have the same DN. If you attempt to add two entries with the same name, the directory server will reject the attempt to add the second entry; this is similar to a Unix or Microsoft Windows file system, which will reject an attempt to create a file that has the same name as an existing file within a directory.

RDNs have to be unique only if they share a common immediate parent. Look at the tree in [Figure 2.12](#). Even though two entries have the RDN `cn=John Smith`, they are in different subtrees, so the tree is completely legal. Whether this is a good way to construct your directory is another matter, one addressed in [Chapter 9](#), Namespace Design.

Figure 2.12. Entries with the Same RDNs Are Permitted If They Are in Different Parts of the Tree



Multivalued RDNs, and Why You Should Avoid Using Them

You've probably noticed that each RDN we've shown is composed of two parts: an attribute name and a value, separated by an equal sign (`=`). An RDN may also contain more than one such name–value pair. Such a construction, called a *multivalued RDN*, looks like this:

```
cn=John Smith + mail=jsmith@example.com
```

The RDN for this entry consists of two `attribute=value` pairs: `cn=John Smith` and `mail=jsmith@example.com`.

Multivalued RDNs can be used to distinguish RDNs that would otherwise be the same. For example, if there were more than one `John Smith` entry in the same container, a multivalued RDN would allow you to assign unique RDNs to each entry. However, you should generally avoid using multivalued RDNs, for two reasons: First, they tend to clutter your namespace, and there are better ways to arrive at unique names for your entries. (Approaches for uniquely naming your entries are discussed in [Chapter 9](#), Namespace Design.) Second, according to X.500 specifications, from which LDAP is derived, the values in a multivalued RDN are a set, which means that the ordering is not significant. Therefore, the following two DNs refer to the same entry:

```
cn=John Smith + mail=jsmith@example.com, dc=example, dc=com
```

```
mail=jsmith@example.com + cn=John Smith, dc=example, dc=com
```

Client applications that need to compare DNs must be sophisticated enough to understand that these two DNs are equivalent. Realistically, few applications handle this correctly. Also, many off-the-shelf directory applications do not allow you to create entries with multivalued RDNs, nor do they properly handle any such entries that may exist in your directory. Our advice is to avoid using multivalued RDNs in your directory.

Escaping

Certain characters must be escaped when they appear within a component of a DN. For example, what if you have the entry `o=United Widgets,Ltd.` in your directory? The comma is part of the organization's name, not a separator between DN components. To resolve the ambiguity, you must escape all literal commas (those within an RDN) with a backslash (\). In our example, then, the DN would be

Table 2.1. Characters That Must Be Escaped If Contained in Distinguished Names

Character	Decimal Value	Escape Sequence
Space at the beginning or end of a DN or RDN	32	\<space>
Octothorp (#) character at the beginning of a DN or RDN	35	\#
Comma (,)	44	\,
Plus sign (+)	43	\+
Double quote ("")	34	\"
Backslash (\)	92	\\\
Less-than symbol (<)	60	\<
Greater-than symbol (>)	62	\>
Semicolon (;)	59	\;

`o=United Widgets\, Ltd., c=GB`

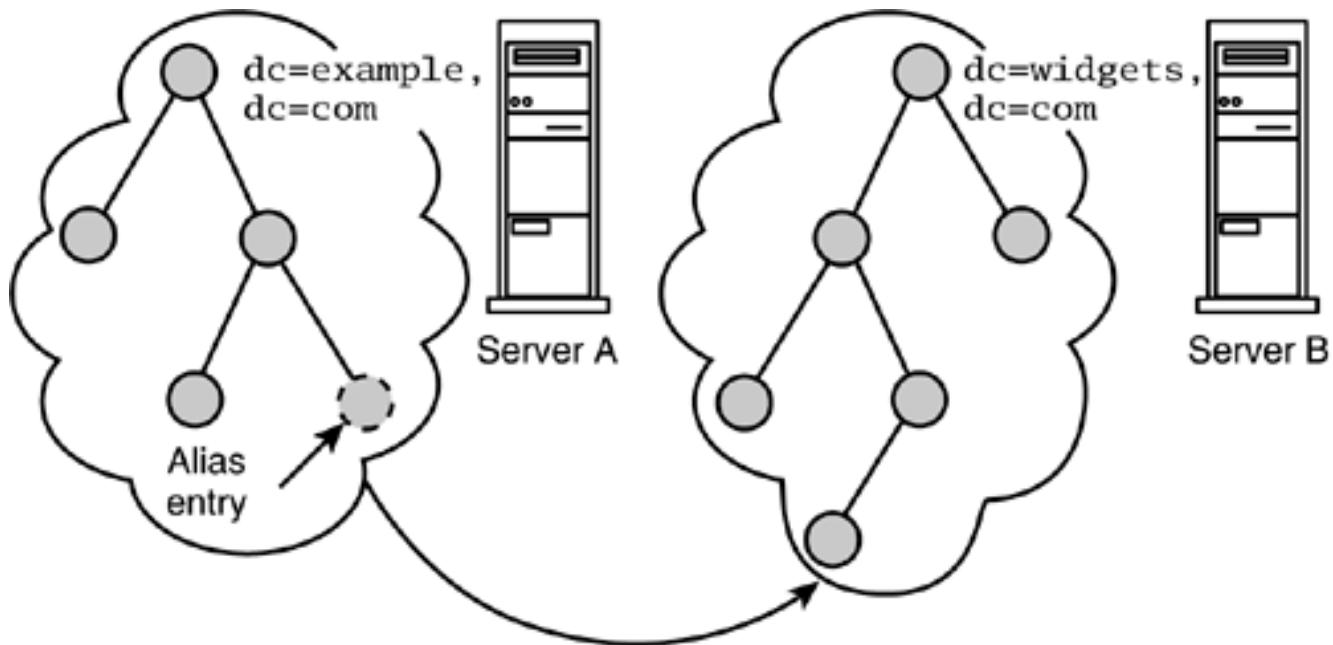
[Table 2.1](#) shows all the characters that must be escaped, according to the LDAPv3 specification.

Aliases

Alias entries in the LDAP directory allow one entry to point to another one, which means that you can devise structures that are not strictly hierarchical. Alias entries perform a function

like symbolic links in the Unix file system or shortcuts in the Windows 95/NT file system. In [Figure 2.13](#), the dotted entry is an alias entry pointing to the real entry.

Figure 2.13. An Alias Entry Points to Another Directory Entry



To create an alias entry in the directory, you must first create an entry with the object class `alias` and an attribute named `aliasedObjectName`. The value of the `aliasedObjectName` attribute must be the DN of the entry you want this alias to point to.

In general, we recommend that you avoid the use of aliases. Because aliases can point to any directory entry, even one that is on a different server, aliases may exact a severe performance penalty. Consider the directory trees shown in [Figure 2.13](#). Alias entries in one of the trees point to entries in the other tree, which is housed in another server. To support searching across the entire `dc=example,dc=com` tree, Server A must contact Server B each time an alias entry is encountered while the search operation is being serviced.

This requirement can significantly slow down searches. In addition, when an entry is deleted on one server, it might still be referred to by an alias on another server. The result can be a dangling reference, which must be cleaned up somehow.

Instead of using aliases, use referrals or LDAP URLs in entries to point to the information you need to reference. More information on using referrals can be found in [Chapter 10](#), Topology Design.

The LDAP Functional Model

Now that you understand the LDAP information and naming models, you need some way to access the data stored in the directory tree. The LDAP functional model describes the operations that you can perform on the directory using the LDAP protocol.

The LDAP functional model consists of a set of operations divided into three groups. The *interrogation operations* allow you to search the directory and retrieve directory data. The *update operations* allow you to add, delete, rename, and change directory entries. The *authentication and control operations* allow clients to identify themselves to the directory and control certain aspects of a session.

In addition to these three main groups of operations, version 3 of the LDAP protocol defines a framework for adding new operations to the protocol via *LDAP extended operations*. Extended operations allow the protocol to be extended in an orderly fashion to meet new marketplace needs as they emerge. Extended operations were described earlier, in the section titled LDAP Extensibility.

The LDAP Interrogation Operations

The two LDAP interrogation operations allow LDAP clients to search the directory and retrieve directory data. The search operation allows a client to find entries in the directory, and the compare operation allows a client to test whether an entry contains a particular attribute value.

The LDAP Search Operation

The LDAP search operation is used to search the directory for entries and retrieve individual directory entries. There is no LDAP read operation. When you want to read a particular entry, you must use a form of the search operation in which you restrict your search to just the entry you want to retrieve. Later in the chapter we'll discuss how to search the directory and retrieve specific entries, as well as how to list all the entries at a particular location in the tree.

The LDAP search operation requires eight parameters:

1. Base object for the search
2. Search scope
3. Alias dereferencing options
4. Size limit
5. Time limit
6. Attributes-only parameter
7. Search filter
8. List of attributes to return

Base Object

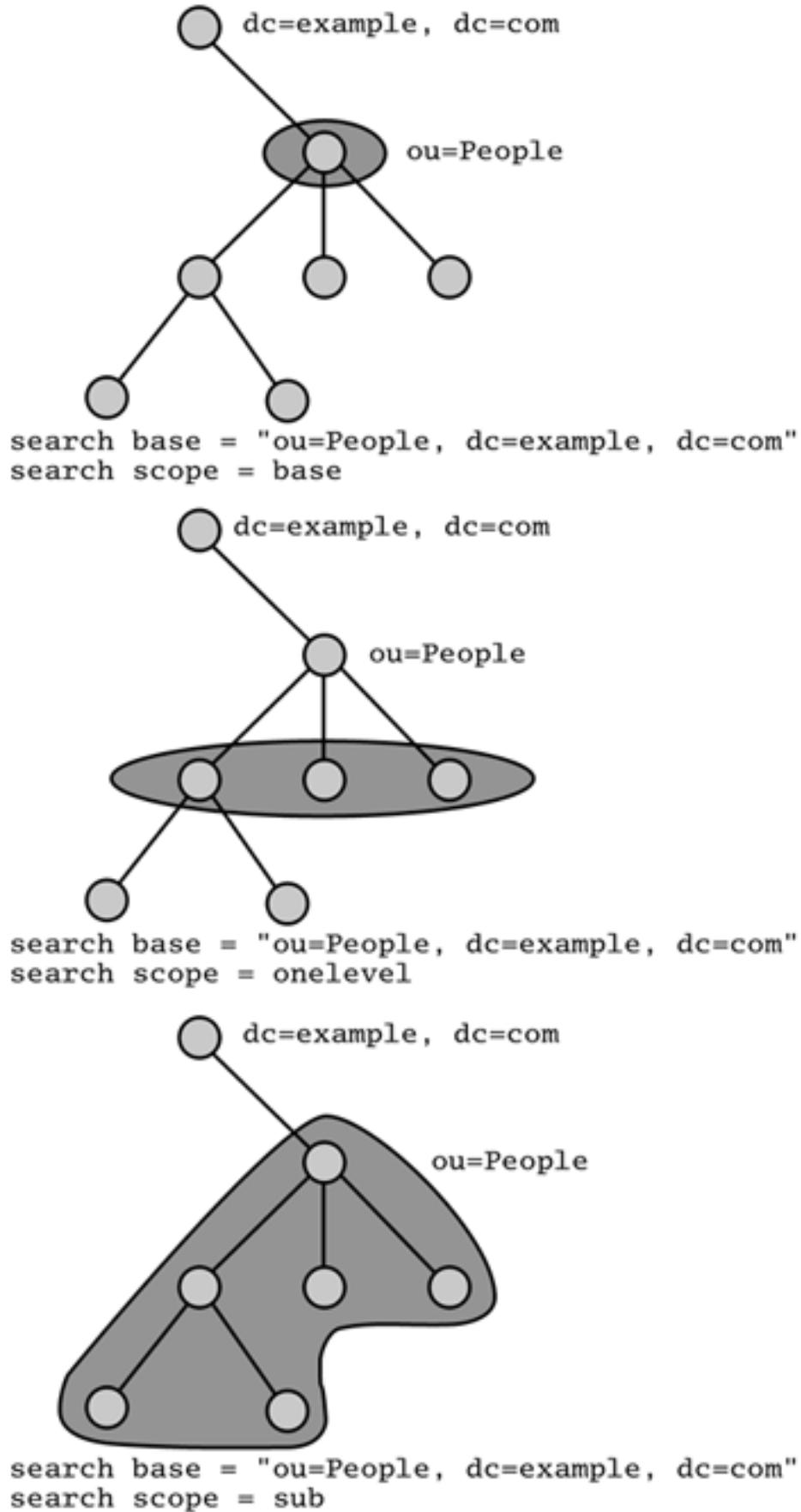
The first parameter is the base entry for the search (the terms *entry* and *object* are used interchangeably). This parameter, expressed as a DN, indicates the top of the tree you want to search.

Search Scope

The second parameter is the scope. There are three types of scope. A scope of **sub** (subtree) indicates that you want to search the entire subtree from the base object all the way down to the leaves of the tree. A scope of **onelevel** indicates that you want to search only the

immediate children of the entry at the top of the search base. A scope of `base` indicates that you want to limit your search to just the base object; this scope is used to retrieve one particular entry from the directory. [Figure 2.14](#) depicts the three search scope types. The base object and the search scope together define the area of the directory tree you want to search.

Figure 2.14. The Three Types of Search Scope



Alias Dereferencing Options

The third search parameter, `derefAliases`, tells the server whether aliases should be dereferenced when it is performing the search. This parameter has four possible values:

1. `neverDerefAliases`. Do not dereference aliases in searching or in locating the base object of the search.
2. `derefInSearching`. Dereference aliases in searching subordinates of the base object, but not in locating the base object of the search.
3. `derefFindingBaseObject`. Dereference aliases in locating the base object of the search, but not in searching subordinates of the base object.
4. `derefAlways`. Dereference aliases both in searching subordinates of the base object and in locating the base object of the search.

Size Limit

The fourth search parameter is the size limit. This parameter tells the server that the client is interested in receiving only a certain number of entries. For example, if the client passes a size limit of 100, but the server locates 500 matching entries, only the first 100 will be returned to the client, along with a result code of `LDAP_SIZELIMIT_EXCEEDED`. A size limit of 0 means that the client wants to receive all matching entries. (Note that servers may impose a maximum size limit that cannot be overridden by unprivileged clients.)

Time Limit

The fifth search parameter is the time limit. This parameter tells the server the maximum time in seconds that it should spend trying to honor a search request. If the time limit is exceeded, the server will stop processing the request and send a result code of `LDAP_TIMELIMIT_EXCEEDED` to the client. A time limit of 0 indicates that no limit should be in effect. (Note that servers may impose a maximum time limit that cannot be overridden by unprivileged clients.)

Attributes-Only Parameter

The sixth search parameter, `attrsOnly`, is a Boolean parameter. If it is set to `true`, the server will send only the attribute types to the client; attribute values will not be sent. This parameter can be used if the client is interested in finding out which attributes are contained in an entry but not in receiving the actual values. If this parameter is set to `false`, attribute types and values will be returned.

Search Filter

The seventh search parameter is the search filter, an expression that describes the types of entries to be returned. The filter expressions used in LDAP search operations are flexible; they are discussed in detail in the next section, The LDAP Search Filters.

Proposed Method for Retrieving All Operational Attributes

A proposal under review by the Internet Engineering Task Force (IETF) would allow the plus sign (+) to be used to request all operational attributes. If the proposal is adopted and implemented, clients will be able to request all operational attributes contained in an entry without knowing in advance the attribute names. For more information, see the Internet Draft *LDAPv3: All Operational Attributes* (<http://www.ietf.org/internet-drafts/draft-zeilenga-ldapv3bis-opattrs-06.txt>).

List of Attributes to Return

The eighth and final search parameter is a list of attributes to be returned for each matching entry. If this list is empty, all user attributes are returned. The special value * also means that all user attributes are to be returned, but it allows you to specify additional nonuser (operational) attributes that should be returned. (Without this special value, there would be no way to request all user attributes plus some operational attributes.)

Occasionally, you will want to verify that an entry exists but you won't be interested in retrieving any of the attributes. If you want to retrieve no attributes at all, you should specify the attribute name 1.1. [Table 2.2](#) provides some examples of attribute lists and the corresponding attributes returned by the server.

Note

Readers familiar with the ldapsearch command-line utility will note that it's not necessary to supply all the search parameters just discussed. The reason is that the ldapsearch utility provides default values for all options except the base object and the search filter.

Table 2.2. Examples of Attribute Lists and Corresponding Attributes Returned by the Server

Attribute List	Attributes Returned
cn, sn, givenName	cn, sn, and givenName only
*	All user attributes
1.1	No attributes
modifiersName	modifiersName only (an operational attribute)

`* , modifiersName`

All user attributes plus `modifiersName`

The LDAP Search Filters

An LDAP filter is a Boolean combination of attribute–value assertions. An attribute–value assertion consists of two parts: an attribute name and a value assertion, which you can think of as a value with wildcards allowed. The following sections look at the various types of search filters. We use the RFC 2254 notation for LDAP search filters, which is convenient because the filters are encoded as text strings (within the LDAP protocol itself, the filters are encoded with BER, as discussed earlier). This notation is also what you use when using the `ldapsearch` command-line utility, which will be discussed later in this chapter.

Equality Filters

An *equality filter* allows you to look for entries that exactly match a particular value. Here's an example:

`(sn=smith)`

This filter matches entries in which the `sn` (surname) attribute contains a value that is exactly `smith`. Because the equality matching rule associated with the `sn` attribute is `caseIgnoreMatch`, the case of the attribute and the filter is not important when matching entries are being located.

Substring Filters

When you use wildcards in filters, they are called *substring filters*. For example, the filter `(sn=smith*)` matches any entry that has an `sn` attribute value that begins with "smith". Entries with a surname of `Smith`, `Smithers`, `Smithsonian`, and so on will be returned.

Wildcards may appear anywhere in the filter expression, so the filter `(sn=*smith)` matches entries in which the surname ends with "smith" (for example, `Blacksmith`). The filter `(sn=smi*th)` matches entries in which the surname begins with "smi" and ends with "th", and the filter `(sn=*smith*)` matches entries that contain the string "smith" in the surname attribute. Note that the wildcard character matches zero or more instances of any character, so the filter `(sn=*smith*)` would match the entry with the surname Smith as well as any surnames in which the string "smith" was embedded.

Approximate Filters

In addition to the equality and substring filters, servers support an *approximate filter*. For example, on most directory servers, the filter `(sn~=jensen)` returns entries in which the surname attribute has a value that sounds like "jensen" (for example, `jenson`). Exactly how the server implements this filter is particular to each vendor and the languages supported by the server. Netscape Directory Server, for example, uses the metaphone algorithm to locate entries when an approximate filter is used. Internationalization also throws an interesting wrinkle into the concept of approximate matching; each language may need its own particular sounds—like algorithms. For example, algorithms used to implement approximate

matching for English are different from those for Japanese. It's likely that your directory software supports approximate matching for English but not for other languages.

"Greater Than or Equal To" and "Less Than or Equal To" Filters

LDAP servers also support "greater than or equal to" and "less than or equal to" filters on attributes that have some inherent ordering. For example, the filter (`sn<=Smith`) returns all entries in which the surname is less than or equal to `Smith` lexicographically. The ordering used depends on the matching rules associated with a particular attribute. The `sn` attribute, which is defined with the `caseIgnoreOrderingMatch` matching rule, is ordered lexicographically without respect to case. An attribute that has `INTEGER` syntax would be defined with a matching rule that would order values numerically. Attributes that have no inherent ordering, such as JPEG photos, cannot be searched for with this type of filter.

If you find that you need a greater-than or less-than filter (without the equals part), note that "greater than" is the complement of "less than or equal to" and "less than" is the complement of "greater than or equal to." In other words, `(age>21)` is equivalent to `(!(age<=21))`. Similarly, the filter `(age<21)`, which is also not a valid LDAP filter, is equivalent to `(!(age>=21))`.

In these cases, `!` is the negation operator, which we will discuss in more detail shortly.

Presence Filters

Another type of search filter is the *presence filter*. It matches any entry that has at least one value for the attribute. For example, the filter `(telephoneNumber=*)` matches all entries that have a telephone number.

Extensible Matching

The last type of search filter is the *extensible match filter*. It is supported only by LDAPv3 servers. The purpose of an extensible match filter is to allow new matching rules to be implemented in servers and used by clients. Recall our earlier example involving the `caseIgnoreMatch` rule. Each matching rule has an associated method for comparing values, depending on whether case is to be considered significant when values are being compared. When new attribute types are defined, it may also be necessary to define a new way of comparing values. Extensible matching also allows language-specific matching rules to be defined so that values in languages other than English can be meaningfully compared.

As an added benefit, extensible matching allows you to specify that the attributes that make up the DN of the entry should be searched. For example, using extensible matching you can locate all the entries in the directory that contain the attribute value assertion `ou=Engineering` anywhere in their DN. Without extensible matching, it's not possible to search on the individual components of the DN.

The syntax of an extensible matching filter is a bit complicated. It consists of five parts, three of which are optional:

1. An attribute name. If omitted, any attribute type that supports the given matching rule is compared against the value.
2. The optional string `:dn`, which indicates that the attributes forming the entry's DN are

to be treated as attributes of the entry during the search.

3. An optional colon and matching-rule identifier that identifies the particular matching rule to be used. If no matching rule is provided, the default matching rule for the attribute being searched should be used. If the attribute name is omitted, the colon and matching rule must be present.
4. The literal string "`:=`", used to separate the matching-rule identifier from the attribute value.
5. An attribute value to be compared against.

Formally, the grammar for the extensible search filter is

```
attr [":dn"] [":" matchingrule] "==" value
```

where

- `attr` is an attribute name.
- `matchingrule` is usually given by an object identifier (OID), although if a descriptive name has been assigned to the matching rule, that may be used as well. The OIDs of the matching rules supported by your directory server will be given in its documentation.
- `value` is an attribute value to be used for comparison.

Although LDAP largely does away with the mandatory use of OIDs, you will see them from time to time, especially if you use extensible matching rules or if you design your own schema extensions. The topic of extending your directory schema is discussed in [Chapter 8, Schema Design](#). Let's look at some examples of extensible matching filters:

- The following filter specifies that all entries in which the `cn` attribute matches the value `Barbara Jensen` should be returned:

```
(cn:1.2.3.4.5.6:=Barbara Jensen)
```

When comparing values, the matching rule given by the OID `1.2.3.4.5.6` should be used.

- The following filter specifies that all entries that contain the string `jensen` in the surname should be returned:

```
(sn:dn:1.2.3.4.5.7:=jensen)
```

Object Identifiers

Object identifiers, commonly referred to as OIDs, are unique identifiers assigned to objects. They are used to uniquely identify many different types of things, such as X.500 directory object and attribute types. In fact, just about everything in the X.500 directory system is identified by an OID. OIDs are also used to uniquely identify objects in other protocols, such as the Simple Network Management Protocol (SNMP).

OIDs are written as strings of dotted decimal numbers. Each part of an OID represents a node in a hierarchical OID tree. This hierarchy allows an arbitrarily large number of objects to be named, and it supports delegation of the namespace. For example, all the user attribute types defined by the X.500 standards begin with "2.5.4". The `cn` attribute is assigned the OID `2.5.4.3`, and the `sn` attribute is assigned the OID `2.5.4.4`.

An individual subtree of the OID tree is called an *arc*. Individual arcs may be assigned to organizations, which can then further divide the arc into *subarcs*, if so desired. For example, Netscape Communications Corporation has been assigned an arc of the OID namespace for its own use. Internally, it has divided that arc into subarcs for use by the various product teams. Delegating the management of the OID namespace in this fashion can prevent conflicts.

The X.500 protocol makes extensive use of OIDs to uniquely identify various protocol elements. LDAP, on the other hand, favors short, textual names for things: `cn` to describe the common name attribute and `person` to identify the `person` object class, for example. To maintain compatibility with X.500, LDAP allows a string representation of an OID to be used interchangeably with the short name for the item. For example, the search filters `(cn=Barbara Jensen)` and `(2.5.4.3=Barbara Jensen)` are equivalent. Unless you're working with an LDAP-based gateway into an X.500 system, you should generally avoid using OIDs in your directory-enabled applications.

The `sn` attributes within the DN are also searched. When comparing values, the matching rule given by the OID `1.2.3.4.5.7` should be used.

- The following filter returns any entries in which the `o` (organization) attribute exactly matches `Example` and any entry in which `o=Example` is one of the components of the DN:

`(o:dn:=Example)`

- The following filter returns any entries in which a DN component with a syntax appropriate to the given matching rule matches `Example`:

`(:dn:1.2.3.4.5.8:=Example)`

The matching rule given by the OID 1.2.3.4.5.8 should be used.

Negation

Any search element can be negated if the filter is preceded with an exclamation point (!). For example, the filter `(!(sn=Smith))` matches all entries in which the `sn` attribute does not contain the value `smith`, including entries with no `sn` attribute at all.

Combining Filter Terms

Filters can also be combined by AND and OR operators. The AND operator is signified by an ampersand (&), and the OR operator is signified by the vertical bar (|). When combining search filters, you use *prefix notation*, in which the operator precedes its arguments. Those familiar with the "reverse polish notation" common on Hewlett-Packard calculators will be familiar with this concept (although reverse polish is a postfix notation, not a prefix notation like that used in LDAP search filters).

Let's look at some examples of combinations of LDAP search filters. The filter `(&(sn=Smith)(L=Mountain View))` matches all entries with a surname of `smith` that also have an `L` (locality) attribute of `Mountain View`. In other words, this filter finds everyone named Smith in the Mountain View location. The filter `(| (sn=Smith) (sn=Jones))` matches everyone with a surname of `Smith` or `Jones`.

You use parentheses to group more complex filters to make the meaning of the filter unambiguous. For example, if you want to search the directory for all entries that have an e-mail address but do not have a telephone number, you use the following filter:

```
(&(mail=*)(!(telephoneNumber=*)))
```

Note that the parentheses bind the negation operator to the presence filter for telephone number.

Technically speaking, parentheses are always required, even if the filter consists of only a single term. Some LDAP software allows you to omit the enclosing parentheses and inserts them for you before sending the search request to the server. However, if you are developing your own software using one of the available SDKs, you need to include the enclosing parentheses.

[Table 2.3](#) summarizes the six types of search filters and the three Boolean operators.

Escaping in Search Filters

If you need to search for an attribute value that contains one of five specific characters, you need to substitute the character with an escape sequence consisting of a backslash and a two-digit hexadecimal sequence representing the character's value. [Table 2.4](#) shows the characters that must be escaped, along with the escape sequence you should use for each. For example, to search for all entries in which the `cn` attribute exactly matches the value `A*Star`, you use the filter `(cn=A\2AStar)`.

Note that the rules for escaping search filters and the rules for escaping distinguished names

are different and not interchangeable.

Table 2.3. Types of LDAP Search Filters

Filter Type	Format	Example	Matches
Equality	(attr=value)	(sn=jensen)	Surnames exactly equal to <code>jensen</code>
Substring	(attr=[leading]* [any]*[trailing])	(sn=*jensen*) (sn=jensen*)	Surnames containing the string "jensen" Surnames starting with the string "jensen"
		(sn=*jensen)	Surnames ending with the string "jensen"
		(sn=jen*s*en)	Surnames starting with "jen", containing an "s", and ending with "en"
Approximate	(attr~=value)	(sn~=jensin)	Surnames approximately equal to <code>jensin</code> (for example, surnames that sound like "jensin"—note the misspelling)
Greater than or equal to	(attr>=value)	(sn>=Jensen)	Surnames lexicographically greater than or equal to <code>Jensen</code>
Less than or equal to	(attr<=value)	(sn<=Jensen)	Surnames lexicographically less than or equal to <code>Jensen</code>

Presence	(attr=*)	(sn=*)	All surnames
AND	(&(filter1) (filter2)...))	(&(sn=Jensen) (objectclass=person))	Entries with an object class of <code>person</code> and surname exactly equal to <code>Jensen</code>
OR	((filter1) (filter2)...))	((sn~="Jensin") (sn="jensin"))	Entries with a surname approximately equal to <code>Jensin</code> or a surname ending in "jensin"
NOT	(!(filter))	(!(mail=*))	All entries without a <code>mail</code> attribute

Table 2.4. Characters That Must Be Escaped If Used in a Search Filter

Character Sequence	Decimal Value	Hex Value	Escape Sequence
* (asterisk)	42	0x2A	\2A
((left parenthesis)	40	0x28	\28
) (right parenthesis)	41	0x29	\29
\ (backslash)	92	0x5c	\5C
NUL (the null byte)	0	0x00	\00

Common Types of Searches

Although the LDAP search operation is flexible, some types of searches you'll use more frequently than others:

- **Retrieving a single entry.** To retrieve a particular directory entry, use a scope of `base`, a search base equal to the DN of the entry you want to retrieve, and a filter of `(objectclass=*)`. The filter, which is a presence filter on the `objectclass` attribute,

will match any entry that contains at least one value in its `objectclass` attribute. Because every entry in the directory must have an `objectclass` attribute, this filter is guaranteed to match any directory entry. Because you've specified a scope of `base`, only one entry will be returned by the search (if the entry exists at all). This is how you use the search operation to read a particular entry.

- **Listing all entries directly below an entry.** To list all the directory entries at a particular level in the tree, use the same filter (`objectclass=*`) as when retrieving a particular entry, but use a scope of `onelevel` and a search base equal to the DN just above the level you want to list. All the entries immediately below the search base entry will be returned. The search base entry itself is not returned in a `onelevel` search. (The search base entry *is* returned in a `base` or `sub` search if it matches the search filter.)
- **Searching for matching entries within a subtree.** Another common search operation looks within a subtree of the directory for all entries that match particular search criteria. To perform this type of search, use a filter that selects the entries you're interested in retrieving—or (`objectclass=*`) if you want all entries—along with a scope of `sub` and a search base equal to the DN of the entry at the top of the tree you want to search.

Hiding LDAP Filters from Users

You might justifiably be thinking that your users will never be able to understand LDAP filter syntax. The prefix notation it uses is hardly intuitive, after all! Bear in mind, though, that any good directory access GUI hides the details of filter construction from end users.

Instead of requiring users to type raw LDAP filters, a set of pop-up menus and text boxes is typically used to allow the user to specify the search criteria, and the GUI client constructs the filter for the user. In [Figure 2.15](#), for example, Netscape Communicator's **Search** window uses the provided information to construct the filter `(&(cn=*Smith*) (L=*Dearborn*))`.

Figure 2.15. A GUI Interface for Searching the Directory



If you are a directory administrator, it's a good idea to become familiar with LDAP filter syntax. You can use this knowledge to provide complex "canned" queries for your end users, for example. Filter syntax also crops up in LDAP URLs and configuration files. Spending a little time understanding filter syntax is well worth the effort.

The Compare Operation

The second of the two interrogation operations, the LDAP compare operation, is used to check whether a particular entry contains a particular attribute value. The client submits a compare request to the server, supplying a DN, an attribute name, and a value. The server returns an affirmative response to the client if the entry named by the DN contains the given value in the given attribute type. If not, a negative response is returned.

It may seem odd that the compare operation even exists. After all, if you want to determine whether a particular entry contains a particular attribute value, you can just perform a search with a search base equal to the DN of the entry, a scope of **base**, and a filter expressing the test you want to conduct. If the entry is returned, the test was successful; if no entry is returned, the test was not successful.

The reasons that the compare operation exists are mainly historical and related to LDAP's roots in X.500. In only one case do the compare and search operations behave differently. If a comparison is attempted on an attribute but the attribute is not present in the entry, the compare operation returns a special indication to the client that the attribute does not exist. The search operation, on the other hand, simply does not return the entry in such cases. This capability to distinguish between "the entry has the attribute but contains no matching value" and "the entry does not have the attribute at all" may be convenient in some situations. The other advantage of the compare operation is that it is more compact in terms of the number of protocol bytes exchanged between the client and the server.

The LDAP Update Operations

LDAP has four update operations: add, delete, rename (modify DN), and modify. These four operations define the ways that you can manipulate the data in your directory.

The Add Operation

The add operation allows you to create new directory entries. It has two parameters: the distinguished name of the entry to be created, and a set of attributes and attribute values that will constitute the new entry. For the add operation to complete successfully, four conditions must be met:

1. The parent of the new entry must already exist in the directory.
2. There must not be an entry of the same name.
3. The new entry must conform to the schema that is in effect.
4. Access control must permit the operation.

If all these conditions are met, the new entry is added to the directory.

The Delete Operation

The delete operation removes an entry from the directory. It has a single parameter: the DN of the entry to be deleted. For the delete operation to complete successfully, three conditions must be met:

1. The entry to be deleted must exist.
2. The entry to be deleted must have no children.
3. Access control must permit the entry to be deleted.

If all these conditions are met, the entry is removed from the directory.

The Rename (Modify DN) Operation

The rename, or modify DN, operation is used to rename and/or move entries in the directory. It has four parameters: the DN of the entry to be renamed, the new RDN for the entry, an optional argument giving the new parent of the entry, and the delete-old-RDN flag. For the modify DN operation to succeed, the following conditions must be met:

- The entry being renamed must exist.
- The new name for the entry must not already be in use by another entry.
- Access control must permit the operation.

If all these conditions are met, the entry is renamed and/or moved.

If the entry is to be renamed but will still have the same parent entry, the new-parent argument is left blank. Otherwise, the new-parent argument gives the DN of the container where the entry is to be moved. The delete-old-RDN flag is a Boolean flag that specifies whether the old RDN of the entry is to be retained as an attribute of the entry or removed. [Figures 2.16](#) through [2.20](#) show the various combinations of renaming and moving entries that can be performed with the modify DN operation.

Figure 2.16. Renaming an Entry without Moving It

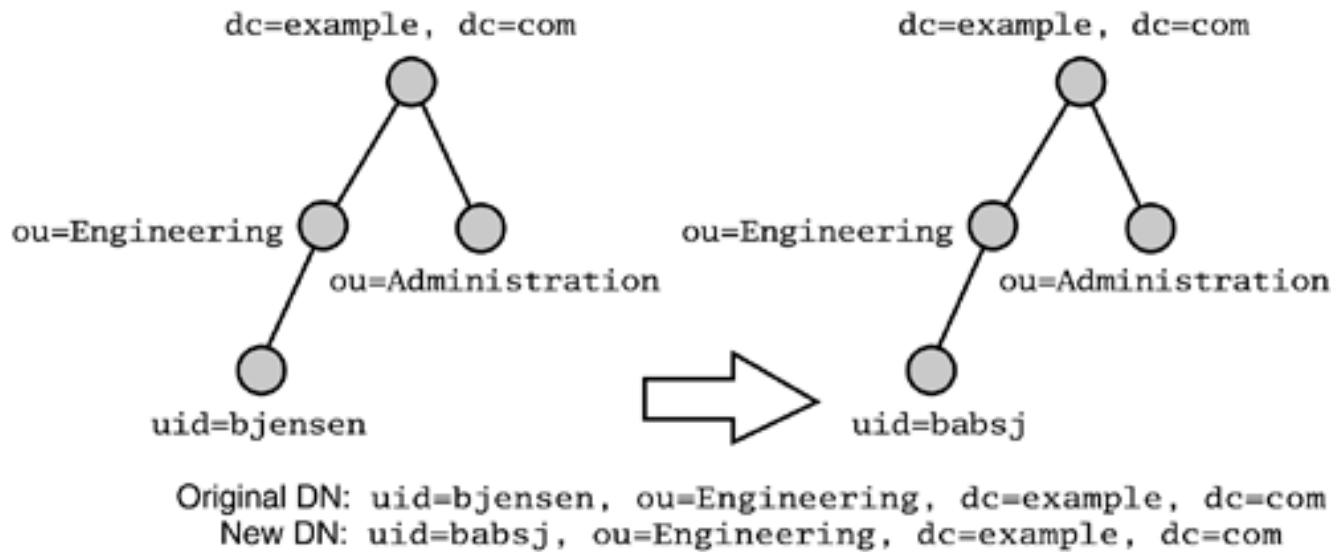
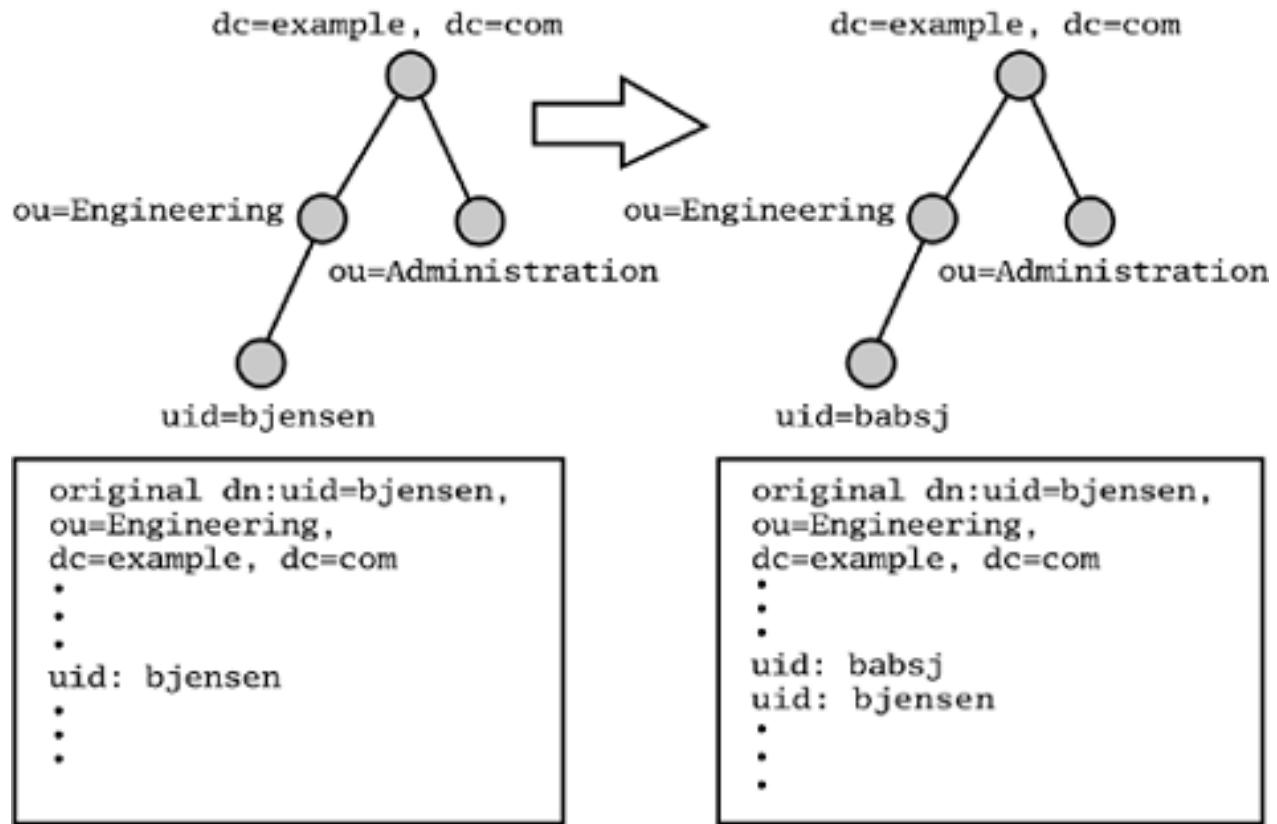


Figure 2.20. Renaming an Entry, `deleteoldrdn=false`



LDAPv2 did not have a modify DN operation; it had only a modify RDN operation. As the name implies, modify RDN allows only the RDN of an entry to be changed. This means that an LDAPv2 server may rename an entry but may not move it to a new location in the tree. To accomplish a move with LDAPv2, you must copy the entry, along with any child entries underneath it, to the new location in the tree and delete the original entry or entries.

Figure 2.17. Moving an Entry without Changing Its RDN

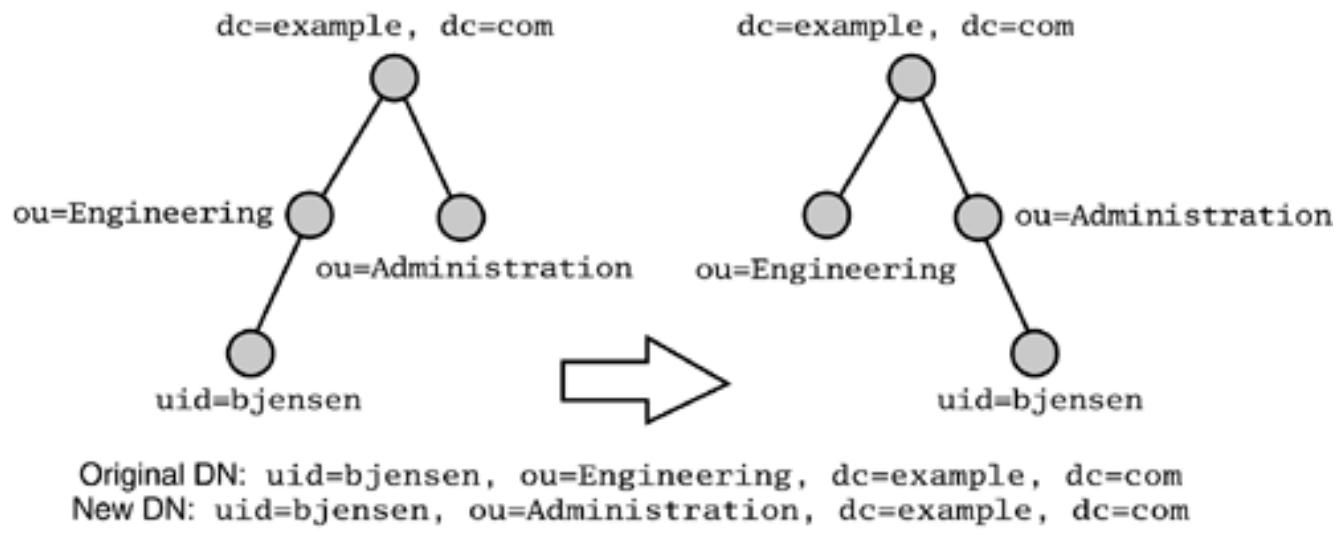
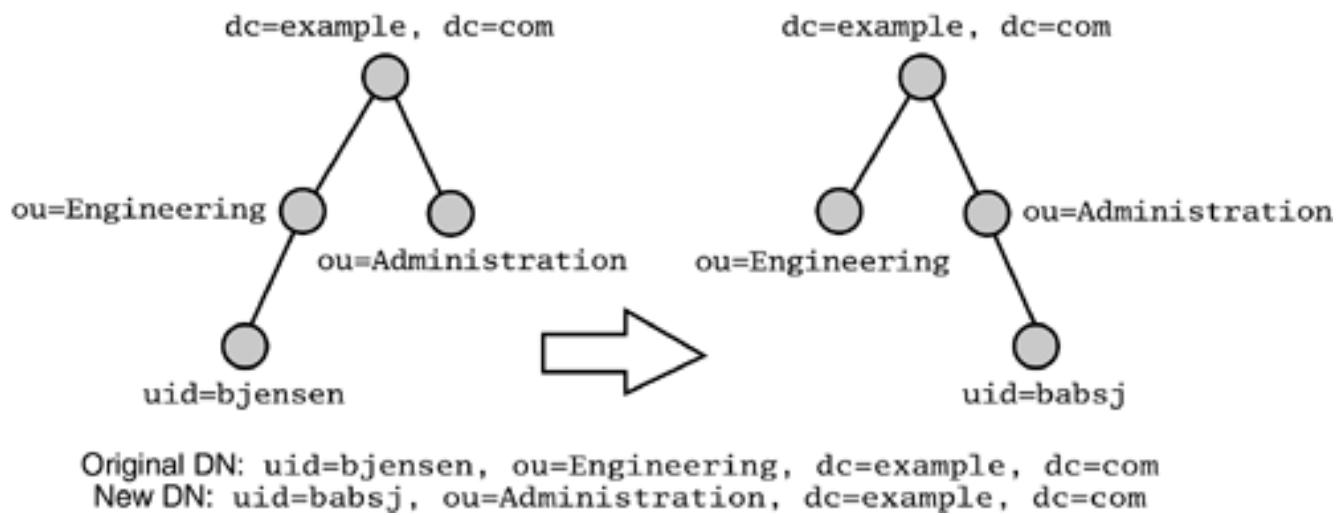


Figure 2.18. Moving an Entry and Changing Its RDN Simultaneously



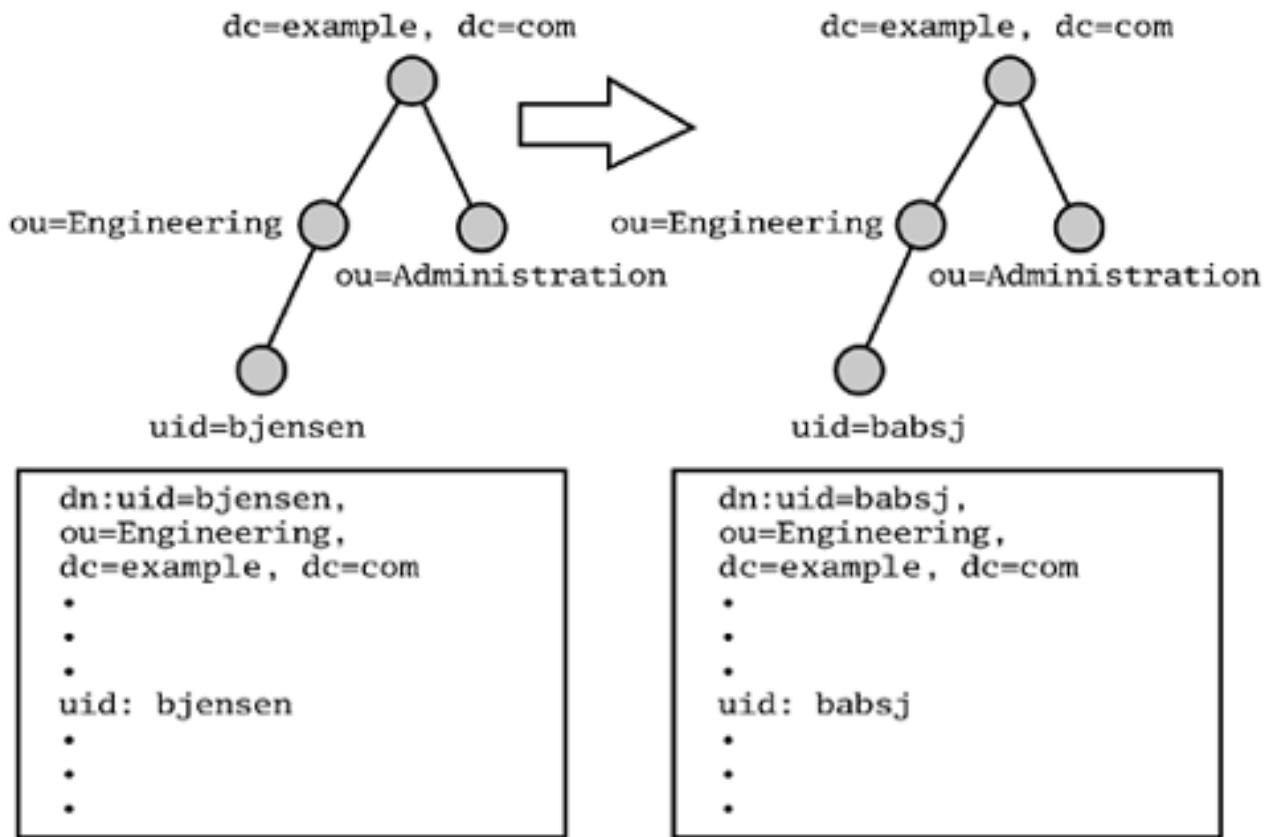
The Modify Operation

The modify operation allows you to update an existing directory entry. It takes two parameters: the DN of the entry to be modified and a set of modifications to be applied. These modifications can specify that new attribute values are to be added to the entry, that specific attribute values are to be deleted from the entry, or that all attribute values for a given attribute are to be replaced with a new set of attribute values. The modify request can include as many attribute modifications as needed.

For the modify operation to succeed, the following conditions must be met:

- The entry to be modified must exist.
- All the attribute modifications must succeed.

Figure 2.19. Renaming an Entry, `deleteoldrdn=true`



- The resulting entry must obey the schema that is in effect.
- Access control must allow the update.

If all these conditions are met, the entry is modified. Note that all the modifications must succeed, or else the entire operation fails and the entry is not modified. This requirement prevents inconsistencies that might arise from half-completed modify operations.

This last point raises one additional but important topic about the LDAP update operations: Each operation is *atomic*, meaning that the whole operation is processed as a single unit of work. Either this unit completely succeeds, or no modifications are performed. For example, a modify request that affects multiple attributes within an entry cannot half-succeed, with certain attributes updated and others not updated. If the client receives a success result from the server, then all the modifications were applied to the entry. If the server returns an error to the client, then none of the modifications were applied.

The LDAP Authentication and Control Operations

LDAP has two authentication operations (bind and unbind) and one control operation (abandon).

The Bind Operation

By providing a DN and a set of credentials, a client can use the bind operation to authenticate itself to the directory. The server checks whether the credentials are correct for the given DN and, if they are, notes that the client is authenticated as long as the connection remains open or until the client reauthenticates. The server can grant privileges to the client on the basis of its identity.

There are several different types of bind methods. In a simple bind, the client presents a DN and a password in cleartext to the LDAP server. The server verifies that the password matches the password value stored in the `userPassword` attribute of the entry and, if so,

returns a success code to the client.

The simple bind does send the password over the network to the server in the clear. However, you can protect against eavesdroppers intercepting passwords by encrypting the connections using Secure Sockets Layer (SSL) or TLS, which are discussed in the next section, The LDAP Security Model.

LDAPv3 also includes a new type of bind operation: the SASL bind. SASL is an extensible, protocol-independent framework for performing authentication and negotiation of security parameters. With SASL, the client specifies the type of authentication protocol it wants to use. If the server supports the authentication protocol, the client and server perform the agreed-on protocol.

For example, the client could specify that it wants to authenticate using the DIGEST-MD5 SASL mechanism. If the server implements DIGEST-MD5 (all LDAPv3-compliant servers must), it constructs a challenge and sends it to the client. The client computes the response to the challenge and sends the response to the server, which verifies the response and returns a final result to the client. DIGEST-MD5 authentication, if properly implemented, is immune to eavesdroppers.

Incorporation of SASL into LDAPv3 means that new authentication methods, such as smart cards or biometric authentication, can be easily implemented for LDAP without the protocol having to be revised.

It's perfectly legal for a client to bind, perform some operations, bind again, and perform more operations. If it does, all operations performed after the client rebinds are performed with the new bind identity. In the event that a bind operation fails, the client is treated as if it has bound anonymously. We discuss anonymous binds shortly, in the section titled The LDAP Security Model.

Note

In the Active Directory Services Interface (ADSI) SDK, the meaning of the word *bind* is slightly different. When you bind to a directory entry using ADSI, you not only authenticate to the directory; you also set the base object for subsequent directory searches.

The Unbind Operation

The second authentication operation is the unbind operation. The unbind operation has no parameters. When a client issues an unbind operation, the server discards any authentication information it has associated with the client's connection, terminates any outstanding LDAP operations, and disconnects from the client, thus closing the TCP connection.

Although it's considered good practice for a client to issue an unbind operation before disconnecting, server implementations must behave properly if the client disconnects without unbinding.

The Abandon Operation

The abandon operation has a single parameter: the message ID of the LDAP operation to abandon. The client issues an abandon operation when it is no longer interested in obtaining

the results of a previously initiated operation. Upon receiving an abandon request, the server terminates processing of the operation that corresponds to the message ID. The abandon request, typically used by GUI clients, is sent when the user cancels a long-running search request.

Note that it's possible for the abandon request (coming from the client) and the results of the abandoned operation (going to the client) to pass each other in flight. The client needs to be prepared to receive (and discard) results from operations that it has abandoned but that the server sent anyway. If you are using an LDAP SDK, however, you don't need to worry about this; the SDK takes care of this housekeeping for you.

The LDAP Security Model

We've discussed three of the four LDAP models so far. We have a set of directory entries, which are arranged into a hierarchy, and a set of protocol operations that allow us to authenticate to, search, and update the directory. All that remains is to provide a framework for protecting the information in the directory from unauthorized access. This is the purpose of the LDAP security model.

The security model relies on the fact that LDAP is a connection-oriented protocol. In other words, an LDAP client opens a connection to an LDAP server and performs various protocol operations on the same connection. The LDAP client may authenticate to the directory server at some point during the lifetime of the connection, at which point it may be granted additional (or fewer) privileges. For example, a client might authenticate as a particular identity that has been granted read/write access to all the entries in the directory. Before this authentication, it has a limited set of privileges (usually a default set of privileges extended to all users of the directory). After it authenticates, however, it is granted expanded privileges as long as the connection remains open.

What exactly is *authentication*? From the client's perspective, it is the process of proving to the server that the client is a particular entity. In other words, the client asserts that it has a certain identity and provides some credentials to prove this assertion. From the server's perspective, the process of authentication involves accepting the identity and credentials provided by the client and checking whether they prove that the client is who it claims to be.

To illustrate this abstract concept with a concrete example, let's examine how LDAP simple authentication works. In *simple authentication*, an LDAP client provides to an LDAP server a DN and a password, which are sent to the server in the clear (not hashed or encrypted in any way). The server locates the entry in the directory corresponding to the DN provided by the client and checks whether the password presented by the client matches the value stored in the `userPassword` attribute of the entry. If it does, the client is authenticated; if it does not, the authentication operation fails and an error code is returned to the client.

Note

In Netscape Directory Server, a hashed version of the password can be stored instead of the cleartext password. The hash is computed with one of several cryptographic one-way hash algorithms. When servicing a simple bind operation, the server takes the cleartext password provided by the client in the bind operation, hashes it, and compares it to the hashed password stored in the database. If the hashes match, the bind operation succeeds. Because the hash operation is one-way, it is difficult to determine the password if given only the hash. Storing passwords in hashed form improves the security of your directory somewhat.

The process of authenticating to the directory is called *binding*. An identity is bound to the connection when the bind operation achieves a successful authentication. If a client does not authenticate, or if it authenticates without providing any credentials, the client is bound anonymously. In other words, the server has no idea who the client is, so it grants a default set of privileges to the client. Usually this default set of privileges is minimal. In some instances, the default set of privileges is completely restrictive: No part of the directory may be read or searched. How you treat anonymously bound clients is up to the directory administrator and depends on the security policy appropriate to your organization. You can find more information on security and privacy in [Chapter 12](#), Privacy and Security Design.

Many different types of authentication systems are independent of LDAP. LDAPv2 supported only simple authentication, in which a DN and password are transmitted in the clear from the client to the server.

Note

The statement that LDAPv2 supported only simple authentication is not completely correct because LDAPv2 also supported Kerberos version 4 authentication, which does not require that passwords be sent in the clear. However, Kerberos v4 was not commercially successful and was superseded by Kerberos version 5. Kerberos support was therefore dropped from the core LDAPv3 protocol, although it's entirely feasible to support it via a SASL mechanism.

Acknowledging the need to support many different authentication methods, LDAPv3 has adopted the SASL framework. SASL provides a standard way for multiple authentication protocols to be supported by LDAPv3. Each type of authentication system corresponds to a particular SASL mechanism. A SASL mechanism is an identifier that describes the type of authentication protocol being supported.

After the server verifies the identity of the client, it can choose to grant additional privileges on the basis of a site-specific policy. For example, you might have a policy that, when authenticated, enables users to search the directory but does not enable them to modify their own directory entries. Or you might have a more permissive policy that allows some authenticated users to modify certain attributes of their own entries, whereas other users (your administrative staff) may modify any attribute of any entry. The way you describe the access rights, the entities to which those rights are granted, and the directory entries to which those rights apply are collectively called *access control*.

LDAPv3 Authentication Methods

During work on LDAPv3, it became clear that it was necessary to define the minimum set of authentication methods that must be supported by LDAPv3 servers. Without such a definition, a client from one vendor and a server from another vendor might not have any authentication methods in common other than simple authentication. Recall that LDAP simple authentication sends passwords in the clear over the network, with no encryption. Allowing such a situation to exist would severely limit LDAP's adoption.

To address this problem, the LDAP Working Group defined authentication methods whose implementation was mandatory in RFC 2829, *Authentication Methods for LDAP*. In this document, LDAP servers are broken down into three distinct groups, with separate requirements for each:

1. Read-only public directory servers may allow anonymous authentication (no password).
2. Servers that support password-based authentication *must* support the DIGEST-MD5 SASL mechanism documented in RFC 2831, *Using Digest Authentication as a SASL Mechanism*.
3. Servers that require session protection (encryption) and authentication must implement the StartTLS extended operation, as defined in RFC 2830, *Lightweight Directory Access Protocol (v3): Extension for Transport Layer Security*. This extended operation allows an LDAP client to request encryption of all data flowing between it and the server, and it allows the client and server to authenticate each other using public key certificates. The StartTLS extended operation is described in detail shortly, in the section titled Transport Layer Security (TLS).

Of course, any server can use a more secure authentication method than is strictly required. For example, servers that support password-based authentication may also support StartTLS. The intent of RFC 2829 is to ensure that any LDAPv3-compliant client can authenticate to any LDAPv3-compliant server in a secure fashion, without sending passwords in the clear.

Access Control Models

It may come as somewhat of a disappointment to learn that LDAP does not currently define a standard access control model. However, this does not mean that individual LDAP server implementations have no access control model. In fact, any commercially successful server software must have such a model.

Netscape Directory Server, for example, has a rich access control model. The model works by describing what a given identity can do to a particular set of entries, with granularity down to the attribute level. For example, with the Netscape server it is possible to specify an access control instruction (ACI) that allows a person to modify only the **description** attribute of his or her own entry. Or the model can allow you to grant complete rights to the directory to all persons in a particular group. This approach allows easy creation of a set of directory administrators; a given person's rights can be easily revoked by removal of the person from the group. The model is fully documented in the *Netscape Directory Server Administrator's Guide*.

The IETF continues to work on defining a standard access control model and a standard syntax for representing access control rights. The promise for the future is that you, as a directory deployer, will be able to deploy directory servers from several vendors and implement a consistent security policy across those servers—whether they cooperate to serve a distributed directory or are replicas of each other. Unfortunately, that is not the case today. You would be wise to document your access control policy in plain language so that you can adapt it to whatever model and syntax emerge from the standards bodies in the future.

Transport Layer Security (TLS)

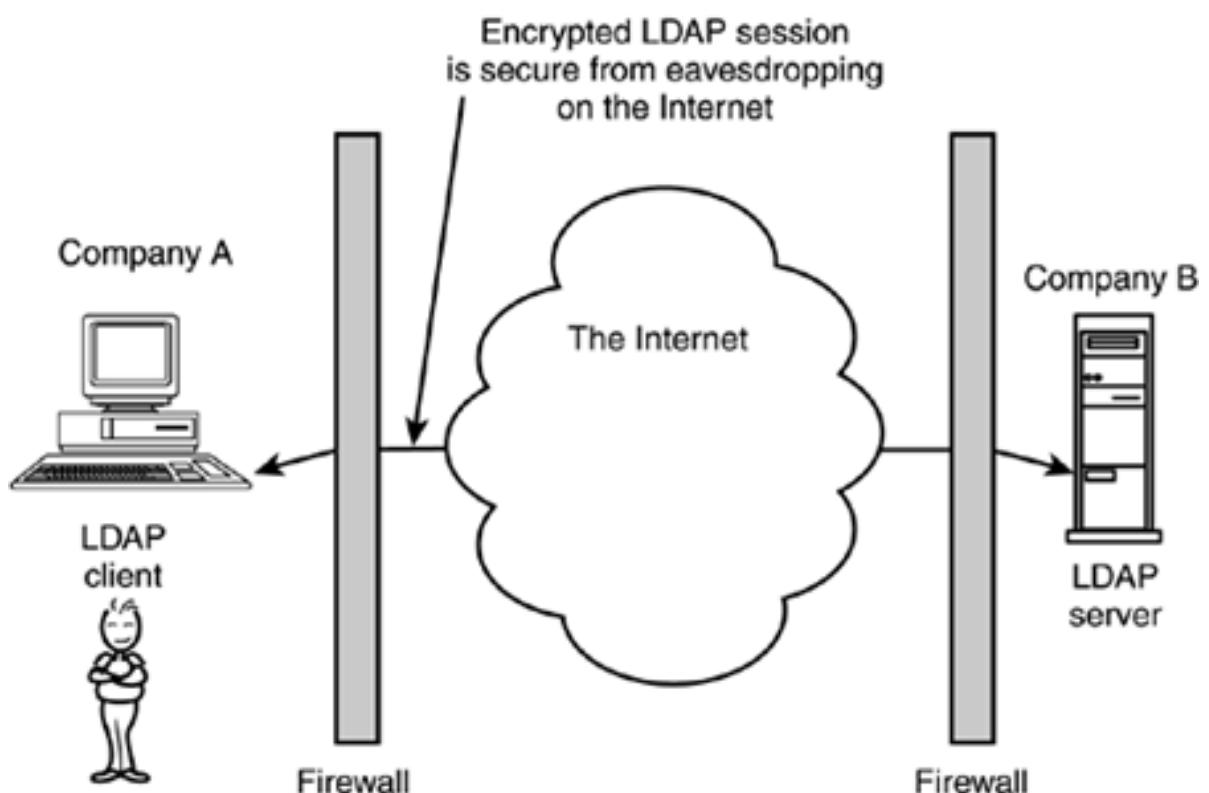
TLS is a security technology that supports privacy, data integrity, and encryption for connection-oriented protocols like TCP. Clients that use TLS to communicate with a server

- Can be confident that communications are immune to eavesdropping
- Can be confident that communications are immune to tampering (so-called man-in-the-middle attacks)
- Can authenticate to the server, using a public key certificate
- Can verify the authenticity of the server to which they have connected, by verifying the server's public key certificate

SSL has been a successful technology for the World Wide Web, securing electronic commerce and other transactions that depend on transmission of data being hidden from eavesdroppers. TLS, the follow-up to SSL, is an emerging Internet standard. LDAP offers a standard way for clients to begin encrypting all data flowing to and from LDAP on the connection using TLS.

Just as SSL and TLS enabled a new class of applications on the Web, they will enable new uses of directory technology. For example, two companies in a trading-partner relationship can allow directory queries from their trading partners to travel over the Internet. Because TLS encrypts these queries and the results, each company can rest assured that the directory data is protected while in transit over the Internet. [Figure 2.21](#) depicts this scenario.

Figure 2.21. TLS Allows Secure Transmission of Directory Data over the Internet



Readers familiar with existing directory server implementations, such as Netscape Directory Server, will point out that there already is a way to use SSL with LDAP. Many server implementations support the use of LDAP-over-SSL, known as LDAPS. Servers that implement LDAPS must provide this service on a TCP port distinct from the normal LDAP service. Typically, a server listens on port 389 for LDAP connections and port 636 for LDAPS connections.

By contrast, TLS allows a client to begin a connection without encryption, and to negotiate encryption and authentication after the connection is established. This means that an LDAP server supporting TLS can support both types of clients (secure and nonsecure) on the same TCP port.

The StartTLS extended operation, defined in RFC 2830, *Lightweight Directory Access Protocol (v3): Extension for Transport Layer Security*, is the means by which an LDAP client indicates to an LDAP server that TLS should be used on an existing LDAP connection. After an LDAP client initiates TLS, it can bind through use of the the SASL EXTERNAL mechanism. The server typically maps the certificate provided by the client to a directory entry using an

implementation-specific method (for example, Netscape Directory Server maps certificates to entries using a method configurable via the `certmap.conf` configuration file). An LDAP client usually goes through the following steps to establish a secure, authenticated connection to a directory server:

Step 1. Open a TCP connection to the server.

Step 2. Send a StartTLS extended operation. Lower-layer protocols then negotiate encryption and authentication according to the TLS specification.

Step 3. Bind using the SASL EXTERNAL mechanism if a certificate was provided during TLS negotiation, or using another SASL mechanism, such as DIGEST-MD5.

After TLS has been established and a bind operation performed, the client has a secure, authenticated connection to the directory.

Team LiB

◀ PREVIOUS

NEXT ▶

LDIF

LDAP Data Interchange Format is a standard text-based format for describing directory entries, defined in RFC 2849. LDIF allows you to export your directory data and import it into another directory server, even if the two servers use different internal database formats. In the database/spreadsheet world, the tab-delimited format performs a similar function: It provides a simple format that virtually all spreadsheets and databases can import and export.

There are two different types of LDIF files. The first type describes a set of directory entries, such as your entire corporate directory, or perhaps a subset of it. The other type of LDIF file is a series of LDIF update statements that describe changes to be applied to directory entries. In the following sections we'll look at both types in detail.

LDIF Representation of Directory Entries

[Listing 2.1](#) shows two directory entries in LDIF format.

Listing 2.1 A Typical LDIF File

```
version: 1

dn: uid=bjensen, ou=people, dc=example, dc=com
objectclass: top
objectclass: person
objectclass: organizationalPerson
objectclass: inetOrgPerson
cn: Barbara Jensen
cn: Babs Jensen
givenName: Barbara
sn: Jensen
uid: bjensen
mail: bjensen@example.com
telephoneNumber: +1 408 555 1212
description: Manager, Switching Products Division

dn: uid=ssmith, ou=people, dc=example, dc=com
objectclass: top
```

```
objectclass: person

objectclass: organizationalPerson

objectclass: inetOrgPerson

cn: Steve Smith

cn: Stephen Smith

givenName: Stephen

sn: Smith

uid:ssmith

mail: ssmith@example.com

telephoneNumber: +1 650 555 1212

description: Member of Technical Staff.
```

An individual entry expressed in LDIF format consists of two parts: a DN and a list of attribute values. The DN, which must be the first line of the entry, is composed of the string "dn" followed by a colon (:) and the distinguished name of the entry. After the DN come the attributes of the entry. Each attribute value is composed of an attribute type, a colon (:), and the attribute value. Attribute values may appear in any order; for readability, however, we suggest that you list the `objectclass` values for the entry first and group multiple values for the same attribute type together, as in [Listing 2.1](#).

Any line in an LDIF file may be folded into multiple lines, which is typically done when an individual line is extremely long. To fold a line, insert a newline character and a space character into the value. Folding is not required, but some editors cannot handle extremely long lines. [Listing 2.2](#) shows an entry with a folded line; note how the `description` attribute is folded into four lines.

Listing 2.2 An LDIF File with a Folded Attribute Value

```
version: 1

dn: uid=bjensen, ou=people, dc=example, dc=com

objectclass: top

objectclass: person

objectclass: organizationalPerson

objectclass: inetOrgPerson

cn: Barbara Jensen

cn: Babs Jensen

givenName: Barbara
```

```
sn: Jensen  
uid: bjensen  
mail: bjensen@example.com  
telephoneNumber: +1 408 555 1212  
description: I will be out of the  
office from August 12, 2001, to September 10, 2001. If you need  
assistance with the Ostrich project, please contact Steve Smith  
at extension 7226.
```

If an LDIF file contains an attribute value or a DN that is not ASCII, that value or DN must be encoded in a special format called *base 64*. This encoding method can represent any arbitrary data as a series of printable characters. When an attribute is base 64-encoded, the attribute type and value are separated by two colons instead of a single colon. [Listing 2.3](#) shows an entry in LDIF format that contains a base 64-encoded binary attribute (`jpegPhoto`). Notice that, in addition to being base 64-encoded, the attribute is folded.

Listing 2.3 An Entry in LDIF Format Containing a Base 64–Encoded Attribute Value

```
dn: uid=bjensen, ou=people, dc=example, dc=com  
objectclass: top  
objectclass: person  
objectclass: organizationalPerson  
objectclass: inetOrgPerson  
cn: Barbara Jensen  
cn: Babs Jensen  
givenName: Barbara  
sn: Jensen  
uid: bjensen  
mail: bjensen@example.com  
telephoneNumber: +1 408 555 1212  
jpegPhoto:: /9j/4AAQSkZJRgABAAAAAQABAAD/2wBDABALDA4MChAODQ4  
SERATGCgaGBYWGDEjJR0oOjM9PDkzODdASFxOQERXRTc4UG1RV19iz2hnP  
k1xeXBkeFx1Z2P/2wBDARESEhgVGC8aGi9jQjhCY2NjY2NjY2NjY2N
```

jY2NjY2NjY2NjY2NjY2NjY2NjY2NjY2NjY2NjY2NjY2P/wAARCACcA
LgDASIAhEBAxEB/8QAHwAAAQUBAQEBAQEAAAAAAAAAAECAwQFBgcICQo
L/8QAtRAAAgEDAwIEAwUFBAQAAAF9AQIDAAQRBRlhMUEGE1FhByJxFDKBk
aEII0KxwRVS0fAkM2JyggkKFhcYGRolJicoKSo0NTY3ODk6Q0RFRkdISUp
TVFVWV1hZWmNkZWZnaGlqc3R1dnd4eXqDhIWGh4iJipKTlJWWl5iZmqKjp
KWmp6ipqrKztLW2t7i5usLDxMXGx8jJytLT1NXW19jZ2uHi4+Tl5uf06er
x8vP09fb3+

More formally, the syntax of an entry represented in LDIF format is as follows:

```
( "dn:" <DN of entry> | "dn::" <base 64-encoded DN of entry>)  
<attribute type> ("::" <attribute value> | "::" <base 64 attribute value>)  
...  
...
```

A complete formal definition of the LDIF syntax is available in RFC 2849.

LDIF Update Statements

The second type of LDIF file describes a set of changes to be applied to one or more directory entries. An individual LDIF update statement consists of a DN, a change type, and possibly a set of modifications. Typically you will use this type of LDIF file as input to a command-line utility such as the `ldapmodify` program, which is included with Netscape Directory Server and the Netscape LDAP SDK. The `ldapmodify` program reads the update statements, converts those statements to LDAP protocol operations, and sends them to a server for processing. For more information on using `ldapmodify` to apply updates to a directory server, see the section titled `The ldapmodify Command-Line Utility` later in this chapter.

Four types of changes can be described by an LDIF update statement. These change types correspond exactly to the types of update operations that can be performed over the LDAP protocol: add a new entry, delete an existing entry, modify an existing entry, and rename an existing entry. Although the examples in the following sections do not show either folding or base 64 encoding, both are permitted in LDIF update statements.

Adding a New Entry

The `add changetype` statement indicates that an entry is to be added to the directory. The syntax of this update statement is

```
dn: dn of entry to be added
```

```
changetype: add
```

```
attribute type: value
```

```
...
```

For example, you would use the following to add a new entry to the directory:

```
dn: uid=bjensen, ou=people, dc=example, dc=com
changetype: add
objectclass: top
objectclass: person
objectclass: organizationalPerson
objectclass: inetOrgPerson
cn: Barbara Jensen
cn: Babs Jensen
givenName: Barbara
sn: Jensen
uid: bjensen
mail: bjensen@example.com
telephoneNumber: +1 408 555 1212
```

Deleting an Entry

The `delete changetype` statement indicates that an entry is to be removed from the directory. The syntax of this type of update statement is

```
dn: dn of entry to be deleted
changetype: delete
```

For example, you would use the following to delete an entry from the directory:

```
dn: uid=bjensen, ou=people, dc=example, dc=com
changetype: delete
```

Modifying an Entry

The `modify changetype` statement indicates that an existing entry is to be modified. It also allows you to add new attribute values, remove specific attribute values, remove an attribute entirely, or replace all attribute values with a new set of values. The syntax of the `modify` update statement is

```
dn: dn of entry to be modified  
changetype: modify  
modifytype: attribute type  
[attribute type: attribute value]  
-  
...
```

Note that there is an additional operator: `modifytype`. This operator can be `add`, `delete`, or `replace`, and it is interpreted as follows.

To add one or more new attribute values, use an `add modifytype` statement and include the attribute values you want to add. The following example adds two new values to the `telephoneNumber` attribute; if values for this attribute already exist, they are unaffected:

```
dn: uid=bjensen, ou=people, dc=example, dc=com  
changetype: modify  
add: telephoneNumber  
telephoneNumber: +1 216 555 1212  
telephoneNumber: +1 408 555 1212
```

To delete one or more specific attribute values, use a `delete modifytype` statement and include the values you want to delete. The following example removes the value `+1 216 555 1212` from the `telephoneNumber` attribute; any other `telephoneNumber` attribute values are unaffected:

```
dn: uid=bjensen, ou=people, dc=example, dc=com  
changetype: modify  
delete: telephoneNumber  
telephoneNumber: +1 216 555 1212
```

To completely remove an attribute, use a `delete modifytype` statement, but do not include any specific attribute value to be deleted. The following example completely removes the

`telephoneNumber` attribute from the entry:

```
dn: uid=bjensen, ou=people, dc=example, dc=com
changetype: modify
delete: telephoneNumber
```

To replace an attribute with a new set of values, use a `replace modifytype` statement and include the values that should replace any existing attribute values. The following example replaces any existing values of the `telephoneNumber` attribute with the two given values:

```
dn: uid=bjensen, ou=people, dc=example, dc=com
changetype: modify
replace: telephoneNumber
telephoneNumber: +1 216 555 1212
telephoneNumber: +1 405 555 1212
```

Multiple `modifytype` statements can be combined into a single update statement. Each set of lines constituting one `modifytype` statement must be separated by a line that contains only a single dash. For example, the following update statement adds a new value to the `mail` attribute, removes a specific value from the `telephoneNumber` attribute, completely removes the `description` attribute, and replaces the `givenName` attribute with a new set of values:

```
dn: uid=bjensen, ou=people, dc=example, dc=com
changetype: modify
add: mail
mail: bjensen@example.com
-
delete: telephoneNumber
telephoneNumber: +1 216 555 1212
-
delete: description
-
replace: givenName
givenName: Barbara
```

```
givenName: Babs
```

-

When multiple modifications are included in a single LDIF update statement and the `ldapmodify` program sends the corresponding LDAP operations to an LDAP server, the server performs the update only if all the individual attribute modifications succeed. In the last example, if the entry did not contain the `telephoneNumber` attribute value `+1 216 555 1212`, it would not be possible to delete that specific value. The server treats each update statement as a single unit, so none of the attribute modifications would be made, and an error would be returned to the client.

Renaming and/or Moving an Entry

The `moddn changetype` statement indicates that an existing entry is to be renamed and optionally moved to a new location in the directory tree. The syntax of the `moddn` update statement is

```
dn: dn of entry to be modified  
changetype: moddn  
[newsuperior: dn of new parent]  
[deleteoldrdn: ( 0 | 1 )]  
[newrdn: new relative distinguished name for the entry]
```

If an entry's RDN is to be changed, the `newrdn` and `deleteoldrdn` parameters must be provided. If an entry is to be moved to a new location in the tree, the `newsuperior` parameter must be provided. Both operations can be performed at once; that is, an entry can have its RDN changed at the same time it is moved to a new location in the tree.

For example, to change an entry's name without moving it to a new location in the tree, you'd use the following:

```
dn: uid=bjensen, ou=People, dc=example, dc=com  
changetype: moddn  
newrdn: uid=babsj  
deleteoldrdn: 0
```

After this update was performed on the server, the entry would look like this:

```
dn: uid=babsj, ou=People, dc=example, dc=com
```

```
[other attributes omitted for brevity]
```

```
uid: babsj
```

```
uid: bjensen
```

Notice that the old RDN, `uid: bjensen`, is still present in the entry. When `0` is provided for the `deleteoldrdn` flag, the old RDN is retained as an attribute of the entry. If you want the old RDN to be removed from the entry, include `deleteoldrdn: 1` in your `moddn` update statement. If this were done, the entry would look like this after being renamed:

```
dn: uid=babsj, ou=People, dc=example, dc=com
```

```
[other attributes omitted for brevity]
```

```
uid: babsj
```

If you want to move an entry to a new location in the tree, you can use the `newsuperior` parameter to specify the DN of the entry to which you would like the entry to be moved. For example, if you want to move Babs's entry under the `Terminated Employees` organizational unit, you would use the following LDIF update statement:

```
dn: uid=bjensen, ou=People, dc=example, dc=com
```

```
changetype: moddn
```

```
newsuperior: ou=Terminated Employees, dc=example, dc=com
```

The `moddn changetype` statement may behave differently depending on whether the server supports LDAPv3. If the server supports only LDAPv2, the `newsuperior` parameter may not be used; LDAPv2 does not support moving an entry to a new location in the tree.

LDAP Server Software

When you deploy LDAP in an organization, you will install and run one or more LDAP servers. An LDAP server is a program that implements the LDAP protocol and manages a database that holds the actual directory data. Most LDAP servers also include software that helps you manage the directory. For example, Netscape Directory Server 6.0 consists of

- `ns-slapd`, the LDAP server itself
- A set of shared libraries (in Unix) or dynamic link libraries (DLLs, in Windows) that the server uses to support its database, multimaster replication, access control processing, internationalization support, encryption, and other features
- Command-line utility software that lets you import data, export data, back up your database, and perform other essential functions
- A console application that allows you to manage the directory through a graphical user interface
- An HTTP-to-LDAP gateway that provides a simple directory lookup and management interface that can be accessed from any Web browser

Directory server software from other vendors is structured in approximately the same way. Microsoft Active Directory, for example, includes the server, command-line utilities, and graphical tools for managing directory content and the operation of the directory. The details of installing, configuring, and managing a particular LDAP server implementation vary widely, however. [Chapter 4, Overview of Netscape Directory Server](#), looks at the feature set, as well as provides examples of how to accomplish some of these tasks with Netscape Directory Server.

LDAP Command-Line Utilities

Most commercial and open-source LDAP implementations come with a set of command-line utilities that allow you to query and manipulate your directory's contents. These tools are flexible, and with a little additional knowledge about shell scripting, you may find that the command-line utilities meet most of your basic directory access needs. In this section we'll describe the three tools you're likely to encounter and how to use them.

Versions of these tools are available from multiple sources. We'll describe the version of the tools bundled with Netscape Directory Server 6. Other versions have different parameters. Consult the documentation for your particular command-line utilities if you are not using the Netscape tools.

The examples we've provided assume that you have installed Netscape Directory Server 6 and have imported the sample LDIF file named `Example.ldif` provided with the server software. If you need to obtain a copy of the server software, you can download an evaluation copy from the Netscape Web site at <http://enterprise.netscape.com>.

The ldapsearch Command-Line Utility

The ldapsearch command-line utility allows you to search a directory server. Search parameters are supplied on the command line, and search results are output in LDIF format. A basic search operation might look like this:

```
ldapsearch -h ldap.example.com -s sub -b "dc=example,dc=com" "(cn=Barbara Jensen)"  
version: 1  
  
dn: uid=bjensen, ou=People, dc=example,dc=com  
cn: Barbara Jensen  
cn: Babs Jensen  
sn: Jensen  
givenName: Barbara  
objectClass: top  
objectClass: person  
objectClass: organizationalPerson  
objectClass: inetOrgPerson  
ou: Product Development  
ou: People  
L: Cupertino  
uid: bjensen  
mail: bjensen@example.com  
telephoneNumber: +1 408 555 1862  
facsimileTelephoneNumber: +1 408 555 1992  
roomNumber: 0209
```

In this example, we sent an LDAP search operation to the directory server running on host `ldap.example.com`. A search scope of subtree (`-s sub`) was used, with a search base of `dc=example,dc=com` (`-b "dc=example,dc=com"`). The search filter "`(cn=Barbara Jensen)`" indicates that we want to find all entries where the common name (`cn`) attribute exactly matches the string "Barbara Jensen". One matching entry was returned and displayed. If you are unfamiliar with the terms *search base*, *search filter*, or *search scope*, review the information on the LDAP search operation earlier in this chapter, in the section titled The LDAP Functional Model.

Notice that the search base (`dc=example,dc=com`) and the search filter (`cn=Barbara Jensen`) were enclosed in quotes. This is necessary because those arguments to the `ldapsearch` command contain spaces. If the quotation marks were not used, those arguments would be split by the command interpreter or shell and would confuse the command-line utility. Certain other characters, such as the asterisk, may have meaning to the shell. If you receive an error such as "ldap_search: Bad search filter," check whether your command line is missing some required quotation marks.

Retrieving a Single Entry

To retrieve a single entry, use a search base equal to the DN of the entry you want to retrieve and a search scope of `base`:

```
ldapsearch -h ldap.example.com -s base -b "uid=bjensen,ou=people,dc=example,dc=com" "(  
objectclass=*)"  
  
version: 1  
  
dn: uid=bjensen, ou=People, dc=example,dc=com  
  
cn: Barbara Jensen  
  
cn: Babs Jensen  
  
sn: Jensen  
  
givenName: Barbara  
  
objectClass: top  
  
objectClass: person  
  
objectClass: organizationalPerson  
  
objectClass: inetOrgPerson  
  
ou: Product Development  
  
ou: People  
  
L: Cupertino  
  
uid: bjensen  
  
mail: bjensen@example.com  
  
telephoneNumber: +1 408 555 1862  
  
facsimileTelephoneNumber: +1 408 555 1992  
  
roomNumber: 0209
```

Binding (Authenticating)

In the previous examples we did not supply an identity to authenticate as. This is called an *anonymous bind*. Most directories are configured to limit the types of data that may be viewed by clients that bind anonymously. To gain more access, you must bind to the directory as a specific user. You can perform simple authentication using the `-D bind dn` and `-w bind password` options. Note what happens when we add these two parameters to the previous `ldapsearch` command:

```
ldapsearch -h localhost -D "uid=bjensen,ou=people,dc=example,dc=com" -w hifalutin -s sub  
→ -b "dc=example,dc=com" "(cn=Barbara Jensen)"  
  
version: 1  
  
dn: uid=bjensen, ou=People, dc=example,dc=com  
  
cn: Barbara Jensen  
  
cn: Babs Jensen  
  
sn: Jensen  
  
givenName: Barbara  
  
objectClass: top  
  
objectClass: person  
  
objectClass: organizationalPerson  
  
objectClass: inetOrgPerson  
  
ou: Product Development  
  
ou: People  
  
L: Cupertino  
  
uid: bjensen  
  
mail: bjensen@example.com  
  
telephoneNumber: +1 408 555 1862  
  
homeTelephoneNumber: +1 408 555 9233  
  
facsimileTelephoneNumber: +1 408 555 1992  
  
roomNumber: 0209
```

Notice that the `homeTelephoneNumber` attribute is now returned. The reason is that the directory server has been configured with tighter access control restrictions on that attribute, and by authenticating, we now have access to it.

Besides simple authentication, the command-line utilities allow you to use SSL client authentication to bind to the directory (see the section [Using SSL to Search the Directory](#) later in this chapter).

Retrieving Only Certain Attributes

By default, the server returns all attributes of an entry except for operational attributes, which we'll discuss in a moment. What if you're interested in retrieving only some of the attributes available? You

can specify a space-separated list of attribute types at the end of the `ldapsearch` command line. For example, if you're interested in retrieving only the `mail` and `roomNumber` attributes of an entry, then you should include those attribute names at the end of the `ldapsearch` command line, as follows:

```
ldapsearch -h localhost -s sub -b "dc=example,dc=com" "(cn=Barbara Jensen)" mail  
↳ roomNumber  
  
version: 1  
  
dn: uid=bjensen, ou=People, dc=example,dc=com  
  
mail: bjensen@example.com  
  
roomNumber: 0209
```

Recall that operational attributes are not returned unless specifically requested. If you want to retrieve one or more operational attributes, you need to specify their names on the command line. For example, if you want to retrieve the `modifiersName` and `modifyTimeStamp` attributes of an entry to determine who last updated the entry and when, use the following command:

```
ldapsearch -h ldap.example.com -s sub -b "dc=example,dc=com" "(cn=Barbara Jensen)"  
↳ modifiersName modifyTimeStamp  
  
version: 1  
  
dn: uid=bjensen, ou=People, dc=example,dc=com  
  
modifiersName: cn=directory manager  
  
modifyTimeStamp: 20010719043240Z
```

What if you want to retrieve all user attributes and some operational attributes? Use the special attribute type `*` along with the names of the operational attribute types you wish to retrieve. Because the asterisk has special meaning to Unix and Windows command shells, you should enclose it in quotation marks:

```
ldapsearch -h localhost -s sub -b "dc=example,dc=com" "(cn=Barbara Jensen)" "*"  
↳ modifiersName modifyTimeStamp  
  
version: 1  
  
dn: uid=bjensen, ou=People, dc=example,dc=com  
  
cn: Barbara Jensen  
  
cn: Babs Jensen  
  
sn: Jensen  
  
givenName: Barbara  
  
objectClass: top
```

```
objectClass: person
objectClass: organizationalPerson
objectClass: inetOrgPerson
ou: Product Development
ou: People
L: Cupertino
uid: bjensen
mail: bjensen@example.com
telephoneNumber: +1 408 555 1862
facsimileTelephoneNumber: +1 408 555 1992
roomNumber: 0209
modifiersName: cn=directory manager
modifyTimeStamp: 20010719043240Z
```

More Complex Filters

In the preceding examples we used simple filters with a single term. What if you wanted to perform more complex searches? It's possible to use multiple filters and combine them with AND (`&`), OR (`|`), and NOT (`!`) operators, as described earlier in this chapter. For example, if you want to produce a list of all entries where the locality attribute matches either `cupertino` or `sunnyvale` you use the following filter:

```
ldapsearch -h localhost -s sub -b "dc=example,dc=com" "(|(L=cupertino)(L=sunnyvale))"
```

Note that the OR operator precedes the operands, and parentheses are used to group the operands. LDAP search filters use prefix notation, as described earlier.

Suppose that the previous search operation returned some entries representing printers, and you weren't interested in those. You could further limit the search to person objects by adding a term, `(objectclass=person)`:

```
ldapsearch -h localhost -s sub -b "dc=example,dc=com" "(&(|(L=cupertino)(L=sunnyvale))(
➥ objectclass=person))"
```

The prefix filter notation makes this difficult to read. In a more familiar infix notation, it means

```
(location = cupertino OR location = sunnyvale) AND objectclass = person
```

Using SSL to Search the Directory

If you want to use SSL to encrypt the communication between Idapsearch and the server, you can do so

by using the **-Z** and **-P** options. The **-Z** option enables SSL, and the **-P** option gives the path to the certificate database. The certificate database is consulted by the client to determine whether the certificate provided by the client is trusted. The certificate database format used by the command-line utilities is the same format as that used by Netscape Communicator 4.x and 6.x:

```
ldapsearch -Z -P /home/bjensen/.netscape/cert7.db -h localhost -D "uid=bjensen,ou=people,  
➥ dc=example,dc=com" -w hifalutin -s sub -b "dc=example,dc=com" "(cn=Barbara Jensen)"
```

In this example, we still provide the bind DN and password. SSL is being used only to encrypt the communications.

SSL can also be used to carry authentication credentials. If you have a client certificate generated by a certificate authority and installed into the certificate database, you can use SSL client authentication. With SSL client authentication, the client presents a digital certificate to the server. The server verifies the authenticity of the client's certificate. If verification succeeds, then the server attempts to map the certificate to an entry in the directory. If the mapping step succeeds, then the client is authenticated as the mapped entry. The actual mapping step in Netscape Directory Server is configured via the `certmap.conf` configuration file. For more information, consult the *Netscape Directory Server 6 Administrator's Guide*.

To use SSL client authentication, use the **-Z** and **-P** options as in the previous example, and add the **-W** option (key database password) and the **-N** option (certificate name to use). Here's an example:

```
ldapsearch -h localhost -Z -P /home/bjensen/.netscape/cert7.db -W "mycertdbpassword" -N  
➥ "My Certificate" -s sub -b "dc=example,dc=com" "(cn=Barbara Jensen)"
```

For more information on these options, see the next section, *ldapsearch Command-Line Option Reference*.

ldapsearch Command-Line Option Reference

[Table 2.5](#) is a comprehensive reference for all the `ldapsearch` command-line options.

Table 2.5. Options for the `ldapsearch` Command

Option	Description
-n	Show what would be done, but do not send any search requests to the server.
-v	Display verbose diagnostic output.
-h host	Connect to the LDAP server running on <code>host</code> . The default host is <code>localhost</code> .

-p port	Connect to the server running on <code>port</code> instead of the default port. By default, <code>ldapsearch</code> uses port 389, unless you use the <code>-z</code> option to request an SSL connection, in which case <code>ldapsearch</code> uses port 636.
-V n	Use LDAP protocol version <code>n</code> . The default is 3. If you're communicating with a server that understands only LDAPv2 you can use the <code>-V 2</code> option to force the utility to use LDAPv2. Values other than 2 or 3 are illegal.
-z	Use SSL when connecting to the server.
-P path	Specify the pathname to the SSL certificate database. The certificate database is needed so that the utility can determine whether the server's certificate is trusted. The certificate database must be in the standard Netscape Communicator format. If you're using Communicator on a Unix system, your certificate database is <code>\$HOME/.netscape/cert7.db</code> .
-N name	Specify the name of the certificate to use for SSL client authentication. If you have more than one certificate in your certificate database, the <code>-N</code> option allows you to specify which one should be used.
-K path	Specify the pathname to the key database used for SSL client authentication. The key database contains your encrypted private keys. If the <code>-P</code> option is given, then the default for the <code>-P</code> option is the file <code>key3.db</code> in the same directory as <code>cert7.db</code> .
-m path	Specify the path to the security module database, which contains the implementations of the security protocols. You will not normally need to provide this option, unless you're using a nonstandard security module. If the <code>-P</code> option is given, then the default for the <code>-m</code> option is the file <code>secmod.db</code> in the same directory as <code>cert7.db</code> .
-W passwd	Specify the password for the SSL key database. If you're using SSL client authentication, you must provide the password to unlock the key database so that the <code>ldapsearch</code> command-line utility can use your private key.
-D binddn	Bind as <code>binddn</code> , if using simple authentication.
-w passwd	Use <code>passwd</code> when binding with simple authentication.
-E	Request that the server expose (report) bind identity. When this option is used, the client sends an authentication request control to the server. This control requests that the server return an authentication response control that describes the actual bind identity assigned. This is useful when SSL client authentication or SASL is being used, when the server may map the client credentials (for example, a certificate) to a directory entry.

-R	Do not automatically follow referrals. By default, the command-line utility attempts to follow any referrals encountered. A referral is returned when the server does not hold the required entries—for example, when directory data is distributed across several servers.
-O hoplimit	If following referrals, do not follow more than <code>hoplimit</code> referrals. This option is useful in preventing the client from getting stuck chasing a circular referral reference.
-M	Manage references. A reference (a <code>ref</code> attribute in an entry) normally results in a referral being returned to the client. If you want to examine or update the referral information, use the <code>-M</code> option to send a <code>ManageDSAIT</code> control to the server. (See Chapter 3 , LDAPv3 Extensions, for more information.)
-0	Ignore version mismatches between the <code>ldapsearch</code> command-line utility and the LDAP shared library (Unix) or DLL (Windows). Note that this option is the character "0" (zero).
-i charset	Specify the character set for command-line input. By default, the input character set is the default locale (the character set specified by the <code>LANG</code> environment variable). This option affects only strings provided in the command line, such as the search base, bind DN, and so on. It does not affect LDIF file input, which must be in the UTF-8 character set.
-k dir	Specify a directory containing character set conversion routines. The default is the current directory. Netscape Directory Server 6 installs a set of conversion routines in the directory <code>lib/nls/conv31</code> below the installation directory.
-Y proxyid	Specify the authorization ID to use for the proxied authorization control. When the <code>-Y</code> option is used, the client attempts to impersonate the given ID. If the client has the appropriate permission, the server processes the client requests as if they had been performed by the impersonated identity. The authorization ID is of the form <code>dn: entrydn</code> , where <code>entrydn</code> is the DN of the entry to impersonate. For example, <code>dn: uid=bjensen, ou=people, dc=example, dc=com</code> .
	For more information on proxied authorization, see Chapter 3 , LDAPv3 Extensions.
-H	Display help text, with a short description of each option and its usage.
-t	Write values to files in the <code>temp</code> directory. When this option is used, the utility writes each value of each attribute into a separate file in the directory named by the user's <code>TMP</code> environment variable and prints the name of the file(s) to the standard output. This option is most useful when you want to save a binary attribute such as a JPEG file or X.509 certificate to a file for further processing.

-U	Display the location of files containing attribute values. When -U is used in conjunction with the -t option, the <code>ldapsearch</code> utility outputs file URLs describing the location of the files it just wrote. The file URLs are of the form <code>attrname:< file:///path</code> , where <code>attrname</code> is the name of the attribute written, and <code>path</code> is the path of the file containing the attribute value. The LDIF file written can be read by the <code>ldapmodify</code> utility, which opens the <code>file:///</code> URLs and sends the contents to the server.
-u	Include user-friendly names (UFNs) in the output. In addition to displaying the DN of an entry, the utility displays the user's DN without tags. For example, the DN <code>uid=bjensen, ou=people, dc=example, dc=com</code> will be displayed in UFN format as <code>bjensen, people, example, com</code> . User-friendly naming is a holdover from LDAP's X.500 roots and is not widely used today.
-o	Print directory data in an older format. Attribute types and values are separated with an equal sign (<code>=</code>), and long attribute values are not split into multiple lines.
-T	Don't fold (wrap) long lines. This option is useful when you want to export data to a file and modify that file. The -T option prevents long lines from being split into multiple lines, which can be confusing if <code>grep</code> , <code>sed</code> , or other tools are being used to perform global search and replace operations on an LDIF file. Normally, the <code>ldapsearch</code> utility splits lines longer than 72 characters into multiple lines.
-e	Minimize base 64 encoding of values. Normally, the <code>ldapsearch</code> utility encodes any attribute values that contain binary characters, newline characters, and leading or trailing spaces. With the -e option, the utility uses base 64 encoding only when the attribute value contains binary data.
-1	Omit the leading <code>"version: 1"</code> in LDIF output. The latest LDIF standards specify that the first line of the LDIF file should indicate the LDIF version. Some older utilities may not expect this. Use the -1 option to create an LDIF file that these utilities can understand.
-A	Retrieve attribute names only. The client will not retrieve or display attribute values.
-B	Print binary values. This option is useful only with the -o option. When printing attribute values in old format, the <code>ldapsearch</code> utility will normally not display non-ASCII values. The -B option overrides this behavior and causes the utility to print non-ASCII values.
-x	Perform sorting on the server. When this flag is provided, any sort specifications (see the -S option) will be processed on the server. For more information, see Chapter 3 , LDAPv3 Extensions.

-F sep	Print <code>sep</code> between the attribute type and value. The <code>-F</code> option is valid only when used with the <code>-o</code> option.
-S attr	Sort the results by the attribute <code>attr</code> . To reverse the sort order, precede the attribute name with a dash (<code>-S -sn</code> , for example). You may use multiple <code>-S</code> options to specify more than one sort key. Sort keys are processed in the order they appear on the command line. For example, to sort by surname and then given name, use the options <code>-S sn -S givenName</code> .
-a deref	Some directory server software supports aliases, entries that point to another entry. When the server encounters an alias, it can choose to dereference the alias or return the actual alias entry. The <code>-a</code> option specifies how aliases are to be dereferenced. You can control the dereferencing behavior separately for the two phases of the search operation. Phase 1 locates the base object of the search (the entry given in the <code>-b</code> option), and phase 2 returns the entries below the base object that match the search criteria. <code>deref</code> may be one of the following:
	<ul style="list-style-type: none"> ● <code>find</code>. Dereference aliases only when locating the base object of the search, but not when returning entries subordinate to the base object. ● <code>search</code>. Dereference aliases when returning the entries subordinate to the base object, but not when locating the base object itself. ● <code>never</code>. Never dereference aliases. Return the alias entries themselves. ● <code>always</code>. Always dereference aliases, no matter when they are encountered.
	Note that Netscape Directory Server does not support server-side dereferencing of aliases, so this option has no effect with that server.
-s scope	Search scope. One of <code>base</code> , <code>one</code> , or <code>sub</code> .
-l time	Time limit in seconds for searching. The server will terminate a search operation if it does not complete in <code>time</code> seconds. The server may impose a smaller time limit than the client may not increase.
-a size	Size limit in entries for searching. The server will return at most <code>size</code> entries to the client. The server may impose a smaller size limit than the client may not increase.
-G	Use the Virtual List View feature to restrict results to a particular subset of the search results. There are two forms of the arguments: <ul style="list-style-type: none"> ● <code>before:after:index:count</code>. <code>index</code> specifies the absolute offset from the beginning of the search result set, <code>before</code> tells the server how many entries before the index entry to return, and <code>after</code> tells the server how many entries after the index entry to

`before:after:index:count`

return. `count` is the total requested size of the result set. For example, the argument `-G 5:4:105:0` instructs the server to send entries 100 through 109 to the client (5 entries before index 105, 4 entries after).

or

- `before:after:value`. Locate the first entry matching `value`, and return `before` entries before the matched entry and `after` entries after the matched entry. For example, the argument `-G 0:9:j` instructs the server to find the first entry matching "j" and to return that entry and the next 9 entries.

`before:after:value`

The `-G` option is always used in conjunction with a server-side sort request. Therefore, the `-G` option must always be used with the `-S` option and the `-x` option.

For more information on the Virtual List View feature, see [Chapter 3](#), LDAPv3 Extensions.

The `ldapmodify` Command-Line Utility

The `ldapmodify` utility allows you to perform one or more updates against a directory server. The LDIF file describing the updates to be performed is read from the standard input, or from a file if the `-f` option is used. The `ldapmodify` command applies the updates in the order they appear in the file. If you haven't read the preceding section on [LDIF](#), you may want to do so now.

Many command-line options for `ldapmodify` are the same as for `ldapsearch`. All the options that specify the server host name, bind DN, passwords, and SSL options are the same.

Suppose that you need to change Barbara Jensen's e-mail address to `babs@example.com`. The following `ldapmodify` command line accomplishes this:

```
ldapmodify -h ldap.example.com -D "cn=directory manager" -w secret < updates.ldif
```

where `updates.ldif` contains

```
dn: uid=bjensen, ou=people, dc=example, dc=com
changetype: modify
replace: mail
mail: bjensen@example.com
```

The following example shows how you can use the `-f` option to achieve the same result:

```
ldapmodify -h ldap.example.com -D "cn=directory manager" -w secret -f updates.ldif
```

Adding Entries

By default, the `ldapmodify` utility expects that you will provide LDIF update statements as input (LDIF update statements always contain a `changetype` line in every entry). If the input file contains LDIF entries to be added to the directory, and those entries do not have a `changetype` line, you can use the `-a` option on the command line. To illustrate, the following two examples produce the same result:

```
ldapmodify -h ldap.example.com -D "cn=directory manager" -w secret < updates.ldif
```

where `updates.ldif` contains

```
version: 1

dn: uid=bjensen, ou=people, dc=example, dc=com

changetype: add

objectclass: top

objectclass: person

objectclass: organizationalPerson

objectclass: inetOrgPerson

cn: Barbara Jensen

cn: Babs Jensen

givenName: Barbara

sn: Jensen

uid: bjensen

mail: bjensen@example.com

telephoneNumber: +1 408 555 1212

description: Manager, switching products division
```

is equivalent to

```
ldapmodify -h ldap.example.com -D "cn=directory manager" -w secret -a < updates.ldif
```

where `updates.ldif` contains

```
version: 1

dn: uid=bjensen, ou=people, dc=example, dc=com

objectclass: top

objectclass: person
```

```

objectclass: organizationalPerson
objectclass: inetOrgPerson
cn: Barbara Jensen
cn: Babs Jensen
givenName: Barbara
sn: Jensen
uid: bjensen
mail: bjensen@example.com
telephoneNumber: +1 408 555 1212
description: Manager, switching products division

```

Continuous Mode (**-c**) and Rejects File (**-e**) Options

Normally, the `ldapmodify` command stops if it encounters an error. You can instruct the utility to continue in the event of errors by using the **-c** (continuous mode) option. If you do this, you can also use the **-e** option to write any rejected LDIF update statements to a separate file. You can then fix the problem that caused the entries to be rejected and just apply the updates that were previously rejected. The following example continues if errors are encountered and writes any rejected entries to the file `rejects.ldif`:

```

ldapmodify -h ldap.example.com -D "cn=directory manager" -w secret -c -e rejects.ldif <
➥ updates.ldif

```

ldapmodify Command-Line Option Reference

[Table 2.6](#) is a comprehensive reference for all the `ldapmodify` command-line options.

Table 2.6. Options for the `ldapmodify` Command

Option	Description
-n	Show what would be done, but do not send any update requests to the server. The -n option can be used to verify that the LDIF file is correctly formatted.
-v	Display verbose diagnostic output.
-h	Connect to the LDAP server running on <code>host</code> . The default host is <code>localhost</code> .

- P Connect to the server running on `port` instead of the default port. By default, ldapmodify uses port 389, unless you use the `-Z` option to request an SSL connection. In this case, ldapmodify uses port 636.
- V n Use LDAP protocol version `n`. The default is `3`. If you're communicating with a server that understands only LDAPv2, you can use the `-V 2` option to force the utility to use LDAPv2. Values other than `2` or `3` are illegal.
- Z Use SSL when connecting to the server.
- P path Specify the pathname to the SSL certificate database. The certificate database is needed so that the utility can determine whether the server's certificate is trusted. The certificate database must be in the standard Netscape Communicator format. If you're using Communicator on a Unix system, your certificate database is `$HOME/.netscape/cert7.db`.
- N name Specify the name of the certificate to use for SSL client authentication. If you have more than one certificate in your certificate database, the `-N` option allows you to specify which one should be used.
- K path Specify the pathname to key database used for SSL client authentication. The key database contains your encrypted private keys. If the `-P` option is given, then the default for the `-P` option is the file `key3.db` in the same directory as `cert7.db`.
- m path Specify the path to the security module database. The security module database contains the implementations of the security protocols. You will not normally need to provide this option, unless you are using a nonstandard security module. If the `-P` option is given, then the default for the `-m` option is the file `secmod.db` in the same directory as `cert7.db`.
- W passwd Specify the password for the SSL key database. If you're using SSL client authentication, you must provide the password to unlock the key database so that the ldapsearch command-line utility can use your private key.
- D binddn Bind as `binddn`, if using simple authentication.
- w passwd Use `passwd` when binding with simple authentication.
- E Request that the server expose (report) bind identity. When this option is used, the client sends an authentication request control to the server. This control requests that the server return an authentication response control that describes the actual bind identity assigned. This is useful when using SSL client authentication or SASL, when the server may map the client credentials (for example, a certificate) to a directory entry.
- R Do not automatically follow referrals. By default, the command-line utility attempts to follow any referrals encountered. A referral is returned when the server does not hold the required entries—for example, when directory data is distributed across several servers.

-O hoplimit If following referrals, do not follow more than `hoplimit` referrals. This option is useful in preventing the client from getting stuck chasing a circular referral reference.

-M Manage references. A reference (a `ref` attribute in an entry) normally results in a referral being returned to the client. If you want to examine or update the referral information, use the **-M** option to send a `ManageDSAIT` control to the server. (See [Chapter 3](#), LDAPv3 Extensions, for more information.)

-0 Ignore version mismatches between the `ldapsearch` command-line utility and the LDAP shared library (Unix) or DLL (Windows). Note that this option is the character "0" (zero).

-i charset Specify the character set for command-line input. By default, the input character set is the default locale (the character set specified by the `LANG` environment variable). This option affects only strings provided in the command line, such as the search base, bind DN, and so on. It does not affect LDIF file input, which must be in the UTF-8 character set.

-k dir Specify the directory containing character set conversion routines. The default is the current directory. Netscape Directory Server 6 installs a set of conversion routines in the directory `lib/nls/conv31` below the installation directory.

-Y proxyid Specify the authorization ID to use for the proxied authorization control. When the **-Y** option is used, the client attempts to impersonate the given ID. If the client has the appropriate permission, the server processes the client requests as if they have been performed by the impersonated identity. The authorization ID is of the form `dn:entrydn`, where `entrydn` is the DN of the entry to impersonate—for example, `dn:uid=bjensen, ou=people, dc=example, dc=com`.

For more information on proxied authorization, see [Chapter 3](#), LDAPv3 Extensions.

-H Display help text, with a short description of each option and its usage.

-c Specify that continuous mode is to be used. If an error is encountered while the directory is being updated, continue. By default, the `ldapmodify` tool stops when an error is returned by the server.

-A Display non-ASCII values in conjunction with **-v**.

-f file Read LDIF from `file` instead of standard input.

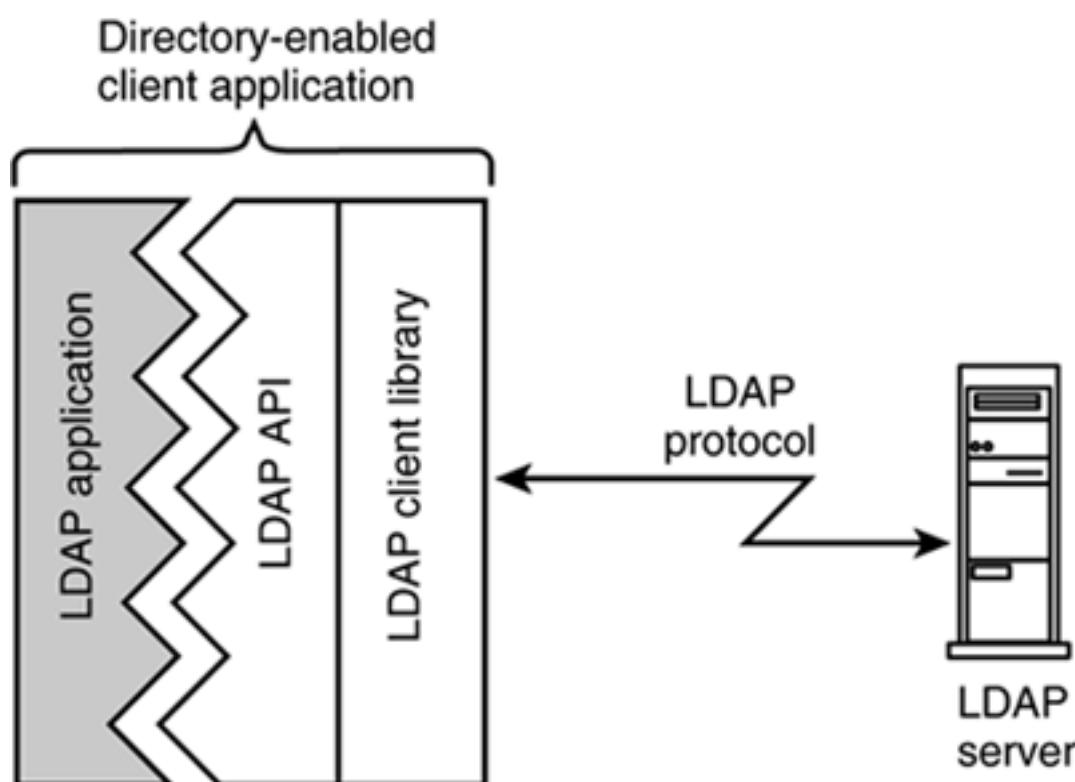
-a Add entries. Assume that the LDIF file contains only entries (no `changetype` lines).

- b** Read values that start with "/" from files. For example, the attribute value
`Jpegphoto: /tmp/file001.jpg`
causes the `ldapmodify` utility to open the file `/tmp/file001.jpg` and send its contents to the server.
- F** Force application of all changes, regardless of replica lines.
- e rejfile** Save rejected entries to file `rejfile`. You can fix the problem that caused the entries to be rejected and then use `rejfile` as input to `ldapmodify` to send just the rejected entries.
- B suffix** Bulk import to `suffix`. Completely replace the contents of `suffix` with the LDIF input file. The LDIF input file must contain LDIF entries, without `changetype` lines. For more information, see [Chapter 3](#), LDAPv3 Extensions.
- q** Be quiet when adding or modifying entries. Do not produce diagnostic output.

LDAP APIs

Early on, the developers of LDAP realized that directory-enabled applications would be created much more quickly if a standard API existed for accessing and updating the directory. The original LDAP distribution from the University of Michigan (often referred to as the U-M LDAP release; see [Chapter 1](#), Directory Services Overview and History) included a C programming library and several sample client programs built on this library. For a while, the C API included in the U-M distribution was the only API/SDK available. With the current industry momentum behind LDAP, however, the number of SDKs is increasing, and additional SDKs are becoming available. (We will discuss these additional SDKs later in this section and in [Chapter 21](#), Developing New Applications.) [Figure 2.22](#) shows how the LDAP SDK fits into a directory-enabled client application.

Figure 2.22. The LDAP API Provides a Common Interface to an LDAP Client Library SDK



The LDAP C API for LDAPv2 is documented in RFC 1823, and a proposed C API for LDAPv3 is in draft form at this time (available from the IETF Web site at <http://www.ietf.org>). The C API document simply defines the API calls and their semantics.

To obtain an SDK, you need to download one from any of the following sources:

- The OpenLDAP Project includes an LDAPv2/LDAPv3 SDK and is available in source code from <http://www.openldap.org>.
- An LDAPv2/LDAPv3 C SDK is available free of charge in binary form from Sun Microsystems at <http://wwws.sun.com/software/download/developer>. The Mozilla LDAP C SDK, upon which the Netscape SDK is based, is available in source code form from <http://www.mozilla.org/directory>.
- Microsoft provides an LDAPv2/LDAPv3 SDK with Windows 2000. Documentation is available from the MSDN site at <http://msdn.microsoft.com>.

All the C SDKs can, of course, be used from a C++ program.

Overview of the LDAP C API

The LDAP C API defines a set of core functions, listed in [Table 2.7](#), that map almost one to one onto the LDAP protocol operations. These APIs provide an asynchronous interface to the directory; that is, the calls are used to initiate a protocol operation to the server, and the `ldap_result()` call is used later to collect results from the previously initiated operations. This capability allows a client to issue multiple protocol requests or perform other work, such as updating window contents, while the operation is in progress on the server.

The API also provides a synchronous interface, in which the API calls are blocked until all results are returned from the server. The synchronous calls are generally simpler to use and are appropriate for simple command-line clients and multithreaded applications.

In addition to the API calls listed in [Table 2.7](#) and their synchronous counterparts, the LDAP C API (1) defines a set of utility routines that can be used to parse returned results from the server; (2) iterates over sets of entries, attributes, and attribute values; and (3) performs other useful operations. For a complete description of the various API calls available in the SDK you're using, consult the documentation.

Table 2.7. The Main LDAP C API Functions

Function	Description
<code>ldap_search()</code>	Searches for directory entries
<code>ldap_compare()</code>	Tests whether an entry contains a given attribute value
<code>ldap_bind()</code>	Authenticates (proves your identity) to a directory server
<code>ldap_unbind()</code>	Terminates an LDAP session
<code>ldap_modify()</code>	Makes changes to an existing directory entry
<code>ldap_add()</code>	Adds a new directory entry
<code>ldap_delete()</code>	Deletes an existing directory entry
<code>ldap_rename()</code>	Renames an existing directory entry (this function is named <code>ldap_modrdn()</code> in LDAPv2-only SDKs)

`ldap_result()` Retrieves the results of one of the previous operations

A useful reference book that covers the LDAP C API in detail and offers general advice on building directory-enabled applications was written by two of the authors of this book. It is called *LDAP: Programming Directory-Enabled Applications with Lightweight Directory Access Protocol*, by Tim Howes and Mark Smith, published in 1997 by Macmillan Technical Publishing. Note that that book describes the LDAPv2 C API calls only, and not the updated LDAPv3 API calls.

Other LDAP APIs

In addition to the various implementations of the C API, four other APIs are available:

1. Netscape has developed an LDAPv2 and LDAPv3 Java API that, like the LDAP C API, has a close mapping onto the LDAP protocol. The Java API specification, currently in draft form, is available from the IETF Web site at <http://www.ietf.org>. An SDK that implements the draft API is available from <http://wwws.sun.com/software/download/developer> and, like the C SDK, is available in source code form at <http://www.mozilla.org>. Online documentation is also available. The Java classes that implement the Netscape SDK are also included with versions of Netscape Communicator currently being shipped.
2. Perl fans can use Net::LDAP, available from <http://www.cpan.org>, or PerLDAP, available from <http://www.mozilla.org>.
3. Python programmers can use the python-ldap module, available from <http://python-ldap.sourceforge.net>.
4. JavaSoft has developed the proprietary Java Naming and Directory Interface (JNDI). This API/SDK defines a common interface for accessing various different directory systems from a Java application or applet. Additional types of directory systems and protocols can be supported through the development of additional service provider interfaces (SPIs) for JNDI. This feature allows a JNDI client to access several distinct directory services, such as NIS, DNS, LDAP, NDS, or X.500. JNDI is available from JavaSoft at <http://www.javasoft.com>.
5. Microsoft also has a proprietary, object-oriented SDK, called ADSI (Active Directory Services Interface), for accessing multiple directory systems. ADSI APIs are available for Visual Basic, C, and C++. For more information on ADSI, see <http://www.microsoft.com>.

LDAP and Internationalization

Directory services, by their very nature, span language boundaries. Multinational companies might have offices in dozens of countries, each with a distinct language. Electronic commerce sites might have customers in many different countries. To address this growing need, LDAPv3 has been designed so that it can easily support multiple languages.

LDAPv3 uses the UTF-8 (UCS Transformation Format 8) character set for all textual attribute values and distinguished names. UTF-8 is a standard character coding system that can represent text in virtually all written languages in use today. It is defined and developed by the Unicode Consortium, an industry group.

There are two important points to understand about UTF-8. First, because of the way UTF-8 is designed, ASCII data is also valid UTF-8 data, giving it the advantage of being highly compatible with existing English-language directory data; no work needs to be done to transform the data into valid UTF-8.

The second point is that when you use UTF-8, it becomes unnecessary to declare an attribute value to be in a particular character set. In other systems, values must be tagged with their character set (for example, Latin-1, Shift-JIS) so that the data may be interpreted correctly. However, because the UTF-8 character set contains codes for the glyphs of virtually all languages, in this format such tagging is unnecessary. It's even possible to use multiple languages within a single attribute value.

Because LDAPv3 servers can store text in multiple languages, it is useful to have some way to store and access attributes by language type. For example, in an international corporation with offices in the United States and Japan, it may be desirable to store several representations of a Japanese employee's name in the directory, including a version in Japanese and a version in English. The LDAP Extension (LDAPEXT) Working Group in the IETF has proposed a method for accomplishing this through the use of language codes.

A language code is an option on an LDAP attribute name. Separated from the base attribute name with a semicolon, it gives the particular language for the attribute in a standard format. For example, the attribute type `cn;lang-fr` refers to a common name in the French language, and the attribute type `sn;lang-ja` refers to a surname in the Japanese language. All language names are represented by two-character codes defined in ISO Standard 639, *Code for the Representation of Names of Languages*.

The LDAP language code standard also allows names to be represented in a particular regional dialect or usage of a particular language. For example, there are some minor differences in how the English language is written in the United States and the United Kingdom. Whereas the language code `lang-en-US` identifies an attribute in the U.S. dialect, the language code `lang-en-GB` indicates the British dialect. The country codes used to specify the region are defined in ISO Standard 3166, *Codes for the Representation of Names of Countries*.

An LDAP client may use language codes in search filters and attribute lists. In other words, an LDAP client may limit its search to only those attributes in the specific language it is interested in, and by specifying language codes in the list of attributes to be returned, it may request that only specific languages be returned. For example, by including only `cn;lang-fr` and `description;lang-fr` in the list of attributes to be returned, a client could search the French common name attribute with the filter (`cn;lang-fr=Jules`) and specify that the French common name and `description` attributes be returned.

Note that there is no way to retrieve all dialects of a particular language code. For example, the attribute type `cn;lang=en` is not the same as the attribute type `cn;lang=en-US`. Each dialect must be specifically requested. In general, avoid the use of dialects unless necessary. However, attributes with language codes are treated as subtypes of attributes without language codes. So, for example, the attribute `cn;lang=en` is a subtype of the attribute `cn`. Requesting the `cn` attribute retrieves all language code variations of the `cn` attribute.

Language codes are a relatively new development, and not all servers support them at this time. Check with your software vendor to see whether language codes are supported.

LDAP Overview Checklist

The term *LDAP* has come to mean the following things:

- The LDAP protocol itself, a standard, extensible directory access protocol used to access directory services.
- A set of models that guide you in your use of the directory: an information model that describes what you can put in the directory, a naming model that describes how you arrange and refer to directory data, a functional model that describes what you can do with directory data, and a security model that describes how directory data can be protected from unauthorized access.
- LDIF, a standard text format for exchanging directory data.
- LDAP server software, including commercial and open-source implementations.
- A set of command-line utilities commonly bundled with LDAP servers and LDAP-based applications.
- The LDAP APIs, used to develop LDAP client applications.

Further Reading

Active Directory Services Interface (ADSI). Available on Microsoft's SDK World Wide Web site at <http://msdn.microsoft.com>.

Authentication Methods for LDAP (RFC 2829). M. Wahl, H. Alvestrand, J. Hodges, and R. Morgan, 2000. Available on the World Wide Web at <http://www.ietf.org/rfc/rfc2829.txt>.

The C LDAP Application Program Interface (Internet Draft). M. Smith, T. Howes, A. Herron, C. Weider, M. Wahl, and A. Anantha, 1998. Available on the World Wide Web at <http://www.ietf.org>.

Code for the Representation of Names of Languages (ISO Standard 639). International Organization for Standardization, 1st edition, 1988.

Codes for the Representation of Names of Countries (ISO Standard 3166). International Organization for Standardization, 3rd edition, 1988.

The Java LDAP Application Program Interface (Internet Draft). R. Weltman, T. Howes, and M. Smith, 1998. Available on the World Wide Web at <http://www.ietf.org>.

Java Naming and Directory Interface (JNDI). Available on JavaSoft's JNDI Web site at <http://java.sun.com/products/jndi>.

The LDAP Data Interchange Format (LDIF): Technical Specification (RFC 2849). G. Good, 2000. Available on the World Wide Web at <http://www.ietf.org/rfc/rfc2849.txt>.

LDAP Programming with Java. R. Weltman and T. Dahbura, Addison-Wesley, 2000.

The LDAP URL Format (RFC 2255). T. Howes and M. Smith, 1997. Available on the World Wide Web at <http://www.ietf.org/rfc/rfc2255.txt>.

Lightweight Directory Access Protocol (v3) (RFC 2251). M. Wahl, T. Howes, and S. Kille, 1997. Available on the World Wide Web at <http://www.ietf.org/rfc/rfc2251.txt>.

Lightweight Directory Access Protocol (v3): Attribute Syntax Definitions (RFC 2252). M. Wahl, A. Coulbeck, T. Howes, and S. Kille, 1997. Available on the World Wide Web at <http://www.ietf.org/rfc/rfc2252.txt>.

Lightweight Directory Access Protocol (v3): Extension for Transport Layer Security (RFC 2830). M. Wahl, J. Hodges, and R. Morgan, 2000. Available on the World Wide Web at <http://www.ietf.org/rfc/rfc2830.txt>.

Lightweight Directory Access Protocol (v3): UTF-8 String Representation of Distinguished Names (RFC 2253). M. Wahl, S. Kille, and T. Howes, 1997. Available on the World Wide Web at <http://www.ietf.org/rfc/rfc2253.txt>.

The OpenLDAP Project Web site, <http://www.openldap.org>.

PerLDAP: An Object-Oriented LDAP Perl Module for Perl5. Available on Netscape's Directory Developer Central Web site at <http://developer.netscape.com/tech/directory>.

perl-ldap: A Client Interface to LDAP Servers. G. Barr. Available from the Comprehensive Perl Archive Network at <http://www.cpan.org>.

Programming Directory-Enabled Applications with Lightweight Directory Access Protocol. T. Howes and M. Smith, Macmillan Technical Publishing, 1997.

The String Representation of LDAP Search Filters (RFC 2254). M. Wahl, T. Howes, and S. Kille, 1997. Available on the World Wide Web at <http://www.ietf.org/rfc/rfc2254.txt>.

A Summary of the X.500(96) User Schema for Use with LDAPv3 (RFC 2256). M. Wahl, 1997. Available on the World Wide Web at <http://www.ietf.org/rfc/rfc2256.txt>.

The TLS Protocol Version 1.0 (RFC 2246). T. Dierks and C. Allen, 1999. Available on the World Wide Web at <http://www.ietf.org/rfc/rfc2246.txt>.

Understanding X.500: The Directory. D. Chadwick, International Thomson Computer Press, 1996. Now out of print; selected portions available on the World Wide Web at <http://www.salford.ac.uk/its024/X500.htm>.

The Unicode Standard, Version 2.0. Unicode Consortium, Addison-Wesley, 1996.

Use of Language Codes in LDAP (RFC 2596). M. Wahl and T. Howes, 1999. Available on the World Wide Web at <http://www.ietf.org/rfc/rfc2596.txt>.

Using Digest Authentication as a SASL Mechanism (RFC 2831). P. Leach and C. Newman, 2000. Available on the World Wide Web at <http://www.ietf.org/rfc/rfc2831.txt>.

Looking Ahead

[Chapter 3](#), LDAPv3 Extensions, will introduce methods for extending the LDAPv3 protocol and discuss some of the extensions currently available.

Chapter 3. LDAPv3 Extensions

- How LDAPv3 Is Extended
- The Root DSE and Extension Discovery
- Selected LDAPv3 Extensions
- Future Directions: Where Is LDAP Headed Next?
- LDAP Extensions and Future Directions Checklists
- Further Reading
- Looking Ahead

Lightweight Directory Access Protocol (LDAP) has three well-defined mechanisms for extending the core protocol: controls, extended operations, and Simple Authentication and Security Layer (SASL) mechanisms. In this chapter we describe each of these extension methods, as well as some of the extensions supported by Netscape Directory Server 6, and we tell you how you can use these extensions via command-line utilities and in your applications. We also describe where LDAP standards are headed in the future.

How LDAPv3 Is Extended

When the Internet Engineering Task Force (IETF) LDAP Working Group began work on LDAPv3, LDAPv2 work had recently been completed. LDAPv2 had a few shortcomings and missing features, some of which were incorporated into the LDAPv3 work. However, it was obvious that more feature enhancements would be required over time. Given that moving a protocol through the IETF standards process is a lengthy undertaking, an easier way of adding features to LDAP was needed. Instead of creating LDAPv4 and then LDAPv5, it would be better to develop a systematic method for extending LDAPv3 and add capabilities as requirements became clear.

The LDAP designers decided to provide three means for extending the protocol: controls, extended operations, and SASL mechanisms.

LDAP Controls

LDAP *controls* are extra pieces of information that accompany an existing LDAP operation and modify that operation in a useful way. For example, the Server-Side Sorting control modifies the behavior of the LDAP search operation. When the Server-Side Sorting control accompanies a search operation, the server sorts the search results before sending them to the client.

Each control contains an object identifier (OID) that uniquely identifies the type of control, a control-specific payload, and a flag that indicates the control's criticality. If a client sets the `isCritical` flag when sending the control to the server, the server must either perform the operation with the control taken into account, or return an `unavailableCriticalExtension` error and not process the operation at all. If the `isCritical` flag is not set by the client, the server is free to ignore the control if it cannot process it.

Multiple controls may be attached to a single operation. For example, the Virtual List View control is always used in conjunction with the Server-Side Sorting Request control.

LDAP Extended Operations

Sometimes new functionality needs to be added and no existing LDAP operation is similar to the required functionality. In such cases an LDAP extended operation may be defined to carry a completely new operation. An *extended operation* can carry any data. For example, Netscape has defined two LDAPv3 extended operations that mark the beginning and end of a bulk update of a directory server database. These extended operations allow a client to rapidly replace the contents of a portion of the directory server data over the network.

Each LDAP extended operation contains an OID that uniquely identifies the operation and an operation payload. The content of the payload is specific to each extended operation.

It is perfectly acceptable to attach controls to extended operations. Whether the control makes sense for the extended operation depends on the extended operation and the control.

SASL Authentication Mechanisms

One shortcoming of LDAPv2 is that it does not support a strong authentication protocol other than Kerberos v4, which was not widely deployed. The only widely available method for authenticating to an LDAPv2 server is to send a bind distinguished name (bind DN) and a

password to the server, unencrypted. Clearly an authentication method that is not susceptible to eavesdropping was required. Rather than mandate a single strong authentication method for LDAPv3, the standards developers chose to incorporate an authentication framework called *Simple Authentication and Security Layer (SASL)* that allows multiple authentication protocols to exist. SASL is a method for adding authentication and, optionally, security to connection-oriented protocols such as LDAP. LDAP currently uses SASL to provide extensibility in authentication methods. The SASL framework is specified in RFC 2222.

A *SASL mechanism* describes the flow of information that needs to occur to support a particular authentication method. SASL mechanisms are defined for several authentication protocols, such as the Kerberos v5 authentication protocol and the DIGEST-MD5 authentication protocol. For LDAP to use a particular SASL mechanism, a standards document called a SASL profile was included in RFC 2829 that describes how SASL is used within LDAPv3.

When a client initiates a SASL bind operation, it indicates the particular type of SASL mechanism to use. Depending on the type of mechanism, multiple messages may need to flow back and forth between the client and server. For example, the DIGEST-MD5 SASL mechanism defined in RFC 2831 allows the server to verify that the client knows a password without requiring that the client send the password. Instead, the server constructs a challenge, and the client responds to the challenge in such a way that knowledge of the password is proven. This exchange requires four steps: the initial bind request, the server's challenge, the client's response to the challenge, and a response code from the server. SASL allows an arbitrary number of exchanges.

RFC 2829 describes in detail how two SASL mechanisms—DIGEST-MD5 and EXTERNAL—are used within LDAP. DIGEST-MD5 is an authentication method that does not transmit passwords in the clear over the network and is therefore less susceptible to eavesdropping attacks. The EXTERNAL SASL mechanism allows LDAP servers to reuse authentication credentials established by a lower-layer protocol such as Secure Sockets Layer (SSL) or Transport Layer Security (TLS). To ensure that LDAP authentication is secure from eavesdroppers, all LDAPv3-compliant servers that support password-based authentication are required to implement the DIGEST-MD5 SASL mechanism.

The Root DSE and Extension Discovery

Suppose that an LDAP client wants to use an LDAPv3 control, extended operation, or SASL mechanism. How does the client know whether the server supports the required extension? All LDAPv3 extensions supported by a server are advertised in a special LDAP entry called the *root DSE*. The root DSE is a directory entry that contains operational information about the server. Among other things, the root DSE contains a list of every LDAP control, extended operation, and SASL mechanism that the server supports.

Clients that plan to use an LDAP extension mechanism can read the root DSE to learn whether the server they've connected to supports the required extension. If so, the operation(s) can proceed. However, if the server doesn't support the required extension, the client program has a decision to make. If the application cannot function without the LDAP extension, the client has no choice but to abort the operation and inform the user. In other cases, the client may be able to use an alternative approach.

For example, if a client wants the results of a search operation to be sorted, it can check whether the server supports the Server-Side Sorting control and use the control if it is available. If not, it can fall back to an alternative approach of retrieving the unsorted list of entries from the server and sorting the list itself. Graceful degradation of service is possible if clients read the root DSE to determine the available extensions supported by a server. Controls and extended operations are advertised in the root DSE by their unique OID. SASL mechanisms are identified by a unique identifying string that must be registered with the Internet Assigned Numbers Authority (IANA), a standards body that registers names and ensures that every name assigned for a particular purpose is unique.

Let's look at a typical root DSE. This example is from Netscape Directory Server 6. To retrieve the root DSE of a directory server, you must perform a base-level search with a search base of "" (the empty string) and a filter of "(objectclass=*)", as shown in [Listing 3.1](#).

Listing 3.1 Discovering Extensions, Controls, and SASL Mechanisms Supported by a Server

```
ldapsearch -h ldap.example.com -s base -b "" "(objectclass=*)"  
supportedExtension
```

➥ supportedControl supportedSASLMechanisms

dn:

supportedExtension: 2.16.840.1.113730.3.5.7

supportedExtension: 2.16.840.1.113730.3.5.8

supportedExtension: 2.16.840.1.113730.3.5.3

supportedExtension: 2.16.840.1.113730.3.5.5

supportedExtension: 2.16.840.1.113730.3.5.6

supportedExtension: 2.16.840.1.113730.3.5.4

supportedControl: 2.16.840.1.113730.3.4.2

```
supportedControl: 2.16.840.1.113730.3.4.3
supportedControl: 2.16.840.1.113730.3.4.4
supportedControl: 2.16.840.1.113730.3.4.5
supportedControl: 1.2.840.113556.1.4.473
supportedControl: 2.16.840.1.113730.3.4.9
supportedControl: 2.16.840.1.113730.3.4.16
supportedControl: 2.16.840.1.113730.3.4.15
supportedControl: 2.16.840.1.113730.3.4.17
supportedControl: 2.16.840.1.113730.3.4.14
supportedControl: 1.3.6.1.4.1.1466.29539.12
supportedControl: 2.16.840.1.113730.3.4.13
supportedControl: 2.16.840.1.113730.3.4.12
supportedControl: 2.16.840.1.113730.3.4.18
supportedSASLMechanisms: EXTERNAL
supportedSASLMechanisms: DIGEST-MD5
```

The `supportedExtension`, `supportedControl`, and `supportedSASLMechanisms` attribute values indicate that the server supports the given extended operation, control, or SASL mechanism. For example, the root DSE shown previously, from Netscape Directory Server 6, contains the attribute `supportedControl: 2.16.840.1.113730.3.4.2`, which indicates that the server supports the `ManageDSAIT` control. Clients that read the root DSE can test for the presence of the attribute values corresponding to the extensions they want.

Although reading the root DSE is the recommended approach, another method that clients can use to discover whether a server supports a given LDAPv3 extension is simply to try to use a critical control, extended operation, or SASL mechanism and check whether the server returns a result code indicating that the extension is not supported. For example, if a control is not supported, or if it can't be used with the requested operation, the server will return the result code `unavailableCriticalExtension`.

Selected LDAPv3 Extensions

Numerous LDAPv3 extensions are in common use today. In this section we describe some of the controls, extended operations, and SASL mechanisms supported by Netscape Directory Server 6. We also tell how you can take advantage of these features using command-line utilities and C and Java SDKs. The command-line utilities are included with Netscape Directory Server, and the SDKs are available for download from the <http://enterprise.netscape.com> Web site. Both SDKs are also available as source code from <http://www.mozilla.org>.

Note

You may encounter other versions of the `ldapsearch` and `ldapmodify` command-line utilities. For example, the Solaris 8 operating system ships with versions of these utilities that are slightly different from the Netscape versions. If you find that the examples in this chapter don't behave as expected, make sure that you are using the Netscape version.

In the following descriptions, any Internet Drafts or RFC documents may be obtained from the IETF's Web site at <http://www.ietf.org>.

The ManageDSAIT Control

A client uses the `ManageDSAIT` control when it wants to directly manipulate referral information in the directory. Normally, if an entry contains a referral (in a `ref` attribute), the server returns to the client referral instead of the entry containing the referral. However, if the client attaches the `ManageDSAIT` control to the operation, the server knows that the client wants to retrieve or manipulate any `ref` attributes directly. This control is useful for updating or retrieving referral information.

The `ManageDSAIT` control is identified by the OID `2.16.840.1.113730.3.4.2` and is defined in the Internet Draft *Named Subordinate References in Lightweight Directory Access Protocol (LDAP) Directories* (<http://www.ietf.org/internet-drafts/draft-zeilenga-ldap-namedref-05.txt>). Servers that support this control will advertise this capability with the value `2.16.840.1.113730.3.4.2` in the `supportedControl` attribute of the root DSE.

To use the `ManageDSAIT` control with the `ldapsearch` and `ldapmodify` command-line utilities, add the `-M` (uppercase) flag to the other command-line options.

To use the `ManageDSAIT` control with the C SDK, create an `LDAPControl` structure and set its `ldctl_oid` field to the OID for the control: `2.16.840.1.113730.3.4.2`. In the Java SDK, create an `LDAPControl` object, passing the OID to the object constructor function.

The Persistent Search Request and Entry Change Notification Response Controls

The Persistent Search Request control allows a client to receive notification when entries in the directory are changed. Normally a server responds to a search request by sending all the matching entries and a result message. When the Persistent Search Request control is included, the server does not send the result message. Instead, whenever the directory is updated, the server sends the updated entry to the client if the entry matches the search criteria. The persistent search normally does not terminate until the client abandons the operation, although the server is free to terminate the operation if it is short on resources.

For example, a client might be interested in knowing when the e-mail address of any person in the directory changes. To obtain this information, the client can issue a persistent search for all entries that contain an e-mail address. As new entries are added to the directory, or as entries have their e-mail addresses updated, the server sends these updated entries to the client, assuming that the client has access to the entries.

When constructing a Persistent Search Request control, the client can include several options to fine-tune the persistent search behavior:

- The client may indicate that it is interested in entries only if they are the result of a particular LDAP update type or types. For example, a client can request that it be notified only when entries are added to or removed from the directory. The server will not notify the client of changes that arise from LDAP modify or modify DN operations in this example. By default, the server will notify the client about all changes to the directory.
- The client may request that the server not return the initial set of matching entries. The client may want to suppress the initial set of entries if it is interested in learning only about changes to the directory and is uninterested in the current set of matching entries. By default, the server returns all entries that match the search request at the time it is received.

When the server returns entries that result from a persistent search to a client, it includes an Entry Change Notification Response control. This control contains additional information about the operation that caused the entry to be returned to the client:

- The type of change made to the entry (add, delete, modify, or modify DN).
- The previous DN of the entry (included only and useful only if the entry was renamed).
- The change number, a sequential number assigned to each change. The server may not send this information if it is not configured to support an LDAP *changelog*, a special database that tracks changes made to the directory.

These additional pieces of information can be useful to certain types of clients—in particular, clients that maintain copies of all or part of a directory.

The Persistent Search Request control, which is sent from the client to the server, is identified by the OID [2.16.840.1.113730.3.4.3](#). The Entry Change Notification Response control, which is sent from the server to the client, is identified by the OID [2.16.840.1.113730.3.4.7](#). Servers that support these controls will advertise this capability with the values [2.16.840.1.113730.3.4.3](#) and [2.16.840.1.113730.3.4.7](#) in the `supportedControl` attribute of the root DSE.

To use the Persistent Search Request control with the `ldapsearch` utility provided with Netscape Directory Server 6 and later versions (earlier versions of the `ldapsearch` utility do not support this option), use the `-C` option. The syntax is

```
-C ps:changetype[:changesonly][:entrynotificationcontrols]
```

where

- `changetype` is one of `add`, `delete`, `modify`, `rename`, or `any` and indicates the types of changes that should be reported.
- `changesonly` is `true` or `false`. `true` indicates that the client does not want to receive the initial set of matching entries; `false` (the default) indicates that the client wants to receive the initial set of matching entries.
- `entrynotificationcontrols` is `true` or `false`. `true` indicates that the client wants to receive entry notification controls. `false` indicates that the client is not interested in receiving entry notification controls.

For example, the following `ldapsearch` command:

```
ldapsearch -h ldap.example.com -b "dc=example, dc=com" -s sub -C ps:modify:true "(  
mail=*)"
```

performs a persistent search that returns modified entries that also contain an e-mail address (`mail` attribute value). Entry Change Notification Response controls are included.

The C SDK has a convenience function, `ldap_create_persistentsearch_control()`, for creating a Persistent Search Request control; and another function, `ldap_parse_entrychange_control()`, for parsing an Entry Change Notification Response control returned from the server.

The Java SDK has two classes—`LDAPPersistSearchControl` and `LDAPEntryChangeControl`—used for creating persistent search requests and parsing entry notification responses.

The Server-Side Sorting Request and Response Controls

The Server-Side Sorting Request control is used to request that the server sort the search results before sending them to the client. If the server has indexes that can satisfy the sort order requested by the client, it can sort the results much faster than the client can.

The Server-Side Sorting Request control carries one or more sets of an attribute type and a sort order (ascending or descending) to use when sorting on that attribute. This control allows you, for example, to sort by surname first and then by given name, to produce a correctly sorted list of people from a white pages directory.

Sent from the server to the client after a Server-Side Sorting Request control is processed, the Server-Side Sorting Response control carries status and error codes from the sort operation.

The Server-Side Sorting Request control, which is sent from the client to the server, is identified by the OID `1.2.840.113556.1.4.473`. The Server-Side Sorting Response control, which is sent from the server to the client, is identified by the OID `1.2.840.113556.1.4.474`. Both controls are defined in RFC 2891, *LDAP Control Extension for Server Side Sorting of Search Results* (<http://www.ietf.org/rfc/rfc2891.txt>). This document is currently a proposed standard. Servers that support these controls will advertise this capability with the values `1.2.840.113556.1.4.473` and `1.2.840.113556.1.4.474` in the `supportedControl` attribute of the root DSE.

For server side sorting with the `ldapsearch` command-line utility, the `-S` (uppercase) and `-x` options must be used together. The `-S` option is used to specify the sort attribute(s) and sort order, and the `-x` flag is used to indicate that the server should perform the sorting. If you use the `-S` option without the `-x` option, the command-line utility will perform the sorting.

The argument to the `-S` option is a single attribute type, optionally prefixed with a dash to indicate reverse sort order. You may use multiple `-S` options if you want to sort on multiple attributes. The `-S` options are processed in the order they appear on the command line.

For example, to search the directory for all people, sorting the results by surname first and then given name, use this syntax:

```
ldapsearch -h host -s sub -b "dc=example,dc=com" -S sn -S givenname -x "(objectclass=person)"
```

To search the directory for all people, sorting in reverse order by the `uid` (user ID) attribute, use this syntax:

```
ldapsearch -h localhost -s sub -b "dc=example,dc=com" -S -uid -x "(objectclass=person)"
```

The C SDK contains the convenience function `ldap_create_sort_keylist()`, which creates the sort ordering, and the `ldap_create_sort_control()` function, which creates the Server-Side Sorting Request control. Because the server returns a control to the client with additional information about the search request, the C SDK contains the `ldap_parse_sort_control()` function, which interprets the Server-Side Sorting Response control.

The Java SDK features several classes used to support Server-Side Sorting Request and Response controls. The `LDAPSortKey` class is used to specify the sort order, and the `LDAPSortControl` class is attached to the search operation to request server side sorting. The Java SDK makes the Server-Side Sorting Response control data available as an object of type `LDAPSortControl`.

The Virtual List View Request and Response Controls

The Virtual List View (VLV) Request control allows an LDAP client to request that the server send search results in small, manageable chunks. It also allows a client to move forward and backward through the results of a search operation. A GUI client that displays results in a window can use the VLV Request control to request that only enough entries to fill the window be sent, and it can page forward and backward through the result set. The Netscape Communicator version 4 address book fully supports VLV.

The VLV Request control, which is sent from the client to the server, is identified by the OID `2.16.840.1.113730.3.4.9`. The VLV Response control, which is sent from the server to the client, is identified by the OID `2.16.840.1.113730.3.4.10`. Both controls are defined in the Internet Draft *LDAP Extensions for Scrolling View Browsing of Search Results* (<http://www.ietf.org/internet-drafts/draft-ietf-ldapext-ldapv3-vlv-09.txt>). Servers that support these controls will advertise this capability with the values `2.16.840.1.113730.3.4.9` and `2.16.840.1.113730.3.4.10` in the `supportedControl` attribute of the root DSE.

To use VLV correctly with the command-line utilities, you must combine server side sorting (discussed in the previous section) with a VLV specification. The sorting information tells the server how to sort the result set, and the VLV specification tells the server which part of the result set to send.

There are two types of VLV specifications. The first allows you to specify an absolute position in the result set. For example, if the result set has 150 entries, you can specify that you want the server to send you entries 10 through 20. This feature is used most frequently by a GUI client that presents an interface that allows a user to page through a complete list of all entries. Imagine a directory with 10,000 user entries. Without VLV, the client would need to retrieve all 10,000 entries, sort them, and cache the sorted list. With VLV, the client can retrieve only as many entries as will fill its window, and it can retrieve the next set of entries as the user navigates forward and backward through the list.

To use this type of VLV specification with the Netscape command-line utility, you must specify a sort order and a VLV specification. For information about sort ordering, review the previous section, The Server-Side Sorting Request and Response Controls. The VLV specification takes the following form:

`before:after:index:content_count`

where

- `before` is the number of entries before the target to include.
- `after` is the number of entries after the target to include.
- `index` is the offset of the target entry within the result set. An `index` of 1 always means the first entry. If `index` and `content_count` are equal, the last entry in the result set is selected.
- `content_count` is the expected size of the result set. If 0 (zero) is provided for `content_count`, the actual size of the result set, as computed by the server, is used. A client might use 0 for `content_count` if it does not know the size of the result set. If the client specifies a nonzero `content_count`, the server uses this information, along with the other parameters, to return entries at a relative offset within the result set. The reason that `content_count` exists is so that client applications can implement interfaces that allow users to jump around a long list using a scroll bar. With an index of 33 and a count of 100, the application can jump 33 percent of the way

into the list. The scroll bar can be mapped to the coordinate space of the result set, or vice versa.

Suppose that you want to display in a window the first five directory entries, sorted by surname. The `ldapsearch` command shown in [Listing 3.2](#) illustrates how to do this.

Listing 3.2 Using VLV to Retrieve the First Five Entries, Sorted by Surname

```
ldapsearch -D "uid=kvaughan,ou=people,dc=example,dc=com" -w bribery -s sub -b "ou=People,  
➥ dc=example,dc=com" -S sn -x -G 0:4:1:0 "(objectclass=*)" sn  
version: 1  
  
dn: uid=dakers, ou=People, dc=example,dc=com  
sn: Akers  
  
  
dn: uid=falbers, ou=People, dc=example,dc=com  
sn: Albers  
  
  
dn: uid=calexand, ou=People, dc=example,dc=com  
sn: Alexander  
  
  
dn: uid=ealexand, ou=People, dc=example,dc=com  
sn: Alexander  
  
  
dn: uid=rbannist, ou=People, dc=example,dc=com  
sn: Bannister  
index 1 content count 151
```

The sort specification (`-S sn -x`) tells the server that you want to sort on the surname attribute and that the server should perform the sorting. The VLV specification (`-G 0:4:1:0`) tells the server that you want to show 0 entries before and 4 entries after the target entry at index 1. Because the client does not know the size of the result set, `content_count` is 0.

Now suppose that the user presses the **Page Down** key and the client wants to scroll down through the list, positioning at the top the entry that was previously at the bottom of the list. By changing the VLV specification to `-G 0:4:5:0`, the client can request entries 5, 6, 7, 8, and 9, as shown in [Listing 3.3](#).

Listing 3.3 Using VLV to Retrieve the Next Five Entries, with Overlap

```
ldapsearch -D "uid=kvaughan,ou=people,dc=example,dc=com" -w bribery -s sub -b "ou=People,  
➥ dc=example,dc=com" -S sn -x -G 0:4:5:0 "(objectclass=*)" sn  
version: 1
```

```
dn: uid=rbannist, ou=People, dc=example,dc=com  
sn: Bannister
```

```
dn: uid=abarnes, ou=People, dc=example,dc=com  
sn: Barnes
```

```
dn: uid=abergin, ou=People, dc=example,dc=com  
sn: Bergin
```

```
dn: uid=jbourke, ou=People, dc=example,dc=com  
sn: Bourke
```

```
dn: uid=jbrown, ou=People, dc=example,dc=com  
sn: Brown  
index 5 content count 151
```

Notice that the entry `uid=rbannist,ou=People,dc=example,dc=com` becomes the top entry in the result set. Also note that each VLV search set also includes a final line of output: `index x content count y`. When the server processes a VLV Request control, it returns to the client, in another control, the index of the target entry and an indication of the total number of entries in the result set. The index is useful in an application that implements type-down addressing, in which the press of a key takes the user to the first entry in a list matching that key; for example, an address book application could jump to the first person whose surname starts with "s" when the user presses the S key. In this case, the client would issue a VLV specification of the second form (`before:after:search_value`), take note of the index reported by the server, and then issue a VLV specification of the first form (`before:after:index:content_count`).

The second type of VLV specification allows a client to jump to a particular matching entry and is most commonly used to implement type-down addressing. Specifically, it allows the client to jump to the first entry in the result set where the prefix of the sort value is equal to or greater than the value provided by the user. For example, if a GUI client presents a list of directory entries to a user, the user can type the letter "j" to jump to the first name that begins with the letter "j." The `ldapsearch` command shown in [Listing 3.4](#) performs this VLV search. This VLV specification instructs the server to locate the first entry with a surname beginning with "j," returning 0 entries before and 4 entries after the target entry.

Listing 3.4 Using VLV to Retrieve the First Five Entries Where the Surname Begins with "j"

```
ldapsearch -D "uid=kvaughan,ou=people,dc=example,dc=com" -w bribery -s sub -b "ou=People,  
➥ dc=example,dc=com" -S sn -x -G 0:4:j "(objectclass=*)" sn
```

```
version: 1  
  
dn: uid=mjablons, ou=People, dc=example,dc=com  
sn: Jablonski
```

```
dn: uid=bjablons, ou=People, dc=example,dc=com  
sn: Jablonski
```

```
dn: uid=tjames, ou=People, dc=example,dc=com  
sn: James
```

```
dn: uid=kjensen, ou=People, dc=example,dc=com  
sn: Jensen
```

```
dn: uid=bjensen, ou=People, dc=example,dc=com  
sn: Jensen  
index 48 content count 151
```

If reverse sorting is used, the server makes the target entry the first entry in the result set where the prefix of the sort value is equal to or less than the value provided by the user, as you would expect.

The content count returned by the server can be valuable to GUI clients that display a scroll bar. The content count can be used to appropriately position the scroll bar and size the scroll bar "thumb." The content count is not guaranteed to be accurate, however, because the server may not have sufficient resources to calculate the result set size. The server can return a content count of zero in this case, and clients should be prepared to deal with this.

Both the C and Java SDKs provide convenience functions that help you create VLV Request controls to send to the server and convenience functions that help you parse VLV Response controls returned to the client. The C SDK contains the `ldap_create_virtuallist_control()` function, which helps you create the VLV Request control, and the `ldap_parse_virtuallist_control()` function to parse the VLV Response control sent back to the client. The Java SDK contains the `LDAPVirtualListControl` class, which can be used to create a VLV Request control, and the `LDAPVirtualListResponse` class, which can be used to parse the VLV Response control sent back to the client.

The Proxied Authorization Control

The Proxied Authorization control allows an LDAP client to impersonate another entry for a specific operation. For example, if a client has authenticated to a directory server as a particular user, it can use the Proxied Authorization control to perform an operation as if it had authenticated as a different user. To take advantage of this option, the client must initially authenticate as a user who has permission to use the Proxied Authorization control.

The Proxied Authorization control is most useful in trusted applications that need to perform operations on behalf of many different users. By using the Proxied Authorization control, an application can avoid having to reauthenticate for each operation. As a result, the application's performance improves because now it is unnecessary to send a bind operation each time a new authorization identity is required.

The Proxied Authorization control is identified by the OID `2.16.840.1.113730.3.4.18` and is defined in the Internet Draft *LDAP Proxied Authorization Control* (<http://www.ietf.org/internet-drafts/draft-weltman-ldapv3-proxy-11.txt>). Servers that support this control will advertise this capability with the value `2.16.840.1.113730.3.4.18` in the `supportedControl` attribute of the root DSE.

To use the Proxied Authorization control with the `ldapsearch` command-line utility, use the `-Y` option to specify the DN of the entry to be impersonated. For example, to bind to the directory as the user `jvedder` and perform a search operation as the user `bjablons`, use the following command. Note that the actual proxy DN must be preceded by the string "`dn:uid=bjablons, ou=people,dc=example,dc=com`":

```
ldapsearch -D "uid=jvedder,ou=people,dc=example,dc=com" -w befitting -Y "dn:uid=bjablons, ou=people,dc=example,dc=com" -h localhost -s sub -b "dc=example,dc=com" "(uid=jvedder)"  
→ userpassword
```

Note that the user `jvedder` must have appropriate permission within the subtree `dc=example,dc=com` to use the Proxied Authorization control. If this permission has not been granted, an LDAP error 50, "insufficient access rights," will be returned to the client. For Netscape Directory Server, this right is called the *proxy right*. For more information on granting proxy rights, see the *Netscape Directory Server Administrator's Guide*.

Both the C and Java SDKs provide convenience functions that help you construct Proxied Authorization controls. The C SDK contains a convenience function, `ldap_create_proxyauth_control()`, which assists you in setting up the Proxied Authorization control. The Java SDK includes the `LDAPProxiedAuthControl` class, which can be added to the search constraints of any LDAP operation.

Password Expiration Controls

Netscape Directory Server has a password policy feature that allows you to enforce restrictions on passwords, including maximum password age and password quality. Passwords can be set to expire at a particular point in time. After a password expires, the only LDAP operation permitted is to set a new password. The password policy feature uses LDAP controls to send information about impending password expiration and expired passwords to the client.

Unlike other controls, these LDAP controls are sent from the server to the client to inform it that a password is about to expire and how far in the future the expiration will occur, or that a password has expired. Your application should check for these controls and inform the user about password expiration.

The Password Expiring control is sent from the server to the client to indicate that a password is about to expire. It is identified by the OID `2.16.840.1.113730.3.4.5`. The Password Expired control is sent from the server to the client to indicate that a password has expired. It is identified by the OID `2.16.840.1.113730.3.4.4`. Both controls are defined in the Internet Draft *Password Policy for LDAP Directories* (<http://www.ietf.org/internet-drafts/draft-behera-ldap-password-policy-07.txt>). Servers that support these controls will advertise this capability with the values `2.16.840.1.113730.3.4.5` and `2.16.840.1.113730.3.4.4` in the `supportedControl` attribute of the root DSE.

If a Password Expiring control is received from the server, the Netscape LDAP command-line tools display a message like this:

```
Warning ! Your password will expire after 30 days.
```

You do not need to specify any command-line options to enable recognition of the Password Expiring and Password Expired controls.

The C and Java SDKs both allow you to recognize when Password Expiring or Password Expired controls are returned with a bind result. With the C SDK, applications can use the `ldap_parse_result()` call and check for the presence of either of these two controls. If the Password Expiring control is present, applications can examine the `ldctl_value` field, which contains the number of seconds until the

password expires.

In the Java SDK, applications should use the `getResponseControls()` method to get all the controls returned from the server and check for the presence of either the Password Expiring or the Password Expired control. If an `LDAPPasswordExpiringControl` object is present, the `getSecondsToExpiration()` method can be called to determine the number of seconds until the password expires.

Bulk Import Extended Operations

The bulk import extended operations allow a client to rapidly import a series of entries into the directory, using the LDAP protocol. With previous versions of Netscape Directory Server, if you needed to replace the contents of the directory, you had to log in to the server machine and use the `ldif2db` utility. Now, with the bulk import extended operations, you can use an LDAP client to replace the contents of the directory while taking advantage of the high-speed import capabilities of the server.

There are two extended operations: Bulk Import Start and Bulk Import Finished. The Bulk Import Start extended operation carries a DN that denotes the top of the subtree that the client wants to replace. If the server does not hold the given subtree, it returns an error to the client. Otherwise, the server prepares to receive a bulk update. The client then sends a series of LDAP add operations and finishes the bulk update with the Bulk Import Finished extended operation.

The bulk import capabilities are useful in an environment where the directory contents are replaced periodically—for example, when directory data is periodically regenerated from an external data source like a relational database. A bulk update allows rapid replacement of the server contents, through use of a remote LDAP client.

Note

When Netscape Directory Server is servicing a bulk update operation, the database's contents are taken offline and are not available for searching. Clients that attempt to search the directory are returned an error.

The Bulk Import Start extended operation is identified by the OID `2.16.840.1.113730.3.5.7`. The Bulk Import Finished extended operation is identified by the OID `2.16.840.1.113730.3.5.8`. Neither of these extended operations is currently defined in a standards document. Servers that support these operations will advertise this capability with the values `2.16.840.1.113730.3.5.7` and `2.16.840.1.113730.3.5.8` in the `supportedExtension` attribute of the root DSE.

To perform a bulk import using the `ldapmodify` command-line utility, use the `-B` option to specify the subtree that will be bulk-loaded. Otherwise the command line is the same as if you were adding the entries to an empty directory. For example, if you have an LDAP Data Interchange Format (LDIF) file containing data for the `dc=example,dc=com` subtree in the file `example.ldif`, you can perform a bulk import with the following command:

```
ldapmodify -D "cn=Directory Manager" -w password -a -B "dc=example, dc=com" < example.ldif
```

The `-B` option causes the `ldapmodify` command to bracket the updates with Bulk Update Start and Bulk Update Finished extended operations, specifying that the subtree `dc=example,dc=com` is to be replaced. The `-a` option is required if your LDIF file consists of LDAP entry data instead of LDIF update statements (update statements contain `changetype: add` immediately after the entry DN). Netscape Directory Server supports only bulk add operations. It does not support bulk application of modify, delete, or rename operations.

Note

Netscape Directory Server processes a bulk update by replacing the entire content of the directory database containing the entry named by the `-B` option. It is not possible to replace only a portion of a directory database. For example, if the entire `dc=example,dc=com` subtree is held in a single database, it's not possible to bulk-load only a subtree like `ou=people,dc=example,dc=com`. Even if you supply `ou=people,dc=example,dc=com` for the `-B` option, the entire content of the database will be replaced with the data you provide.

Neither the C nor the Java SDK provides any specific support for the bulk import extended operations at this time, although the basic, low-level extended operation facilities included in the SDKs can be used to create the needed extended operations.

The EXTERNAL SASL Mechanism

The EXTERNAL SASL mechanism allows a client to reuse authentication credentials already established by another protocol. For example, if the LDAP connection has been established by LDAP-over-SSL or LDAP-over-TLS, the server may have already established the client's identity using certificate-based client authentication. The SASL EXTERNAL mechanism allows a client to request that those credentials be used.

The EXTERNAL SASL mechanism is identified by the string "EXTERNAL" and is defined in RFC 2222, *Simple Authentication and Security Layer (SASL)* (<http://www.ietf.org/rfc/rfc2222.txt>). Servers that support this SASL mechanism will advertise this capability with the value `EXTERNAL` in the `supportedSASLMechanisms` attribute of the root DSE.

The command-line utilities fully support SSL client authentication, which uses the EXTERNAL SASL mechanism. SSL client authentication requires use of the following command-line options:

- `-Z`. The instruction telling the utility to use SSL.
- `-P <pathname>`. The path to the SSL certificate database.
- `-N <certname>`. The name of the certificate to use for client authentication.
- `-W`. The SSL key password.

For example, if the SSL certificate database is in the file `/home/bjensen/.netscape/cert7.db`, the certificate name is `bjensencertname`, and the password protecting the key database is "secret," an `ldapsearch` command using SSL client authentication will look like this:

```
ldapsearch -h hostname -p 636 -b "dc=example,dc=com" -N "bjensencertname" -Z -P /home/
➥ bjensen/.netscape/cert7.db -W secret "(uid=gjensen)"
```

SDK Support

To use SSL client authentication and the SASL EXTERNAL bind method with the C SDK, use the `ldapssl_clientauth_init()` and `ldapssl_init()` functions to initialize the SSL library and an LDAP session. Then use the `ldap_sasl_bind()` or `ldap_sasl_bind_s()` functions to authenticate to the directory.

With the Java SDK, you use the `LdapSSLSocketFactory` class, along with a class that you provide that supports SSL. You then use one of the `LDAPConnection.authenticate()` methods that support SASL authentication, supplying EXTERNAL as the mechanism name. For more information, consult the Java SDK documentation.

The DIGEST-MD5 SASL Mechanism

The DIGEST-MD5 SASL mechanism authenticates a client to a directory server. This authentication method is secure from eavesdroppers because it does not send the bind password in the clear over the network. All LDAPv3-compliant servers must support this authentication method.

The DIGEST-MD5 SASL mechanism is identified by the string "DIGEST-MD5" and is defined in RFC 2831, *Using Digest Authentication as a SASL Mechanism* (<http://www.ietf.org/rfc/rfc2831.txt>). Servers that support this SASL mechanism will advertise this capability with the value **DIGEST-MD5** in the **supportedSASLMechanisms** attribute of the root DSE. At this time, support for the DIGEST-MD5 SASL mechanism is not available in either the command-line utilities or the LDAP SDKs.

Future Directions: Where Is LDAP Headed Next?

Over the past few years, LDAP has steadily gained mindshare and acceptance in the marketplace. In the future, we believe that LDAP will become ubiquitous and evolve as described in the discussion that follows.

Increased Integration into Operating Systems and Infrastructure Middleware

In the late 1990s, Novell and Microsoft embraced LDAP as a core technology for their operating systems. They were the first operating system (OS) vendors to integrate LDAP deeply into core OS offerings. Since then, other OS vendors, most notably Sun Microsystems, have integrated LDAP into their operating systems, and they use directories to distribute user, group, and other information. This trend will continue, with OS vendors replacing their legacy naming servers (such as NIS and NIS+) with LDAP.

Middleware vendors have also integrated LDAP into their offerings. Application servers and other middleware tools from companies including Netscape and Netegrity, as well as infrastructure management applications from companies such as Access360, all use LDAP as their store of information about users, groups, and access to protected resources.

Emerging Standards Work

At the time of this writing, three major LDAP-related work topics are being addressed in the IETF: an update to the core LDAPv3 protocol, establishment of a common access control model for LDAP, and development of a vendor-neutral client update protocol.

An Update to the LDAP Standard

The LDAPBIS Working Group has recently been formed to update the LDAPv3 standards and move them from IETF Proposed Standard status (their current status as of the time of this writing) to Draft Standard status, the next level in the IETF standards process. For more information, see the LDAPBIS Working Group page at <http://www.ietf.org/html.charters/ldapbis-charter.html>.

Access Control Model for LDAP

Work is under way to establish a vendor-neutral access control model for LDAP. Establishment of a common access control model will facilitate consistent secure access, replication, and management across heterogeneous LDAP implementations.

At this time, a requirements document, *Access Control Requirements for LDAP*, has been published as RFC 2820. This document outlines the requirements that subsequent access control models must meet. A proposed access control model that is intended to meet these requirements is being developed.

LDAP Client Update Protocol (LCUP)

The LDAP Client Update Protocol (LCUP) is a method for synchronizing cached copies of directory information, such as copies maintained by offline directory clients and high-performance middleware components and application servers. See *LDAP Client Update Protocol* (<http://www.ietf.org/internet-drafts/draft-ietf-ldup-lcup-03.txt>).

Other LDAP-Related Standards Work

Smaller projects are under way in the IETF to define new controls, extended operations, and SASL mechanisms that extend LDAPv3. Some current projects under way at the time of this writing include

- The StartTLS extended operation, which allows a client to begin using TLS on an already established LDAP connection. For more information, see RFC 2830, *Lightweight Directory Access Protocol (v3): Extension for Transport Layer Security* (<http://www.ietf.org/rfc/rfc2830.txt>).
- Development of a method for accessing the directory using UDP, a connectionless protocol. See *Lightweight Directory Access Protocol over UDP/IP* (<http://www.ietf.org/internet-drafts/draft-ietf-ldapext-ldapudp-00.txt>).
- Ways to use the Domain Name System (DNS) to locate LDAP servers. See *A Taxonomy of Methods for LDAP Clients Finding Servers* (<http://www.ietf.org/internet-drafts/draft-ietf-ldapext-ldap-taxonomy-05.txt>).
- Standardization of the C and Java LDAP APIs. See *The C LDAP Application Program Interface* (<http://mozilla.org/directory/ietf-docs/draft-ietf-ldapext-ldap-c-api-05.txt>) and *The Java LDAP Application Program Interface* (<http://www.ietf.org/internet-drafts/draft-ietf-ldapext-ldap-java-api-18.txt>).

LDAP and XML

eXtensible Markup Language (XML) is revolutionizing the way information is exchanged on the Internet. A variety of common tools and techniques have been developed to make it easier to create an XML-based application and to move XML data into and out of a huge variety of data sources, from relational database management systems (RDBMSs) to LDAP directories. XML is a metalanguage that allows document formats to be created that not only have structure but also convey semantic meaning about their contents.

LDAP directories are also all about structure and meaning: Directory entries usually describe real-world objects, their qualities (attributes), and their relationships to other objects in the directory. It's therefore not surprising that XML provides a convenient way to describe directory information. In fact, if XML had been available when the first LDAP implementations were being written, it would have been used as the data interchange format instead of LDIF. In the more distant future, an XML-based directory markup format will probably replace LDIF.

DSML

The current XML industry standard for how to describe directory data is DSML, or Directory Services Markup Language. It was developed by a consortium of companies including Bowstreet, IBM, Netscape, Microsoft, Novell, and Oracle. The specification is managed by the Organization for the Advancement of Structured Information Standards (OASIS).

The currently published version of the DSML specification is 2.0. An XML document conforming to DSML 2.0 contains directory entries and, optionally, schema information. Additionally, DSML 2.0 documents can describe directory queries as well as updates to be applied to a directory.

A directory entry in DSML format might look like [Listing 3.5](#). In LDIF format, the same entry would look like [Listing 3.6](#).

Listing 3.5 A Directory Entry Represented in DSML 2.0 Format

```
<dsml:entry dn="uid=bjensen,ou=people,dc=example,dc=com">

  <dsml:objectclass>

    <dsml:oc-value>top</dsml:oc-value>

    <dsml:oc-value>person</dsml:oc-value>

    <dsml:oc-value>organizationalPerson</dsml:oc-value>

    <dsml:oc-value>inetOrgPerson</dsml:oc-value>

  </dsml:objectclass>

  <dsml:attr name="cn">

    <dsml:value>Barbara Jensen</dsml:value>

    <dsml:value>Babs Jensen</dsml:value>

  </dsml:attr>

  <dsml:attr name="sn"><dsml:value>Jensen</dsml:value></dsml:attr>

  <dsml:attr name="uid"><dsml:value>bjensen</dsml:value></dsml:attr>

  <dsml:attr name="mail">

    <dsml:value>bjensen@example.com</dsml:value>

  </dsml:attr>

  <dsml:attr name="givenname">

    <dsml:value>Barbara</dsml:value>

  </dsml:attr>

</dsml:entry>
```

Listing 3.6 The Directory Entry of [Listing 3.5](#) Represented in LDIF Format

```
dn: uid=bjensen,ou=people,dc=example,dc=com

objectclass: top

objectclass: person

objectclass: organizationalPerson

objectclass: inetOrgPerson
```

```
cn: Barbara Jensen  
cn: Babs Jensen  
sn: Jensen  
uid: bjensen  
mail: bjensen@example.com  
givenname: Barbara
```

The DSML format, although more verbose, is more easily read by a machine and doesn't suffer from some of the parsing problems inherent in the LDIF format. In addition, because XML parsers are readily available, it's easy to adapt one to read DSML. Moreover, although not shown in the example, DSML can carry schema information. This capability allows the reader of a DSML document to better understand the meaning of the attributes contained in the DSML file, without schema information having to be exchanged via another method.

Another useful quality of DSML is its compatibility with protocols such as XML-RPC (XML-Remote Procedure Call) and SOAP (Simple Object Access Protocol) that transmit XML, typically using the Hypertext Transfer Protocol (HTTP) and the HTTP-over-SSL (HTTPS) protocols. This feature has several advantages for developers. First, tools for marshaling XML into XML-RPC and SOAP messages are readily available, making development much easier. Second, firewalls are often configured to allow HTTP and HTTPS protocols to pass. These two facts make it possible for DSML, carried in the HTTP or HTTPS protocol, to become the method of choice for carrying directory data on the public Internet.

Does this mean the death of the LDAP protocol? Probably not. Whereas DSML and other XML directory standards will probably appeal to people writing XML applications, LDAP will appeal to people who are directory savvy and directory centric. Although HTTP and HTTPS transports are ubiquitous and well understood, the LDAP protocol is better suited to writing client applications that need to achieve high throughput to the directory. For example, HTTP doesn't support multiple concurrent requests, whereas LDAP does. LDAP will likely be the protocol of choice for communication between the infrastructure components behind e-commerce applications, whereas DSML and HTTP/HTTPS will be the means for communicating directory data between enterprises.

LDAP Extensions and Future Directions Checklists

LDAPv3 can be extended via three mechanisms:

- LDAP controls, which accompany LDAP operations and carry extra information
- LDAP extended operations, which define new LDAP operations
- SASL mechanisms, which define new ways for clients to authenticate to LDAP servers

Future directions for LDAP include

- Continued integration into operating systems and middleware
- Continued standards work centered on protocol extensions and additional core LDAP features
- Increased use of XML, especially DSML

Further Reading

DSML 2.0 Specification. DSML Working Group, 2001. Available on the World Wide Web at <http://www.oasis-open.org/committees/dsml>.

LDAP Programming with Java. R. Weltman and T. Dahbura, Addison-Wesley, 2000.

IETF LDAP Extensions Working Group documents. Available on the World Wide Web at <http://www.ietf.org/ids.by.wg/ldapext.html>.

IETF LDAP Duplication/Replication/Update Protocols Working Group documents. Available on the World Wide Web at <http://www.ietf.org/ids.by.wg/ldup.html>.

Netscape Directory Server 6 Administrator's Guide. Netscape Communications Corporation, 2001. Available on the World Wide Web at <http://enterprise.netscape.com/docs/directory>.

Netscape Directory Server Resource Kit. Netscape Communications Corporation, 2001. Available on the World Wide Web at <http://enterprise.netscape.com/docs/directory>.

Netscape LDAP SDK for C Programmer's Guide. Netscape Communications Corporation, 2001. Available on the World Wide Web at <http://enterprise.netscape.com/docs/directory>.

Netscape LDAP SDK for Java Programmer's Guide. Netscape Communications Corporation, 2001. Available on the World Wide Web at <http://enterprise.netscape.com/docs/directory>.

Looking Ahead

In this chapter we've discussed the ways in which LDAPv3 is extended via controls, extended operations, and SASL mechanisms. We have also described some real-world implementations of these extensions and how to use them. In [Chapter 4](#), Overview of Netscape Directory Server, we will examine a popular implementation of an LDAPv3 server.

Chapter 4. Overview of Netscape Directory Server

- Basic Installation
- A Brief Hands-on Tour of Netscape Directory Server
- Product Focus and Feature Set
- Extending the Netscape Server: A Simple Plug-in Example
- Further Reading
- Looking Ahead

A quick way to learn how to drive a car is to get behind the wheel, turn the key, and ease away from the curb out into traffic. Preferably, someone who is an experienced driver is seated in the passenger seat to coach you and help you master the rules of the road. Similarly, a good way to learn about directory software is to break open the shrink-wrap on a product, install it, and start turning the knobs and buttons to see what it can do.

This chapter provides a hands-on walk-through of one of the leading LDAP products, Netscape Directory Server. You will play the part of the beginning driver, and this chapter will act as the experienced coach sitting next to you. Our tour of version 6 of the Netscape server has four phases:

1. A walk through the basic installation instructions
2. A brief hands-on tour
3. A discussion of the focus and feature set
4. A demonstration of how to extend the server

Now it's time to get behind the wheel, turn the key, and hit the directory services road.

Basic Installation

First locate a system that meets Netscape's minimum requirements. Netscape Directory Server runs on several popular Unix platforms, including Sun Solaris, as well as on Microsoft Windows 2000 Server. Details of the specific system requirements can be found in the *Netscape Directory Server 6 Installation Guide*. This chapter provides detailed installation instructions for Solaris and Microsoft Windows 2000 Server. [Table 4.1](#) summarizes the system requirements for both.

Once you have located a suitable system, place a copy of the Netscape Directory Server 6 installation package on that computer. For production use you must purchase the software, in which case you receive the software on CD-ROM from Netscape. A full-featured version can also be downloaded for evaluation purposes from the AOL Strategic Business Solutions Netscape Enterprise Web site at <http://enterprise.netscape.com>. The remainder of this section assumes that you have placed the installation package in the `/export` directory on a system running Solaris 8 or on a Windows 2000 system in the root of the `C:` drive.

A basic installation of Netscape Directory Server requires three steps:

- Step 1.** Extract and start the setup program.
- Step 2.** Answer a series of installation questions.
- Step 3.** Complete the installation and load data.

Table 4.1. System Requirements for Running Netscape Directory Server

System Feature	Solaris	Windows 2000 Server	Requirement
Operating system	Sun Solaris 8 with Sun's recommended patches	Windows 2000 Server or Advanced Server with Microsoft's latest service pack	
Processor	UltraSPARC or better	Pentium II or better	
Free disk space	200MB	200MB	
Free RAM	256MB	256MB	
Extraction utility	GNU zip (gzip)	Info-ZIP's UnZip, Nico Mak Computing's WinZip, or a similar utility to extract the contents of <code>.zip</code> files	

Installation package filename for version 6.01	directory-6.01-us.sparc-sun- solaris2.8.tar.gz	d601diu.zip
---	---	-------------

To allow the directory server to accept LDAP connections on a TCP port below 1024 (such as the standard port, 389), you must execute the installation as the system superuser (root) on Solaris. On Windows 2000 you should perform the installation as a user that has administrator privileges.

Extracting and Starting the Setup Program

To extract and launch the setup program on Solaris, execute these commands:

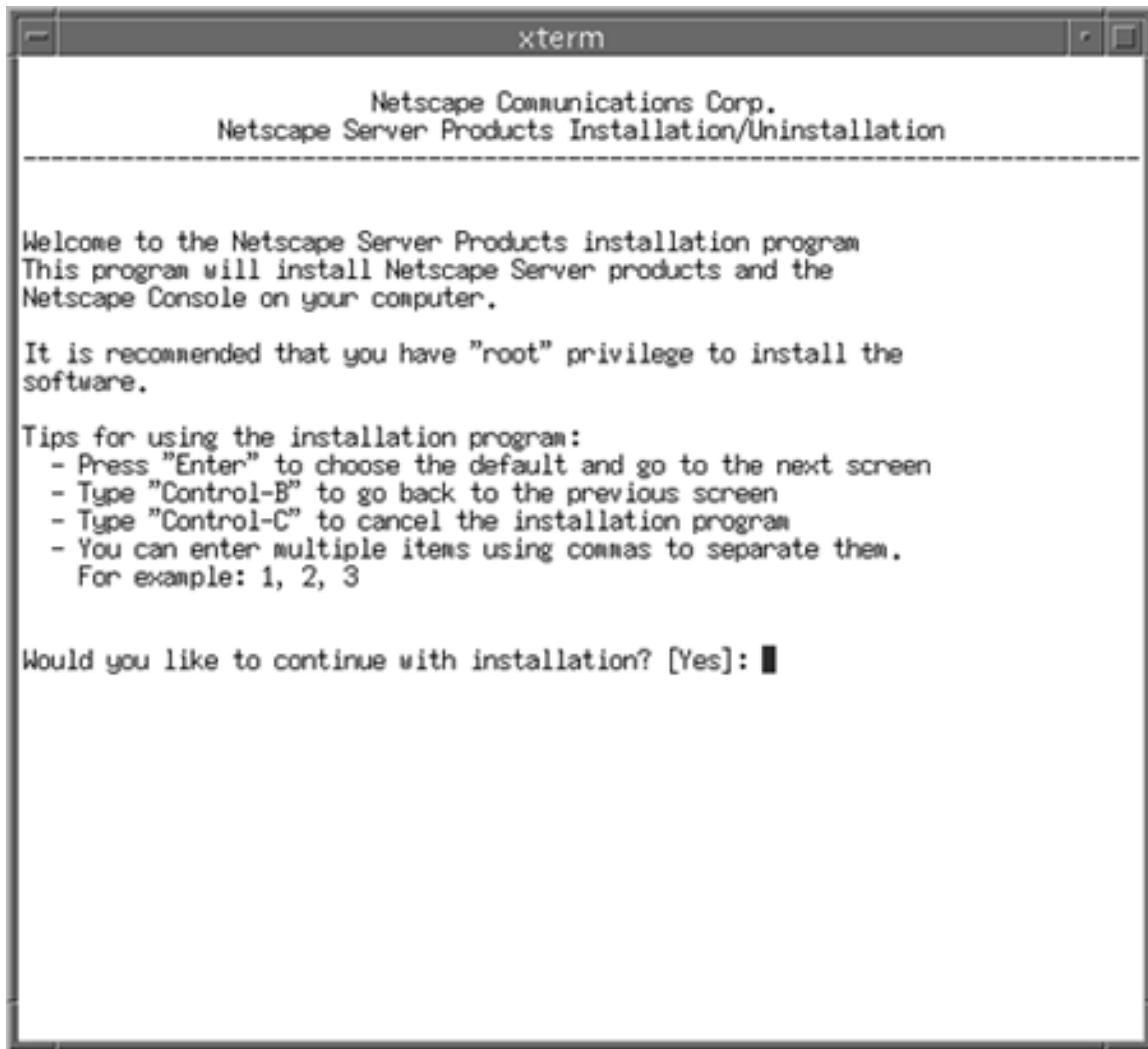
```
su root  
  
mkdir /export/dsinstall  
  
cd /export/dsinstall  
  
gzip -dc ../directory-6.01-us.sparc-sun-solaris2.8.tar.gz | tar -xvof -  
  
.setup
```

To do the same on Windows 2000, execute these commands from the Windows command prompt:

```
md \dsinstall  
  
cd dsinstall  
  
unzip c:\d601diu.zip  
  
setup
```

[Figure 4.1](#) shows the first screen that is presented by the Netscape setup program on Solaris.

Figure 4.1. The First Netscape Directory Server Setup Screen on Solaris



Answering Installation Questions

Netscape supports three installation modes:

1. **Express.** Minimal options; used for product evaluation only.
2. **Typical.** Recommended for most first-time installations.
3. **Custom.** For advanced installations.

In this section the Typical mode is used, which is the default choice. The setup program presents a series of installation-related questions you must answer. On Solaris, follow these steps:

Step 1. Accept the default answers on each setup screen (except on the license screen, where you must type "Yes") until you see a prompt for "Install Location." Type "/export/ds6".

Step 2. Continue and accept the default answers on each setup screen until you see a prompt for "Directory Server Identifier." Type "example".

Step 3. On the next screen, which asks for an "Administrator ID," accept the default ID of "admin" and choose a password (the password is case sensitive). The administrator identity is given full administrative rights to the configuration data in all directory servers.

Step 4. The next screen asks for your directory suffix; this is the base DN, or *naming context*, under which all of your directory's data resides (additional suffixes may be added later). Type "dc=example,dc=com" for the suffix.

Step 5. Accept the default directory manager DN on the next screen (`cn=Directory Manager`) and use the password "secret389" to ensure that the examples in the rest of this chapter work correctly.

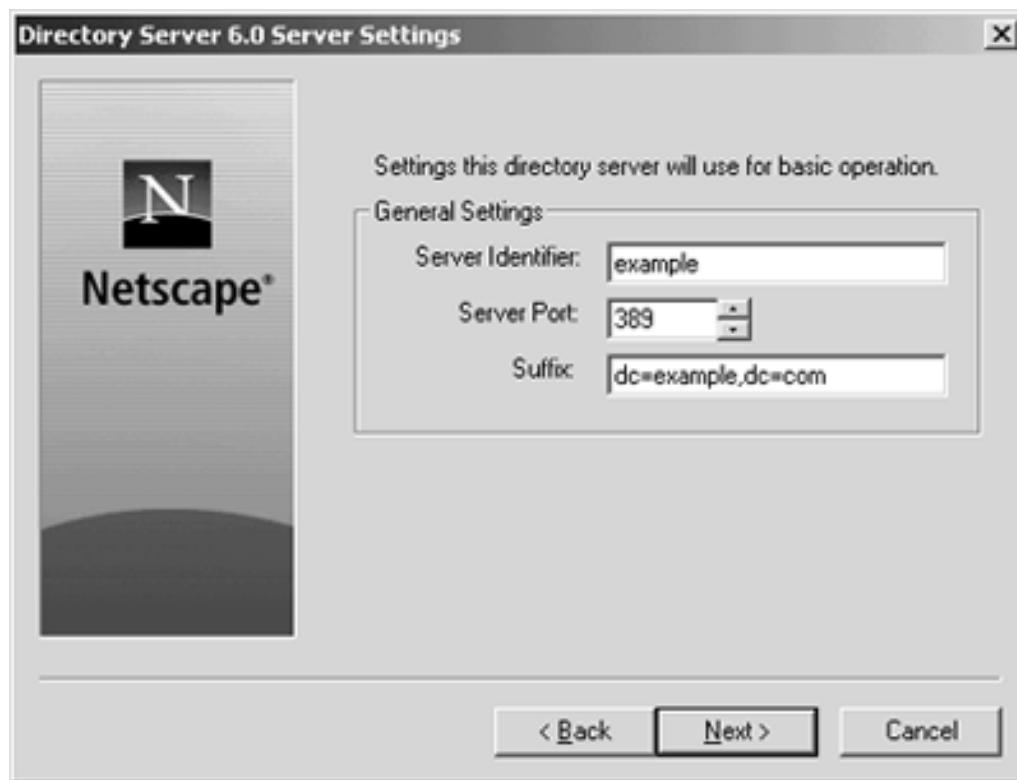
Step 6. Accept the default answers for the remaining setup questions.

You are done when you reach a screen that says, "Extracting Netscape core components." Wait for the setup program to finish placing the directory server files on the disk.

On Microsoft Windows, follow these steps:

Step 1. Accept the default answers until you see a dialog box like the one shown in [Figure 4.2](#) titled **Directory Server 6.0 Server Settings**. Type in "example" as the server identifier, "389" as the server port, and "dc=example,dc=com" as the suffix (naming context).

Figure 4.2. The Directory Server Settings Dialog Box on Windows



Note

By default, Netscape Directory Server is configured to listen for incoming LDAP connections on TCP port 389, and the commands shown in this chapter assume port 389. If another server is already installed that is using port 389, disable or uninstall the other server (which is probably another LDAP server) before installing the Netscape server. If that is not possible, specify a different port in Netscape's Directory Server settings dialog during installation and remember what you chose. Then adjust the LDAP commands used later in this chapter as necessary to specify the port you chose (most commands use port 389 by default). For example, if you choose port 3389 when installing the server, you need to add `-p 3389` to the command-line parameters when issuing an `ldapsearch` or `ldapmodify` command.

Step 2. On the next dialog box, accept the default directory server administrator ID ("admin") and choose a password (the password is case sensitive).

Step 3. Accept the defaults on the remaining dialog boxes, except for the "Directory Server Manager" dialog box, where you should use a password of "secret389" to ensure that the examples in the rest of this chapter work correctly.

Step 4. When you arrive at the final **Configuration Summary** screen, double-check that everything looks correct, and press the **Enter** key or click the **Install** button.

Step 5. Wait for the setup program to finish placing the directory server files on the disk.

Completing the Installation and Loading Sample Data

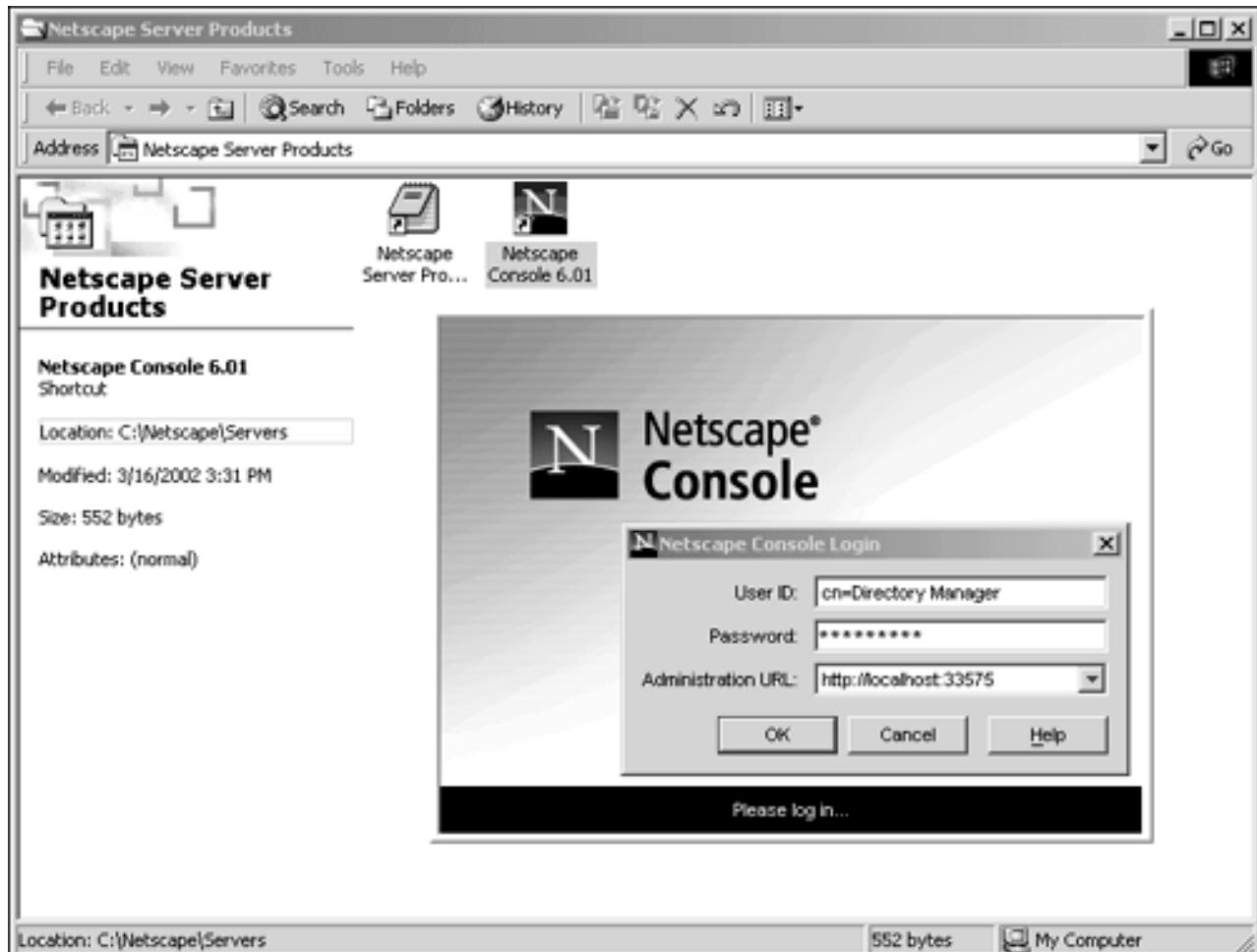
Once the files have been installed on the disk, the Netscape setup program automatically starts Directory Server as well as Administration Server, which is a specialized HTTP server. Netscape Directory Server can be configured and managed with a variety of command-line utilities or through use of a graphical point-and-click console interface named Netscape Console.

Step 1. Start Netscape Console by double-clicking on the **Netscape Console** icon on Microsoft Windows, or by typing these commands on Solaris:

```
cd /export/ds6  
./startconsole
```

Netscape Console is a Java application, and it functions and looks the same on all platforms. [Figure 4.3](#) shows the console login screen.

Figure 4.3. The Netscape Console Login Screen



Step 2. Log in with a user ID of "cn=Directory Manager" and a password of "secret389." Do not change the administration URL; it should be correct by default. After the main console window opens, expand the nodes within the **Servers and Applications** topology tree on the left side of the window until you see a node labeled **Directory Server (example)**. Double-click it. [Figure 4.4](#) shows the Directory Server console window that opens.

Figure 4.4. The Netscape Directory Server Console



Step 3. Load some sample data from the `Example.ldif` file that Netscape ships with its directory server. Click the **Import Databases** task button and type the path for the `Example.ldif` file. On Solaris, it is

```
/export/ds6/slapd-example/ldif/Example.ldif
```

On Microsoft Windows, the correct path is

```
C:\Netscape\Servers\slapd-example\ldif\Example.ldif
```

Step 4. Click the **OK** button to import the data. You should see a message that reads "152 objects imported, 8 objects rejected." Ignore the rejected entries; the setup program created default entries with the same name as the eight rejected ones, and those entries will work for our purposes. The console import task does not overwrite existing data. After the data has been imported, use a text editor to look at the contents of the `Example.ldif` file. [Listing 4.1](#) shows a few entries from `Example.ldif`.

Listing 4.1 A Few Entries from Netscape's `Example.ldif` File

```
dn: dc=example,dc=com
objectclass: top
```

```
objectclass: domain

dc: example

aci: (target ="ldap:///dc=example,dc=com")(targetattr != "userPassword")(version 3.0;acl "Anonymous read-search access";allow (read, search, compare)(userdn = "ldap:///anyone");)

aci: (target="ldap:///dc=example,dc=com") (targetattr = "*")(version 3.0; acl "allow all Admin group"; allow(all) groupdn = "ldap:///cn=Directory Administrators,ou=Groups,dc=example,dc=com";)

dn: ou=People, dc=example,dc=com

objectclass: top

objectclass: organizationalunit

ou: People

aci: (target ="ldap:///ou=People,dc=example,dc=com")(targetattr = "userpassword || telephonenumber || facsimiletelephonenumber")(version 3.0;acl "Allow self entry modification";allow (write)(userdn = "ldap:///self");)

aci: (target ="ldap:///ou=People,dc=example,dc=com")(targetattr != "cn || sn || uid")(targetfilter ="(ou=Accounting)")(version 3.0;acl "Accounting Managers Group Permissions";allow (write) (groupdn = "ldap:///cn=Accounting Managers,ou=groups,dc=example,dc=com");)

aci: (target ="ldap:///ou=People,dc=example,dc=com")(targetattr != "cn || sn || uid")(targetfilter ="(ou=Human Resources)")(version 3.0;acl "HR Group Permissions";allow (write)(groupdn = "ldap:///cn=HR Managers,ou=groups,dc=example,dc=com");)

aci: (target ="ldap:///ou=People,dc=example,dc=com")(targetattr != "cn || sn || uid")(targetfilter ="(ou=Product Testing)")(version 3.0;acl "QA Group Permissions";allow (write)(groupdn = "ldap:///cn=QA Managers,ou=groups,dc=example,dc=com");)

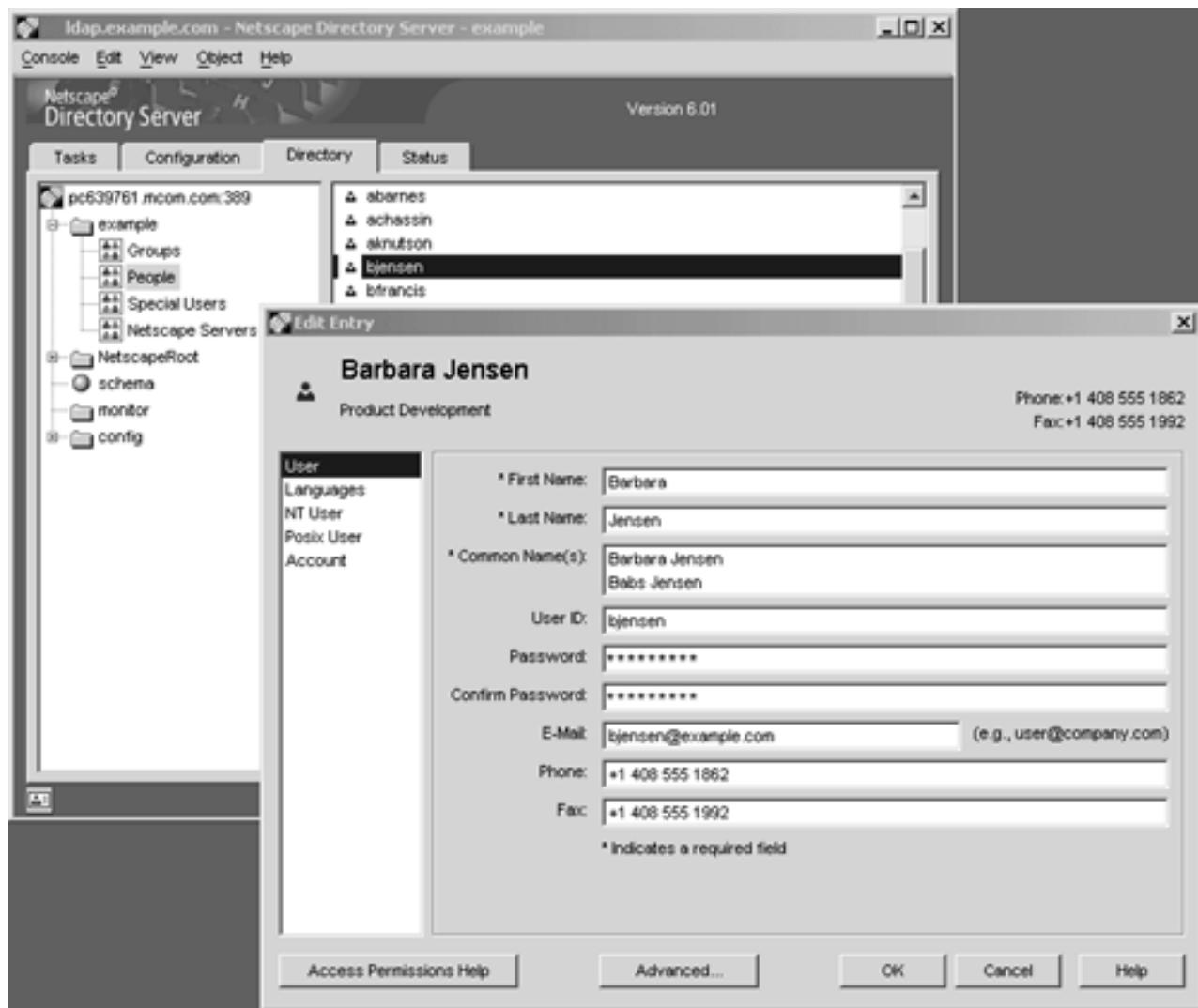
aci: (target ="ldap:///ou=People,dc=example,dc=com")(targetattr != "cn || sn || uid")(targetfilter ="(ou=Product Development)")(version 3.0;acl "Engineering Group Permissions";allow (write)(groupdn = "ldap:///cn=PD Managers,ou=groups,dc=example,dc=com");)
```

```
dn: uid=bjensen, ou=People, dc=example,dc=com
cn: Barbara Jensen
cn: Babs Jensen
sn: Jensen
givenname: Barbara
objectclass: top
objectclass: person
objectclass: organizationalPerson
objectclass: inetOrgPerson
ou: Product Development
ou: People
L: Cupertino
uid: bjensen
mail: bjensen@example.com
telephonenumber: +1 408 555 1862
facsimiletelephonenumber: +1 408 555 1992
roomnumber: 0209
userpassword: hifalutin
```

The `aci` attributes hold Netscape-specific access control information. The access control features of Netscape Directory Server are discussed later in this chapter. Finally, let's confirm that the sample data has been loaded.

Step 5. Click the **Directory** tab near the top of the **Netscape Console** window to see a tree view of the directory information tree (DIT). Click to expand the node labeled **example** (which is a `domain` entry) and select the **People** container (an `organizationalUnit` entry) by clicking on it. A list of user IDs will appear in the right-hand side of the window. The list contains the relative distinguished names (RDNs) of all the entries that are children of the `ou=People,dc=example,dc=com` entry. Double-click any ID to see the attributes of that person. [Figure 4.5](#) shows `bjensen`'s (Barbara Jensen's) entry.

Figure 4.5. Viewing the Barbara Jensen Sample Entry



Step 6. To see all of the LDAP attributes and values in tabular form, click the **Advanced...** button.

Congratulations! You have managed to find first gear, pull away from the curb, and start the car moving down the street.

Team LiB

◀ PREVIOUS

NEXT ▶

A Brief Hands-on Tour of Netscape Directory Server

Now that you have a functioning LDAP server, you can explore its capabilities. Put another way, it is time to shift into second gear. The Netscape server supports all significant LDAP standards, including LDAPv2 (for compatibility with very old applications) and LDAPv3. Netscape also supports many proposed LDAPv3 extensions, which are discussed later in the chapter.

Searching

When you viewed the `dc=example,dc=com` entries after installing the server, you used the **Directory** tab within the Netscape Directory Server console. Now add a new entry using that same interface.

Step 1. Return to the **Directory** tab and select the **People** node on the left side of the window. Within the **Object** menu, execute the **New User** command to open the **Create New User** window. Using the information shown in [Figure 4.6](#), create a new user named Bugs Bunny. Choose a password such as "@home@WB" (the password is not used in any of the examples in this chapter).

Figure 4.6. Creating a New Entry for Bugs Bunny



Next let's execute some directory searches.

Step 2. Use Netscape Console to search for the entry you just added. Locate the entry labeled **People** on the left side of the **Directory** pane. Use the rightmost mouse button to click on the **People** entry and select **Search...** from the context menu that appears. Type "Bugs Bunny" in the text field labeled **for** and press the **Enter** key.

A search result list with one entry should appear. Feel free to try other searches as well. The Netscape Console search window supports several search modes, including one that allows you to type arbitrary LDAP filters.

Step 3. Search for the same entry using the `ldapsearch` command-line tool that is bundled with the Netscape server. Start a Unix shell or a Microsoft Windows command prompt window. On Solaris, type these commands:

```
cd /export/ds6/shared/bin
./ldapsearch -b "dc=example,dc=com" "(cn=Bugs Bunny)"
```

On Microsoft Windows, type these commands:

```
cd \Netscape\Servers\shared\bin  
ldapsearch -b "dc=example,dc=com" "(cn=Bugs Bunny)"
```

In the `ldapsearch` command the argument to the `-b` option is the search base (`dc=example,dc=com`), and the last command-line parameter is the LDAP filter (`(cn=Bugs Bunny)`), which specifies an exact match on the `cn` attribute for the string "Bugs Bunny." [Listing 4.2](#) shows the result of this search: one entry in LDIF format.

Listing 4.2 Result of Search for "Bugs Bunny"

```
version: 1  
  
dn: uid=bbunny,ou=People, dc=example,dc=com  
  
mail: bbunny@example.com  
  
objectClass: top  
  
objectClass: person  
  
objectClass: organizationalPerson  
  
objectClass: inetorgperson  
  
givenName: Bugs  
  
telephoneNumber: +1 555 555 1212  
  
cn: Bugs Bunny  
  
uid: bbunny  
  
sn: Bunny  
  
facsimileTelephoneNumber: +1 555 555 1299
```

There are no surprises here. Netscape has standardized on the `inetOrgPerson` object class defined in RFC 2798 for user entries, so the entry has `objectClass: inetorgperson`. The entry's RDN is `uid=bbunny` because by default Netscape Console uses the `uid` (user id) attribute to name user entries.

Step 4. Try a more complex search. Suppose you want to find all entries that are people in the Product Development department who are located in the Cupertino location. Further, suppose that you want to retrieve only the name, department, and e-mail address of each person. This `ldapsearch` command will do the job (execute this—and all `ldapsearch` commands shown in this chapter—from the same directory as in step 3):

```
ldapsearch -b "dc=example,dc=com" "(&(ou=Product Development)(L=Cupertino))" cn ou mail
```

The search filter consists of two equality filters ANDed together, and the list of requested attributes appears at the end of the command line (`cn ou mail`). This search returns 11 entries. Wouldn't it be nice if they were sorted alphabetically by name? Luckily, Idapsearch supports the LDAPv3 Server-Side Sorting control, and so does Netscape Directory Server.

Step 5. Add the `-x` option to tell the server to sort the entries before returning them, and add the `-Scn` option to specify that the `cn` (common name) attribute should be used as the sort key. Here is the revised `ldapsearch` command:

```
ldapsearch -b "dc=example,dc=com" -x -Scn "(&(ou=Product Development)(L=Cupertino))"  
cn  
 ou mail
```

[Listing 4.3](#) shows the resulting LDIF output.

Listing 4.3 Search Results Sorted by Name

```
version: 1  
  
dn: uid=aworrell, ou=People, dc=example,dc=com  
  
cn: Alan Worrell  
  
ou: Product Development  
  
ou: People  
  
mail: aworrell@example.com
```

```
dn: uid=aknutson, ou=People, dc=example,dc=com  
  
cn: Ashley Knutson  
  
ou: Product Development  
  
ou: People  
  
mail: aknutson@example.com
```

```
dn: uid=bjensen, ou=People, dc=example,dc=com  
  
cn: Barbara Jensen  
  
cn: Babs Jensen  
  
ou: Product Development  
  
ou: People  
  
mail: bjensen@example.com
```

```
dn: uid=cwallace, ou=People, dc=example,dc=com  
  
cn: Cecil Wallace  
  
ou: Product Development  
  
ou: People  
  
mail: cwallace@example.com
```

dn: uid=jmuffle, ou=People, dc=example,dc=com
cn: Jeff Muffle
ou: Product Development
ou: People
mail: jmuffle@example.com

dn: uid=jcampaig, ou=People, dc=example,dc=com
cn: Jody Campaigne
ou: Product Development
ou: People
mail: jcampaig@example.com

dn: uid=jbourke, ou=People, dc=example,dc=com
cn: Jon Bourke
ou: Product Development
ou: People
mail: jbourke@example.com

dn: uid=mlangdon, ou=People, dc=example,dc=com
cn: Marcus Langdon
ou: Product Development
ou: People
mail: mlangdon@example.com

dn: uid=mtalbot, ou=People, dc=example,dc=com
cn: Martin Talbot
ou: Product Development
ou: People
mail: mtalbot@example.com

dn: uid=smason, ou=People, dc=example,dc=com
cn: Sue Mason
ou: Product Development
ou: People
mail: smason@example.com

```
dn: uid=speterso, ou=People, dc=example,dc=com
cn: Sue Peterson
ou: Product Development
ou: People
mail: speterso@example.com
```

Manipulating Netscape Directory Server Databases

Netscape Directory Server uses a high-performance embedded database to store data, and it allows multiple database instances to be active at the same time. Each database instance has a unique name and stores data for one naming context (one subtree within the DIT). The Typical installation used earlier in the chapter created two database instances:

1. **NetscapeRoot**. Holds configuration and administration information that may be shared by more than one Netscape server.
2. **userRoot**. Holds the data that you load into the directory. Earlier in the chapter, you chose `dc=example,dc=com` as the naming context for your data.

Netscape Console has full support for creating and maintaining databases. This section shows how to manipulate directory databases from the command line.

Step 1. Before executing any of the commands shown, ensure that the current working directory is the `slapd-example` instance directory. Start a Unix shell or a Microsoft Windows command prompt window. On Solaris, type this command:

```
cd /export/ds6/slapd-example
```

On Microsoft Windows, type this command:

```
cd \Netscape\Servers\slapd-example
```

In the commands that follow, all of the leading `./` sequences should be omitted if you're working on Microsoft Windows.

Step 2. First use the `suffix2instance` command to display a list of active suffixes (naming contexts) and their corresponding Netscape Directory Server databases:

```
./suffix2instance -s ""
```

The `-s ""` parameter says that all suffixes are to be listed; if you wanted to list suffixes under only a specific subtree, you would include the subtree DN after the `-s`. The output produced is

```
suffix, Instance name pair(s) under "":
suffix "o=NetscapeRoot"; instance name "NetscapeRoot"
suffix "dc=example,dc=com"; instance name "userRoot"
```

Step 3. Shut down the server using the `stop-slapd` command and replace the contents of the `userRoot` database with new data, a process known as *bulk-loading*. [Listing 4.4](#) shows how to use Netscape's `ldif2db` command to load the `Example.ldif` file that is bundled with the server. The commands you need to type are shown in bold.

Listing 4.4 Bulk-Loading of `Example.ldif` Using the `ldif2db` Command

```
./stop-slapd

./ldif2db -n userRoot -i - <ldif/Example.ldif

importing data ...

[26/Aug/2002:16:04:18 -0500] - import userRoot: Index buffering enabled with bucket size 15

[26/Aug/2002:16:04:18 -0500] - import userRoot: Beginning import job...

[26/Aug/2002:16:04:18 -0500] - import userRoot: Processing file stdin

[26/Aug/2002:16:04:19 -0500] - import userRoot: Finished scanning file stdin (160 entries)

[26/Aug/2002:16:04:19 -0500] - import userRoot: Workers finished; cleaning up...

[26/Aug/2002:16:04:22 -0500] - import userRoot: Workers cleaned up.

[26/Aug/2002:16:04:22 -0500] - import userRoot: Cleaning up producer thread...

[26/Aug/2002:16:04:22 -0500] - import userRoot: Indexing complete. Post-processing...

[26/Aug/2002:16:04:22 -0500] - import userRoot: Flushing caches...

[26/Aug/2002:16:04:22 -0500] - import userRoot: Closing files...

[26/Aug/2002:16:04:22 -0500] - import userRoot: Import complete. Processed 160 entries in

➡ 4 seconds. (40.00 entries/sec)
```

The `-n userRoot` parameter selects the `userRoot` database instance, `-i -` indicates that the LDIF file is provided on standard input, and `<ldif/Example.ldif` causes the shell to send the contents of the `Example.ldif` file to `ldif2db`'s standard input.

[Listing 4.5](#) demonstrates the reverse process (creating an LDIF file from an existing database). On Windows, ensure that a directory named `\tmp` exists in the root of the drive where you installed Netscape Directory Server, or use a different pathname for the `example-dump.ldif` output file.

Listing 4.5 Dumping a Database Using the `db2ldif` Command

```
./db2ldif -n userRoot -a /tmp/example-dump.ldif

ldiffile: /tmp/example-dump.ldif

[26/Aug/2002:16:35:28 -0500] - export userRoot: Processed 160 entries (100%).

more < /tmp/example-dump.ldif

version: 1

# entry-id: 1
```

```

dn: dc=example,dc=com
objectClass: top
objectClass: domain
dc: example
aci: (target ="ldap:///dc=example,dc=com")(targetattr !="userPassword")(version 3.0;acl "Anonymous read-search access";allow (read, search, compare)(userdn = "ldap:///anyone"))
aci: (target="ldap:///dc=example,dc=com") (targetattr = "")(version 3.0; acl "allow all Admin group"; allow(all) groupdn = "ldap:///cn=Directory Administrators,ou=Groups,dc=example,dc=com")
nsUniqueId: 093e751b-1dd211b2-80000000-00000000

# entry-id: 2
dn: ou=Groups, dc=example,dc=com
objectClass: top
objectClass: organizationalunit
ou: Groups
nsUniqueId: 093e751c-1dd211b2-80000000-00000000

# entry-id: 3
dn: cn=Directory Administrators, ou=Groups, dc=example,dc=com
cn: Directory Administrators
objectClass: top
objectClass: groupofuniquenames
ou: Groups
uniqueMember: uid=kvaughan, ou=People, dc=example,dc=com
uniqueMember: uid=rdaugherty, ou=People, dc=example,dc=com
uniqueMember: uid=hmiller, ou=People, dc=example,dc=com
nsUniqueId: 093e751d-1dd211b2-80000000-00000000
--More--( 2% )

```

The `-n userRoot` parameter indicates that entries within the default directory database should be extracted (`-s "dc=example,dc=com"` may be used instead to extract the entries on the basis of the LDAP subtree that contains them). The `-a -` parameter says that the output should be sent to standard output, and `>example-dump.ldif` tells the shell to capture the output in a file named `example-dump.ldif`.

[Listing 4.5](#) shows the first portion of the file, as viewed using the `more` command. Each entry in the LDIF file includes an `nsUniqueId` attribute, which is an operational attribute that holds a global unique identifier

(GUID) generated by Netscape Directory Server. The `nsUniqueID` values are used internally by Netscape to support replication and are preserved when entries are renamed. These values may also be used by LDAP clients that need to track entries without relying on the entry's DN. [Listing 4.6](#) shows how to dump a Netscape Directory Server database in Directory Services Markup Language (DSML) format rather than LDIF format. DSML is an XML-based format for representing directory data.

Listing 4.6 Dumping a Database Using the `db2dsml` Command

```
./db2dsml -n userRoot -a /tmp/example-dump.dsml

ldiffile: -

[26/Aug/2002:16:41:12 -0500] - export userRoot: Processed 160 entries (100%).
more < /tmp/example-dump.dsml

<?xml version="1.0" encoding="UTF-8" ?>

<dsml:dsml xmlns:dsml="http://www.dsml.org/DSML">

  <dsml:directory-entries>

    <dsml:entry dn="dc=example,dc=com">

      <dsml:objectclass>
        <dsml:oc-value>top</dsml:oc-value>
        <dsml:oc-value>domain</dsml:oc-value>
      </dsml:objectclass>

      <dsml:attr name="dc">
        <dsml:value>example</dsml:value>
      </dsml:attr>

      <dsml:attr name="nsuniqueid">
        <dsml:value>093e751b-1dd211b2-80000000-00000000</dsml:value>
      </dsml:attr>

      <dsml:attr name="aci">
        <dsml:value>(target = "ldap:///dc=example,dc=com") (targetattr != "userPassword") (
          ↪ version 3.0;acl "Anonymous read-search access" ;allow (read, search, compare)(userdn =
          ↪ "ldap:///anyone");)</dsml:value>
        <dsml:value>(target="ldap:///dc=example,dc=com") (targetattr = "")(version 3.0;
          ↪ acl "allow all Admin group"; allow(all) groupdn = "ldap:///cn=Directory Administrators,
          ↪ ou=Groups, dc=example,dc=com");</dsml:value>
      </dsml:attr>
    </dsml:entry>

    <dsml:entry dn="ou=Groups, dc=example,dc=com">

      <dsml:objectclass>
        <dsml:oc-value>top</dsml:oc-value>
```

```

<dsml:oc-value>organizationalunit</dsml:oc-value>

</dsml:objectclass>

<dsml:attr name="nsuniqueid">

  <dsml:value>093e751c-1dd211b2-80000000-00000000</dsml:value>

</dsml:attr>

<dsml:attr name="ou">

  <dsml:value>Groups</dsml:value>

</dsml:attr>

</dsml:entry>

<dsml:entry dn="cn=Directory Administrators, ou=Groups, dc=example,dc=com">

  <dsml:objectclass>

    <dsml:oc-value>top</dsml:oc-value>

    <dsml:oc-value>groupofuniquenames</dsml:oc-value>

  </dsml:objectclass>

  <dsml:attr name="cn">

    <dsml:value>Directory Administrators</dsml:value>

  </dsml:attr>

  <dsml:attr name="nsuniqueid">

    <dsml:value>093e751d-1dd211b2-80000000-00000000</dsml:value>

  </dsml:attr>

  <dsml:attr name="uniqueMember">

    <dsml:value>uid=kvaughan, ou=People, dc=example,dc=com</dsml:value>

    <dsml:value>uid=rdaugherty, ou=People, dc=example,dc=com</dsml:value>

    <dsml:value>uid=hmiller, ou=People, dc=example,dc=com</dsml:value>

  </dsml:attr>

  <dsml:attr name="ou">

    <dsml:value>Groups</dsml:value>

  </dsml:attr>

</dsml:entry>

--More--(1%)

```

The DSML output is more verbose than LDIF, but it is useful if you want to work with XML-savvy tools and with other applications that understand XML. See [Chapter 3](#), LDAPv3 Extensions, for more information about DSML.

Another important database maintenance task is creating backups of the data. Netscape supports *hot backups* —that is, backups that are performed while the directory server is running and accepting updates. The Netscape server stores its active database files under a subdirectory named **db**, and the hot backup process makes a transactionally consistent copy of all the files.

Step 1. While the server is running, use the `db2bak` command as shown in [Listing 4.7](#) to create a complete backup of the directory data. The sample run is from a Solaris system. Each `.db3` file stores some entry data or an attribute index; on Microsoft Windows the pathnames of the files will be different.

Listing 4.7 Starting the Server and Creating a Backup

```
./start-slapd

./db2bak

[26/Aug/2002:17:01:21 -0500] - Backing up file 1 (/export/ds6/slapd-example/bak/
➥ 2002_08_17_01_20/userRoot/id2entry.db3)

[26/Aug/2002:17:01:21 -0500] - Backing up file 2 (/export/ds6/slapd-example/bak/
➥ 2002_08_17_01_20/userRoot/entrydn.db3)

[26/Aug/2002:17:01:21 -0500] - Backing up file 3 (/export/ds6/slapd-example/bak/
➥ 2002_08_17_01_20/userRoot/parentid.db3)

[26/Aug/2002:17:01:21 -0500] - Backing up file 4 (/export/ds6/slapd-example/bak/
➥ 2002_08_17_01_20/userRoot/aci.db3)

[26/Aug/2002:17:01:21 -0500] - Backing up file 5 (/export/ds6/slapd-example/bak/
➥ 2002_08_17_01_20/userRoot/uid.db3)

[26/Aug/2002:17:01:21 -0500] - Backing up file 6 (/export/ds6/slapd-example/bak/
➥ 2002_08_17_01_20/userRoot/nsUniqueId.db3)

[26/Aug/2002:17:01:21 -0500] - Backing up file 7 (/export/ds6/slapd-example/bak/
➥ 2002_08_17_01_20/userRoot/objectclass.db3)

[26/Aug/2002:17:01:21 -0500] - Backing up file 8 (/export/ds6/slapd-example/bak/
➥ 2002_08_17_01_20/userRoot/mail.db3)

[26/Aug/2002:17:01:21 -0500] - Backing up file 9 (/export/ds6/slapd-example/bak/
➥ 2002_08_17_01_20/userRoot/cn.db3)

...

[26/Aug/2002:17:01:22 -0500] - Backing up file 25 (/export/ds6/slapd-example/bak/
➥ 2002_08_17_01_20/NetscapeRoot/sn.db3)

[26/Aug/2002:17:01:22 -0500] - Backing up file 26 (/export/ds6/slapd-example/bak/
➥ 2002_08_17_01_20/NetscapeRoot/givenName.db3)

[26/Aug/2002:17:01:22 -0500] - Backing up file 27 (/export/ds6/slapd-example/bak/
➥ 2002_08_17_01_20/NetscapeRoot/uid.db3)

[26/Aug/2002:17:01:22 -0500] - Backing up file 28 (/export/ds6/slapd-example/bak/
➥ 2002_08_17_01_20/NetscapeRoot/uniquemember.db3)

[26/Aug/2002:17:01:22 -0500] - Backing up file 29 (/export/ds6/slapd-example/bak/
```

```
↳ 2002_08_26_17_01_20/log.0000000001)

[26/Aug/2002:17:01:22 -0500] - Backing up file 30 (/export/ds6/slapped-example/bak/
↳ 2002_08_26_17_01_20/DBVERSION)
```

The `db2bak` command creates a consistent copy of all the database files. By default, the files are stored under a directory in the file system named according to the current date and time (`/export/ds6/slapped-example/bak/2002_08_26_17_01_20` in the sample run shown).

Step 2. If running on Microsoft Windows, use the commands shown in [Listing 4.8](#) to restore the directory server data from the backup you created. First, stop the server using the `stop-slapd` command (databases cannot be restored while the server is running). Next, simulate loss of the active database files by using the `del` or `rm` command to remove all of the `.db3` files. Finally, execute the `bak2db` command to restore the database files from the backup.

Listing 4.8 Restoring a Database from a Backup

```
stop-slapd

del /S/Q db\*.db3

Deleted file - C:\Netscape\Servers\slapped-example\db\DBVERSION

Deleted file - C:\Netscape\Servers\slapped-example\db\log.0000000001

Deleted file - C:\Netscape\Servers\slapped-example\db\NetscapeRoot\aci.db3

Deleted file - C:\Netscape\Servers\slapped-example\db\NetscapeRoot\ancestorid.db3

Deleted file - C:\Netscape\Servers\slapped-example\db\NetscapeRoot\cn.db3

...

Deleted file - C:\Netscape\Servers\slapped-example\db\userRoot\parentid.db3

Deleted file - C:\Netscape\Servers\slapped-example\db\userRoot\sn.db3

Deleted file - C:\Netscape\Servers\slapped-example\db\userRoot\telephoneNumber.db3

Deleted file - C:\Netscape\Servers\slapped-example\db\userRoot\uid.db3

Deleted file - C:\Netscape\Servers\slapped-example\db\userRoot\uniqueMember.db3
```

```
bak2db C:\Netscape\Servers\slapped-example\bak\2002_08_26_170120
```

```
[27/Aug/2002:10:06:40 -0500] - Restoring file 1 (C:/Netscape/Servers/slapped-example/db/
↳ DBVERSION)

[27/Aug/2002:10:06:40 -0500] - Restoring file 2 (C:/Netscape/Servers/slapped-example/db/log.
↳ 0000000001)

[27/Aug/2002:10:06:43 -0500] - Restoring file 3 (C:/Netscape/Servers/slapped-example/db/
↳ NetscapeRoot/aci.db3)

[27/Aug/2002:10:06:43 -0500] - Restoring file 4 (C:/Netscape/Servers/slapped-example/db/
↳ NetscapeRoot/ancestorid.db3)

[27/Aug/2002:10:06:43 -0500] - Restoring file 5 (C:/Netscape/Servers/slapped-example/db/
```

```

↳ NetscapeRoot/cn.db3)

[27/Aug/2002:10:06:43 -0500] - Restoring file 6 (C:/Netscape/Servers/slapd-example/db/
↳ NetscapeRoot/entrydn.db3)

[27/Aug/2002:10:06:43 -0500] - Restoring file 7 (C:/Netscape/Servers/slapd-example/db/
↳ NetscapeRoot/givenName.db3)

[27/Aug/2002:10:06:43 -0500] - Restoring file 8 (C:/Netscape/Servers/slapd-example/db/
↳ NetscapeRoot/id2entry.db3)

[27/Aug/2002:10:06:43 -0500] - Restoring file 9 (C:/Netscape/Servers/slapd-example/db/
↳ NetscapeRoot/nsUniqueId.db3)

...
[27/Aug/2002:10:06:43 -0500] - Restoring file 25 (C:/Netscape/Servers/slapd-example/db/
↳ userRoot/objectclass.db3)

[27/Aug/2002:10:06:43 -0500] - Restoring file 26 (C:/Netscape/Servers/slapd-example/db/
↳ userRoot/parentid.db3)

[27/Aug/2002:10:06:43 -0500] - Restoring file 27 (C:/Netscape/Servers/slapd-example/db/
↳ userRoot/sn.db3)

[27/Aug/2002:10:06:43 -0500] - Restoring file 28 (C:/Netscape/Servers/slapd-example/db/
↳ userRoot/telephoneNumber.db3)

[27/Aug/2002:10:06:43 -0500] - Restoring file 29 (C:/Netscape/Servers/slapd-example/db/
↳ userRoot/uid.db3)

[27/Aug/2002:10:06:43 -0500] - Restoring file 30 (C:/Netscape/Servers/slapd-example/db/
↳ userRoot/uniquemember.db3)

```

If running on Solaris, use the following commands instead of the ones shown in [Listing 4.8](#):

```

./stop-slapd

rm -rf db/*/*.db3

./bak2db /export/ds6/slapd-example/bak/2002_08_26_17_01_20

```

The pathname used in the `bak2db` command must match that produced by the `db2bak` command you already executed. If necessary, perform a directory listing of the `bak` directory to find the correct name.

Step 3. Execute the `start-slapd` and `ldapsearch` commands shown in [Listing 4.9](#) to restart the server and perform a quick one-level search to verify that the data has been restored. The commands and output shown are from a Windows system. The output indicates that the restore was successful.

```
start-slapd
```

Listing 4.9 Checking That the Data Has Been Restored

```
C:\Netscape\Servers\slapd-example>net start slapd-example  
The Netscape Directory Server 6 (example) service is starting.  
The Netscape Directory Server 6 (example) service was started successfully.
```

```
cd \Netscape\Servers\shared\bin  
  
ldapsearch -v -b "dc=example,dc=com" -s one "(objectClass=*)"  
  
version: 1  
  
dn: ou=Groups, dc=example,dc=com  
  
objectClass: top  
  
objectClass: organizationalunit  
  
ou: Groups  
  
  
dn: ou=People, dc=example,dc=com  
  
objectClass: top  
  
objectClass: organizationalunit  
  
ou: People  
  
  
dn: ou=Special Users,dc=example,dc=com  
  
objectClass: top  
  
objectClass: organizationalUnit  
  
ou: Special Users  
  
description: Special Administrative Accounts  
  
  
dn: ou=Netscape Servers,dc=example,dc=com  
  
objectClass: top  
  
objectClass: organizationalUnit  
  
ou: Netscape Servers  
  
description: Standard branch for Netscape Server registration
```

Controlling Access to Directory Data

Netscape Directory Server allows directory administrators to control access to all data in the DIT down to the entry, attribute, and value levels. Access control instructions are stored in operational attributes named **aci**. The **aci** attributes may appear in any entry, and by default they affect all entries within the subtree where they are stored. For example, access control instructions that are stored in the entry `dc=example,dc=com` govern access to all entries at and below `dc=example,dc=com`, such as those within `ou=People,dc=example,dc=com` and within `ou=Groups,dc=example,dc=com`.

This section introduces Netscape Directory Server's access control mechanism by demonstrating how to add an access control instruction to allow one entry to impersonate another. The actual impersonation is done using the LDAP Proxied Authorization control (see [Chapter 3](#), LDAPv3 Extensions, for more information on this control). The Netscape access control mechanism uses a set of operation-specific rights to control access. To be able to use the Proxied Authorization control, the entry must have **proxy** rights for the entry you wish to impersonate.

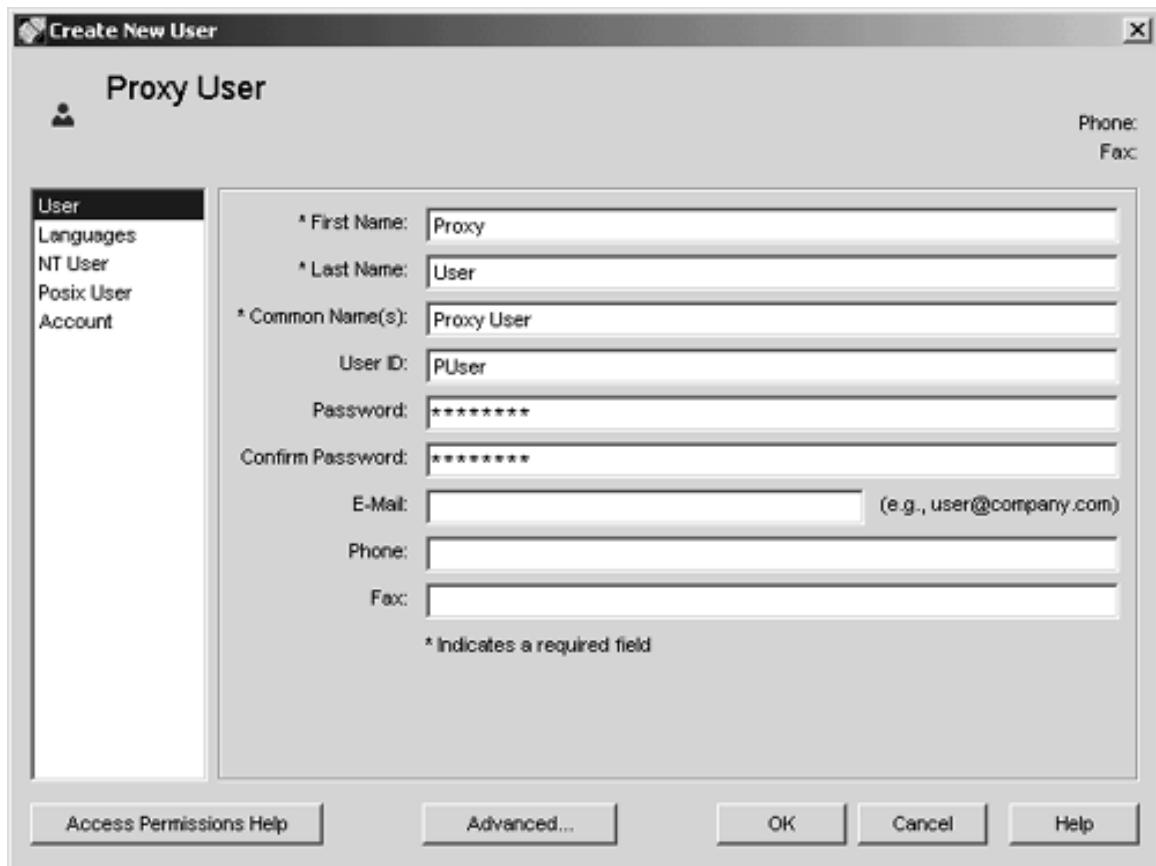
Typically, the **proxy** right is granted to an administrative entry or to an entry that represents a software application. Follow these steps to add a special user to which you will grant the **proxy** right:

Step 1. Start Netscape Console if it is not already running, and log in, entering "cn=Directory Manager" as the DN and "secret389" as the password.

Step 2. Open the Directory Server Administration Console for the sample server by clicking the **Directory** tab, and click to expand the directory node labeled **example**. You should see four organizational unit entries: **Groups**, **People**, **Special Users**, and **Netscape Servers**.

Step 3. Select **Special Users** by clicking it. On the **Object** menu, choose the **New User** command. Create a user named "Proxy User" and give it the password "lrtw,YB!". [Figure 4.7](#) shows how the **Create New User** screen should look before you click the **OK** button.

Figure 4.7. Creating a New Entry Named "Proxy User"



Step 4. Confirm that the new user is not yet able to act as a proxy for other entries. The tests performed rely on the fact that access control instructions to allow people to modify their own entry are included in the [Example.ldap](#) file. [Listing 4.10](#) shows two **ldapmodify** commands that both authenticate as the Proxy User entry and attempt to change the **userPassword** attribute within Sam Carter's entry. The first command does not use Proxied Authorization; the second one does. The text you need to type is shown in bold. Press return twice to insert a blank line after the "**-**" character that appears on a line by itself. On Microsoft Windows, omit the leading **./** from the commands.

Listing 4.10 Failed Attempts to Modify an Entry

```
./ldapmodify -D "uid=puser,ou=Special Users,dc=example,dc=com" -w "lrtw,YB!"
```

```

dn: uid=scarter,ou=People,dc=example,dc=com
changetype: modify
replace: userPassword
userPassword: mySecret42
-
modifying entry uid=scarter,ou=People,dc=example,dc=com
ldap_modify: Insufficient access
ldap_modify: additional info: Insufficient 'write' privilege to the 'userPassword'
➡ attribute of entry 'uid=scarter,ou=people,dc=example,dc=com'.

./ldapmodify -D "uid=puser,ou=Special Users,dc=example,dc=com" -w "lrtw,YB!" -Y "dn:
➡ uid=scarter,ou=People,dc=example,dc=com"
dn: uid=scarter,ou=People,dc=example,dc=com
changetype: modify
replace: userPassword
userPassword: mySecret42
-
modifying entry uid=scarter,ou=People,dc=example,dc=com
ldap_modify: Insufficient access
ldap_modify: additional info: Insufficient 'write' privilege to the 'userPassword'
➡ attribute of entry 'uid=scarter,ou=people,dc=example,dc=com'.

```

The first `ldapmodify` command failed because the Proxy User entry is treated just like any other entry, and therefore it does not have permission to modify Sam Carter's entries. The second `ldapmodify` command failed because Proxy User does not yet have permission to impersonate other users, which makes using the `-y` (Proxied Authorization) option unhelpful. If the server allowed Proxy User to impersonate Sam Carter, Proxy User would be able to modify Sam's entry (just as Sam Carter himself could).

Next, follow these steps to begin the process of adding a new access control instruction (ACI) to the `ou=People,dc=example,dc=com` subtree:

Step 1. Return to the Netscape Directory Server console **Directory** tab and select the node labeled **People**.

Step 2. From the **Object** menu, choose the **Set Access Permissions** command. A **Manage Access Control** window will open. It shows a list of five access control instructions, which are included in the `Example.ldif` file.

Step 3. Click the **New** button. [Figure 4.8](#) shows the **Edit ACI** window that opens. This window has an ACI **Name** field, as well as five tabs:

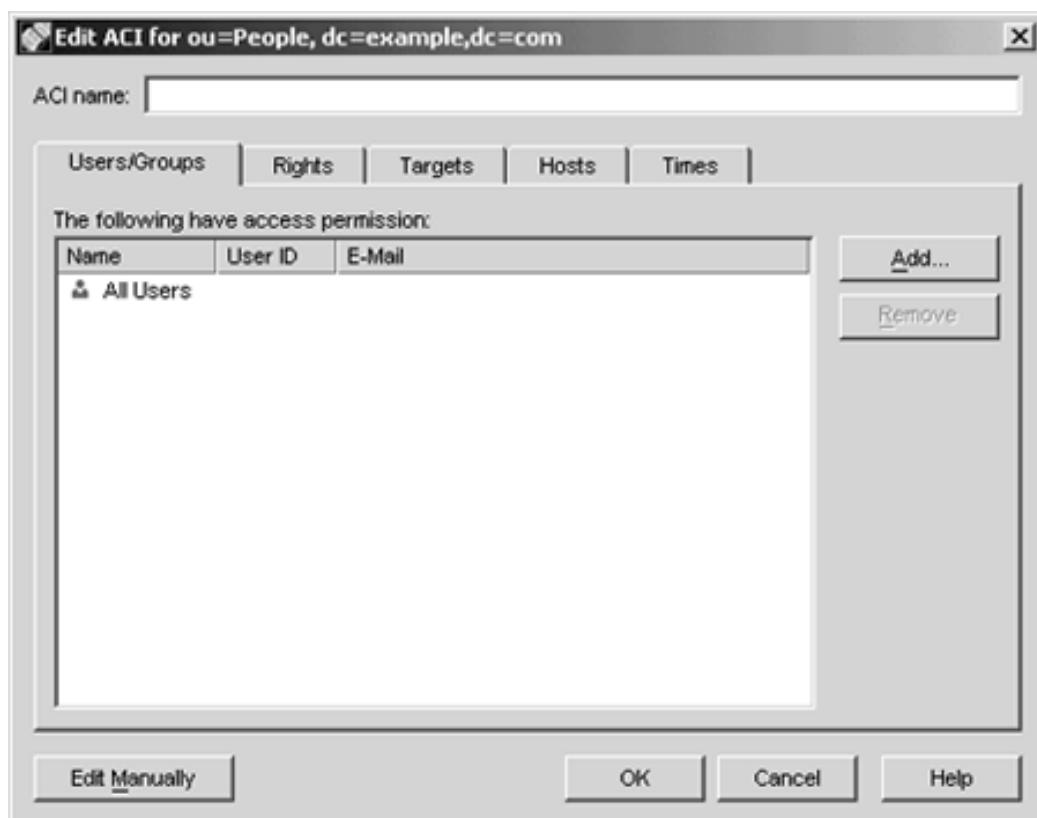
1. **Users/Groups.** Allows you to add users and groups that are given the rights granted by this access control instruction. By default, the set of users and groups is set to **All Users**.

2. **Rights**. Allows you to specify what the users and groups are allowed to do. The available rights are these:
 - **read**. See attribute values, for example, by asking that an attribute be returned from an LDAP search operation.
 - **compare**. Compare attribute values, for example, by using an LDAP compare operation.
 - **search**. Determine if attribute values exist, for example, by using an attribute within an LDAP search filter.
 - **selfwrite**. Allow an entry to add its own DN to an attribute.
 - **write**. Modify attributes, for example, by using an LDAP modify operation.
 - **delete**. Remove entries by using an LDAP delete operation.
 - **add**. Add entries by using an LDAP add operation.
 - **proxy**. Impersonate another entry by using the LDAPv3 Proxied Authorization control.

By default, the new ACI grants all rights except **proxy**.

3. **Targets**. Allows you to limit the set of entries and attributes that this access control instruction governs. For example, you can specify that an ACI governs only the **userPassword** attribute of entries that match the filter (**objectClass= inetOrgPerson**). By default, the ACI affects all entries at and below the entry that contains the **aci** attribute, and all attributes within those entries.
4. **Hosts**. Allows you to limit access based on the LDAP client's host name or IP address. By default, all hosts are treated the same.
5. **Times**. Allows you to limit access based on time of day and day of the week. For example, you could limit access to the hours 8 A.M. to 6 P.M. on weekdays. By default, no time- or day-based restrictions are enforced.

Figure 4.8. The Netscape Directory Server Console Edit ACI Window



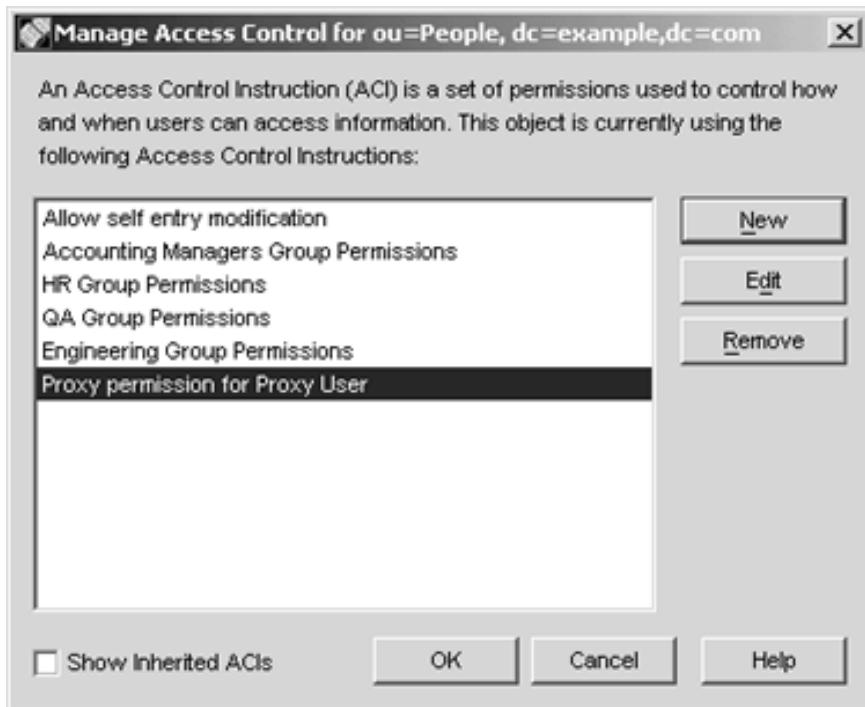
Next, follow these steps to grant the **proxy** right to Proxy User:

Step 1. Type the phrase "Proxy permission for Proxy User" in the ACI **Name** text field.

Step 2. Make sure the **Users/Groups** tab is active and that **All Users** is selected. Click the **Remove** button to delete "All Users" and then click the **Add** button to open the **Add Users and Groups** window. Search for the Proxy User entry and add it to the access permission list. Click the **OK** button to close the **Add Users and Groups** window.

Step 3. Click the **Rights** tab and make sure **Proxy** is the only right checked (you must uncheck all the other rights and then check **Proxy**). Click the **OK** button to save your new access control instruction. [Figure 4.9](#) shows the updated **Manage Access Control** window that includes your new proxy permission ACI.

Figure 4.9. The Manage Access Control Window for `ou=People,dc=example,dc=com`



Step 4. Using Netscape Console to manage access control simplifies the process considerably and allows you to avoid the messy syntax of the **aci** attributes. Use the **ldapsearch** command shown in [Listing 4.11](#) to list the **aci** values present in the `ou=People,dc=example,dc=com` entry.

Listing 4.11 Examining `aci` Values from the Command Line

```
./ldapsearch -b "ou=People,dc=example,dc=com" -s base "(objectClass=*)" aci
version: 1
dn: ou=People, dc=example,dc=com
aci: (target ="ldap://ou=People,dc=example,dc=com")(targetattr ="userpassword
|| telephonenumber || facsimiletelephonenumber")(version 3.0;acl "Allow self entry
modification";allow (write)(userdn = "ldap:///self";)
aci: (target ="ldap://ou=People,dc=example,dc=com")(targetattr !="cn || sn ||
uid")(targetfilter ="(ou=Accounting)")(version 3.0;acl "Accounting Managers
Group Permissions";allow (write) (groupdn = "ldap:///cn=Accounting Managers
,ou=groups,dc=example,dc=com");)
```

```

aci: (target ="ldap:///ou=People,dc=example,dc=com")(targetattr !="cn || sn ||
uid")(targetfilter ="(ou=Human Resources)")(version 3.0;acl "HR Group Permi
ssions";allow (write)(groupdn = "ldap:///cn=HR Managers,ou=groups,dc=example
,dc=com"));

aci: (target ="ldap:///ou=People,dc=example,dc=com")(targetattr !="cn || sn ||
uid")(targetfilter ="(ou=Product Testing)")(version 3.0;acl "QA Group Permisi
ons";allow (write)(groupdn = "ldap:///cn=QA Managers,ou=groups,dc=example,
dc=com"));

aci: (target ="ldap:///ou=People,dc=example,dc=com")(targetattr !="cn || sn ||
uid")(targetfilter ="(ou=Product Development)")(version 3.0;acl "Engineering
Group Permissions";allow (write)(groupdn = "ldap:///cn=PD Managers,ou=grou
ps,dc=example,dc=com"));

aci: (targetattr = "*") (version 3.0;acl "Proxy permission for Proxy User";all
ow (proxy)(userdn = "ldap:///uid=PUser,ou=Special Users,dc=example,dc=com");
)

```

The ACI that was just added is the last one. It targets all attributes (*) and grants the proxy right to the entry with the DN `uid=PUser,ou=Special Users, dc=example,dc=com`. That sounds correct, although you're probably glad you did not have to type it yourself.

Step 5. Use an `ldapmodify` command to impersonate Sam Carter and modify his entry. [Listing 4.12](#) shows the command and the result. The entry modification was a success; the same command that failed earlier worked this time.

Listing 4.12 Using Proxied Authorization to Modify an Entry

```

./ldapmodify -v -D "uid=puser,ou=Special Users,dc=example,dc=com" -w "lrtw,YB!" -Y "dn:
➥ uid=scarter,ou=People,dc=example,dc=com"

ldapmodify: started Wed Aug 27 12:24:35 2002

ldap_init( localhost, 389 )

dn: uid=scarter,ou=People,dc=example,dc=com
changetype: modify
replace: userPassword
userPassword: mySecret42
-
replace userPassword:
mySecret42

```

```
modifying entry uid=scarter,ou=People,dc=example,dc=com  
modify complete  
<Ctrl-C>
```

You can learn more about Netscape Directory Server's access control features by reading Netscape's documentation.

Changing the Server Configuration Using LDAP

Netscape Directory Server exposes all of its configuration as a series of directory server–specific entries that reside within a subtree named `cn=config`. The directory server's configuration entries and attributes are documented in Netscape's *Directory Server Configuration, Command, and File Reference* manual. You can examine all of the stored configuration entries and attributes by viewing the `config/dse.ldif` file within the server instance directory. You can change the configuration using Netscape Console, by stopping the server and editing the `config/dse.ldif` file, or by using LDAP modify operations that target the entries within the `cn=config` subtree.

This section demonstrates how to change a configuration setting using LDAP. Specifically, you will change a setting so that user data can no longer be updated; that is, LDAP add and modify operations will be rejected. Changing configuration settings using LDAP is useful when you're writing automated scripts that help manage a directory service deployment. The setting that you will learn how to change using the `ldapmodify` command-line utility can also be changed using the Netscape Directory Server console. To find the setting that controls whether the `userRoot` database will accept or reject LDAP updates, follow these steps:

Step 1. Open the Directory Server console and click on the **Configuration** tab.

Step 2. Click on the plus sign next to the **Data** node to reveal its contents. Several nodes will be visible, including **Database Link Settings**, **Database Settings**, and `dc=example,dc=com`.

Step 3. Click on the plus sign next to the `dc=example,dc=com` node to show its contents. One node named **userRoot** should be visible.

Step 4. Click on the **userRoot** node and then on the **Database Settings** tab on the right-hand side of the console window. The setting you're looking for is now visible as a check box labeled **Database is read-only**. By default, this is not checked; if it is checked, LDAP operations that change data are rejected by the server.

However, this section shows you how to change the setting without using the console. Follow these steps:

Step 1. Start a Unix shell or a Microsoft Windows command prompt window. On Solaris, type this command:

```
cd /export/ds6/shared/bin
```

On Microsoft Windows, type this command:

```
cd \Netscape\Servers\shared\bin
```

If you're using the Windows command prompt, omit the leading `./` sequences from the commands that follow.

The attribute that controls whether a database instance is writable is named `nsslapd-readonly`, and for the default user database instance it is located in the configuration entry named `cn=userRoot, cn=ldbm database, cn=plugins, cn=config` (`LDBM` stands for LDAP Database Manager and is the general name for Netscape's built-in LDAP data store). Entry updates are allowed by default, so the value of `nsslapd-readonly` is `off` for all database instances; you will change the value within the `userRoot` configuration entry to `on` in order to disable updates for that database instance.

Step 2. Execute the two `ldapmodify` commands shown in [Listing 4.13](#) to change the `nsslapd_READONLY` setting to `on` and to test whether updates were indeed disabled. For demonstration purposes, the first `ldapmodify` command modifies the `nsslapd_READONLY` configuration attribute while authenticated as the Directory Manager entry. That entry has full privileges within Netscape Directory Server; however, the `cn=config` subtree does support fine-grained access control using the same mechanism as the rest of the Netscape server.

Listing 4.13 Disabling Updates by Modifying a Configuration Entry

```
./ldapmodify -D "cn=Directory Manager" -w secret389

dn: cn=userRoot,cn=ldbm database,cn=plugins,cn=config
changetype: modify
replace: nsslapd_READONLY
nsslapd_READONLY: on
-
modifying entry cn=userRoot,cn=ldbm database,cn=plugins,cn=config
<Ctrl-C>

./ldapmodify -D "uid=kvaughan,ou=People,dc=example,dc=com" -w bribery

dn: uid=dmiller,ou=People,dc=example,dc=com
changetype: modify
replace: cn
cn: Dave Miller
cn: David Miller
-
modifying entry uid=dmiller,ou=People,dc=example,dc=com
ldap_modify: DSA is unwilling to perform
ldap_modify: additional info: database is read-only
```

The second `ldapmodify` command authenticates as Kirsten Vaughan (`kvaughan`) and attempts to modify the `cn` attribute values within David Miller's entry (`dmiller`). The `kvaughan` entry is part of a Directory Administrators group that has full access to the entries under the `ou=People,dc=example,dc=com` subtree. Kirsten's password in the `Example.ldif` data is "bribery."

Because the David Miller entry is in the `userRoot` database that has been configured to reject updates, the command fails with the error "DSA is unwilling to perform."

Step 3. Execute the command shown in [Listing 4.14](#) to restore the original configuration setting.

LDAP as a Server Administration Protocol

Exposing an extensive collection of server or application configuration information via LDAP is unusual, but this approach works well for Netscape Directory Server. LDAP is an open protocol that enables remote administration and allows a variety of configuration tools to be developed. Netscape Console communicates with the directory server via LDAP, as do many of Netscape's command-line utilities and scripts. The directory server can check the syntax and range of configuration values before accepting a change, and its powerful access control features can be used to regulate access to the configuration data. In addition, configuration changes take effect instantly; there is no need to restart the server or tell it to read a configuration file.

One potential disadvantage of using LDAP as a server administration protocol is that if intruders are able to get past the LDAP server's access control protection, they can reconfigure the server—but a similar risk exists with any method that supports remote administration.

Listing 4.14 Reenabling Updates by Modifying a Configuration Entry

```
./ldapmodify -D "cn=Directory Manager" -w secret389
dn: cn=userRoot,cn=ldbm database,cn=plugins,cn=config
changetype: modify
replace: nsslapd_READONLY
nsslapd_READONLY: off
-
modifying entry cn=userRoot,cn=ldbm database,cn=plugins,cn=config
<Ctrl-C>
```

Product Focus and Feature Set

Now that you have completed a brief hands-on tour of Netscape Directory Server, it is time to park the car for a while and skim the owner's manual. This section describes the origin, focus, and feature set of the Netscape server with an eye toward introducing you to some common characteristics of LDAP directory service products.

Origin

The first Netscape LDAP product, Netscape Directory Server 1.0, was delivered to the marketplace in September 1996. But Netscape did not start from scratch; it based its product on the open-source LDAP version 2 (LDAPv2) implementation from the University of Michigan. Version 1.0 of the Netscape product supported LDAPv2, server-to-server replication, a sophisticated access control scheme, a synchronization agent for Microsoft Windows NT domain directories, and an HTML template-based HTTP-to-LDAP gateway. Netscape quickly added LDAP support to the rest of its enterprise software, including products such as Netscape Communicator and Netscape SuiteSpot (a collection of integrated servers that included Netscape Enterprise Server, Netscape Messaging Server, and other servers). Netscape also developed software development kits (SDKs) and tools for LDAP developers.

Netscape shipped two major releases over the next three years, adding features such as LDAPv3 support, Netscape Console, and more robust replication features. In March 1999, Netscape and Sun Microsystems entered into the Sun-Netscape Alliance, in which the two companies jointly developed a variety of enterprise server products and delivered them under the iPlanet brand name. Later, Sun acquired Innosoft International, an open standards directory and messaging software company. The best ideas from Sun's own LDAP server (SunDS) and Innosoft's LDAP server (IDDS) were fed into the development of the product that was originally planned as Netscape Directory Server 5.0. The combined product shipped in May 2001 as iPlanet Directory Server 5.0, and it included noteworthy features such as multimaster replication, server-to-server chaining, entry distribution, and role-based access control.

As the Sun-Netscape Alliance was coming to its scheduled end, Netscape and Sun decided to go their separate ways. Sun continues to develop the iPlanet line of server products (now sold under the "Sun ONE" moniker), and Netscape again develops its own line of server products. In December 2001, Netscape shipped Netscape Directory Server 6.0, which includes support for DSML, as well as integration with some America Online services, such as AOL Instant Messenger (America Online is Netscape's parent company).

Product Focus

Netscape Directory Server is designed primarily to address the needs of large enterprises, e-commerce companies, and extranets. Netscape has historically been a performance and scalability leader. The Netscape product supports millions of LDAP entries per server and can process thousands of simple search operations per second. High performance for add and modify operations is not Netscape's focus; Netscape chooses to focus on search performance somewhat at the expense of update performance.

The Netscape server is designed to meet the needs of applications that work with a logically centralized directory service. Netscape has a broad line of LDAP-enabled products, including

- **Netscape Communicator** and **Netscape 7.0**. Web browser and e-mail client suites.

- **Netscape Certificate Management System.** A flexible, standards-based public key infrastructure (PKI) server suite that supports certificate issue, renewal, suspension, revocation, and online status checks.
- **Netscape Enterprise Server.** A high-performance, secure Web application server that is used by many high-traffic Web sites. Enterprise Server can use LDAP for authentication and access control.
- **AIM Enterprise Gateway.** An AOL Instant Messenger (AIM) gateway that allows AIM use to be managed by an enterprise. The AIM Enterprise Gateway acts as a proxy between users inside the corporate firewall and those on AOL's public AIM network, enabling enterprises to manage and control how employees use AIM services. LDAP-enabled identity management features map AIM screen names to corporate employee IDs and group employees by job function or department.
- **Netscape Delegated Administrator.** A product that builds on Netscape Directory Server to provide a Web-based interface for delegated directory and services administration as well as end-user self-administration. This product supports many levels of delegation of authority and many types of directory and service administrators (for example, e-mail administrators).

In addition, leading application and middleware vendors such as Netegrity, Hewlett-Packard, and IBM support Netscape Directory Server in their own LDAP-enabled products. Netscape Directory Server is a multiplatform product that runs on several leading server hardware and software platforms. Netscape also supports hardware-based accelerators to improve Secure Sockets Layer (SSL) and Transport Layer Security (TLS) performance.

The Netscape server is a flexible product that is easy to deploy and manage. It is therefore used in deployments that range from one location with a single server to large, multinational companies with thousands of LDAP servers. Netscape does not focus on the needs of network operating systems (where authentication, file services, and printing services still dominate the requirements). However, the Netscape product is popular with Unix operating system vendors. For example, Hewlett-Packard bundles the Netscape product with its HP/UX operating system to provide authentication, authorization, and centralized storage for the OS and its applications.

Feature Set

The Netscape product provides a broad set of features. Although some of the features are specific to this product, many of them are supported by other leading LDAPv3 servers. The Netscape feature set includes

- Support for LDAPv3 standards, including RFCs 2251, 2252, 2253, 2254, 2255, 2829, and 2830. This feature provides interoperability with LDAP clients and servers from other vendors, as well as strong, standards-based security.
- Support for many LDAPv3 controls, including those that provide Virtual List View, server-side sorting, persistent searching, proxied authorization, [ManageDSAIT](#), and password expiration. Some of these are proposed standards, and some are not; see [Chapter 3](#), LDAPv3 Extensions, for more information on LDAPv3 controls.
- Support for many LDAPv3 extended operations, including StartTLS and online bulk import. See [Chapter 3](#), LDAPv3 Extensions, for more information on LDAPv3 extended operations.
- Entry distribution to allow entries within one subtree of the DIT to be stored on more than one server. Distribution can be based on the location of an entry within a subtree, or a custom distribution algorithm can be specified; for example, a one-way hash of each entry's DN might be used.

- Extensible schema and configurable attribute indexes to support custom applications. New schema information can be added over LDAP. Netscape Directory Server 6 supports one subschema subentry per server; there is no support for using different schemas for different portions of the DIT.
- Server-to-server replication over LDAPv3. This feature provides flexible replication schedules, multiple writable copies (multiple masters), cascaded replication (chains of replicas), and replication monitoring.
- Flexible, scalable directory access control in which the rules are stored in the DIT so that they can be managed by means of LDAP and replicated with the regular directory data. Access control can be based on a DN, simple or dynamic (filter-based) groups, roles, time of day, day of the week, IP address, and other criteria. Fine-grained access control can be applied to entries, attributes, and attribute values.
- Support for several local directory databases on one server, or several remote databases located on network-connected servers (through server-to-server LDAP chaining). The databases support high-speed import from a file (LDIF or DSML format) and online bulk import over LDAP through a proprietary LDAPv3 extended operation.
- Data integrity features, including a transactional data store, referential integrity, attribute uniqueness enforcement, and the ability to restrict attribute values to ASCII (7-bit) characters.
- A server plug-in API that allows developers outside of Netscape to extend the functionality of the directory server in an arbitrary manner. For example, plug-in writers can implement preoperation filters, postoperation triggers, and new controls and extended operations. See the next section, Extending the Netscape Server: A Simple Plug-in Example, for more information on plug-ins.
- Integration with AOL's Instant Messenger (AIM) service to allow retrieval over LDAP of status information, such as whether someone is connected to AIM. This feature is unique to the Netscape server.
- A class of service (CoS) feature that provides collective or shared attribute values. Using CoS, you can manage an attribute value in one place that appears in thousands of directory entries. Although CoS is unique to the Netscape and Sun servers, some other directory servers also support shared attribute values.
- Dynamic groups and roles to provide convenient ways to manage collections of entries. Dynamic groups and roles may be used for access control, to control CoS attributes, and by LDAP applications.
- Support for SSL and TLS, including X.509v3 certificate-based authentication and the ability to algorithmically map information in a certificate to a specific directory entry. Netscape also supports hardware-based accelerators to improve SSL and TLS performance.
- Server monitoring using Simple Network Management Protocol (SNMP) and LDAP.
- Tools for C, C++, Java, JavaScript, and XML developers and for anyone using another language that can call C or Java code (such as Visual Basic). The Netscape SDKs support all popular OS and hardware platforms.

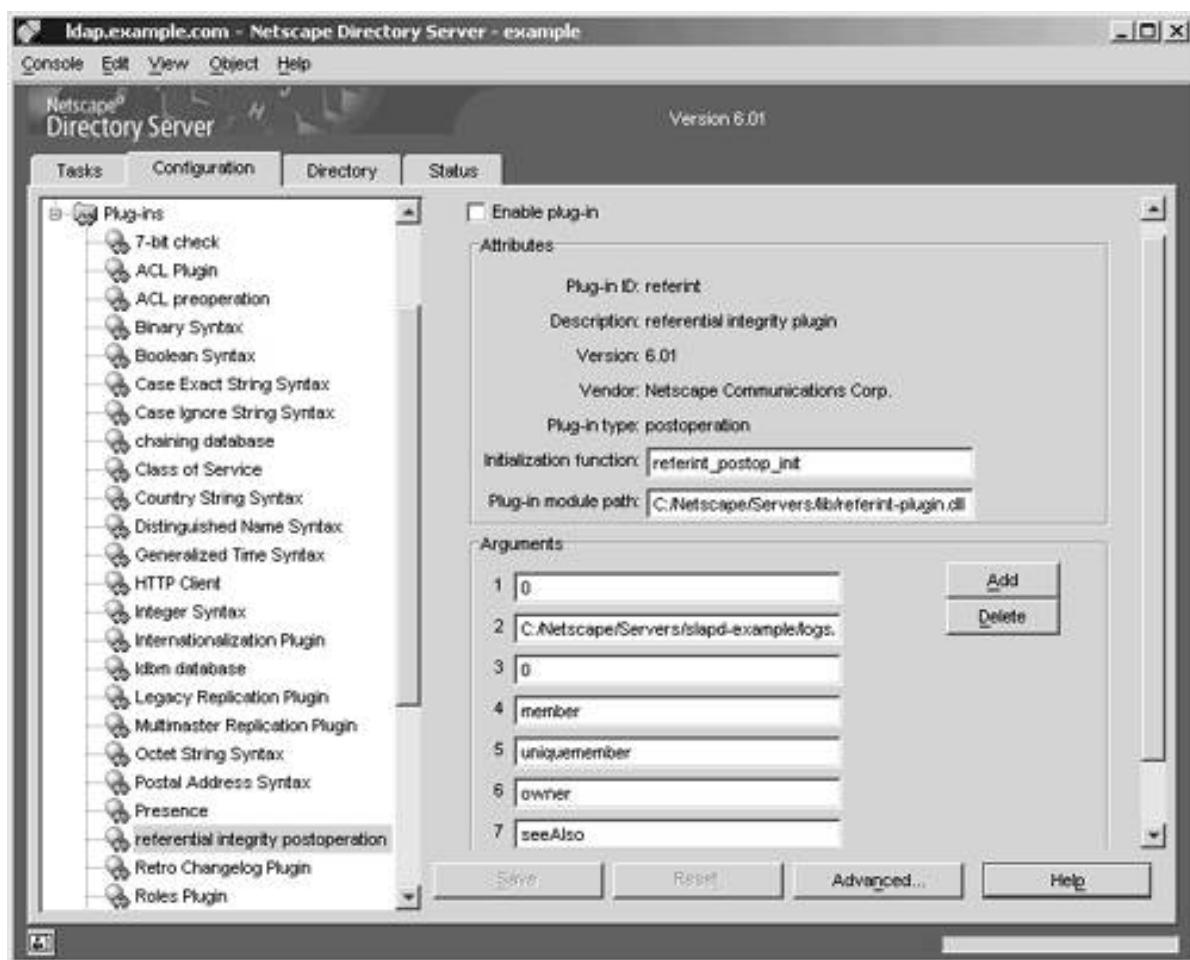
In summary, Netscape Directory Server 6 is a high-performance, highly scalable LDAP server that provides many features.

Extending the Netscape Server: A Simple Plug-in Example

Sometimes the cars available from the major automobile manufacturers simply do not meet all your needs. Or perhaps you just enjoy getting your hands dirty and cannot resist adding a turbocharger or reprogramming your engine computer. Some people need or want to make analogous enhancements to their server software. As mentioned in the preceding Feature Set section, Netscape Directory Server provides a plug-in API to allow developers outside of Netscape to extend the functionality of the server.

A plug-in is a Unix shared object or a Microsoft Windows DLL that presents a C interface to the directory server. Most plug-ins are written in C or C++. Plug-ins can be used to implement preoperation filters, postoperation triggers, new extended operations, new controls, and other extensions of the LDAP server functionality. Netscape uses plug-ins to provide some of the functionality of its server; therefore, many plug-ins are installed by default. [Figure 4.10](#) shows the plug-ins configuration window within Netscape Console, with Netscape's Referential Integrity plug-in selected.

Figure 4.10. The Netscape Console Plug-ins Configuration Window



Each plug-in is managed through a plug-in configuration entry that resides in a special area within the DIT. For example, the configuration for Netscape's Referential Integrity plug-in is stored in an entry named `cn=referential integrity postoperation, cn=plugins,cn=config`. [Listing 4.15](#) shows a sample configuration entry for the Referential Integrity plug-in (in LDIF format).

Listing 4.15 A Plug-in Configuration Entry

```
dn: cn=referential integrity postoperation,cn=plugins,cn=config
objectClass: top
objectClass: nsSlapdPlugin
```

```

objectClass: extensibleObject

cn: referential integrity postoperation

nsslapd-pluginPath: C:/Netscape/Servers/lib/referint-plugin.dll

nsslapd-pluginInitfunc: referint_postop_init

nsslapd-pluginType: postoperation

nsslapd-pluginEnabled: on

nsslapd-pluginarg0: 0

nsslapd-pluginarg1: C:/Netscape/Servers/slapy-example/logs/referint

nsslapd-pluginarg2: 0

nsslapd-pluginarg3: member

nsslapd-pluginarg4: uniquemember

nsslapd-pluginarg5: owner

nsslapd-pluginarg6: seeAlso

nsslapd-pluginarg7: nsroledn

nsslapd-plugindepends-on-type: database

nsslapd-pluginId: referint

nsslapd-pluginVersion: 6.01

nsslapd-pluginVendor: Netscape Communications Corp.

nsslapd-pluginDescription: referential integrity plugin

```

The remainder of this section demonstrates how to develop and use a custom plug-in. Complete information on writing plug-ins can be found in the *Netscape Directory Server Plug-in Programmers' Guide*.

Problem Statement

Netscape Directory Server does not provide a way to limit the values of an attribute to a predefined set. For example, the `inetOrgPerson` object class defined in RFC 2798 allows person entries to include an `employeeType` attribute. Typically only a few values are supposed to be used for this attribute; however, the set of valid values is specific to each directory deployment. Although LDAP clients can be written so that they limit the set of values that users and directory data administrators may enter, it would be nice if the directory server itself were able to reject invalid values. The custom plug-in presented in this part of the chapter is named the *Value Constraint plug-in*. It extends the Netscape server so that LDAP add and modify operations that include invalid `employeeType` values are rejected, with an appropriate error message sent back to the LDAP client.

The Design of the Value Constraint Plug-In

To restrict `employeeType` values to a predefined set, the plug-in must be invoked before an LDAP add or modify operation is processed by the server. Therefore, the Value Constraint plug-in is implemented as a preoperation plug-in. Preoperation plug-ins can be used to alter operation parameters, to reject an operation entirely, or just to make note of something and allow the Netscape server to process the operation as usual.

The valid set of `employeeType` values is assumed to be

- Contractor
- Employee
- Intern
- Temp

The plug-in will check the `employeeType` values sent in LDAP add and modify operations to ensure that they match one of these values. If not, a "constraint violation" error will be returned to the LDAP client, and the server will not process the operation. The "constraint violation" error is a standard LDAP error that indicates that an attribute value is too large, is in the wrong format, or otherwise does not meet constraints imposed by the LDAP information model or by a local policy.

Note

Netscape Directory Server pre- and postoperation plug-ins are executed only when the server is processing regular LDAP operations. They are not executed for special tasks such as `ldif2db` (which bulk-loads an LDIF file into a database instance). Therefore, the Value Constraint plug-in cannot enforce any constraints during data import; you need to use a different method to do so.

The Source Code for the Value Constraint Plug-In

The Value Constraint plug-in source code consists of three files: `valueconstraint.c`, `dllmain.c`, and `valueconstraint.def`. The latter two files are required on Microsoft Windows only; the `dllmain.c` code is not shown in this book because it is simply copied from the sample plug-in code that Netscape ships with its server. The `dllmain.c` file you need to copy can be found in this location (assuming you installed the server in the default location as recommended earlier in this chapter):

C:\Netscape\Servers\plugins\slapd\slapi\examples\dllmain.c

[Listing 4.16](#) shows the contents of the `valueconstraint.def` file, which is a Microsoft Windows DLL definition file.

Listing 4.16 The `valueconstraint.def` File

```
; Microsoft Windows DLL definition file for the valueconstraint
;
;      Netscape Directory Server 6 plugin.
;
; From the book "Understanding and Deploying LDAP Directory Services"
; by Timothy A. Howes, Mark C. Smith, and Gordon S. Good.
;
DESCRIPTION      'valueconstraint plugin for Netscape Directory Server'
EXPORTS
    valueconstraint_init      @1
```

The `valueconstraint_init()` function is the only function that is exported from the Value Constraint plug-in DLL. It is the main entry point to the Value Constraint plug-in, and the directory server will be configured to call this function during server startup.

The plug-in's interesting code is written in C and is housed in a file named `valueconstraint.c`. [Listing 4.17](#) shows the first portion of that file.

Listing 4.17 The Beginning of the `valueconstraint.c` File

```
1. /*
2.  * A valueconstraint plug-in for Netscape Directory Server
3. *
4.  * From the book "Understanding and Deploying LDAP Directory Services"
5.  * by Timothy A. Howes, Mark C. Smith, and Gordon S. Good
6. */
7. #include <string.h>
8. #include <stdio.h>
9. #include "slapi-plugin.h"
10.
11. /* Name and result code macros (to make the code easier to read) */
12. #define PLUGIN_NAME          "valueconstraint"
13. #define PLUGIN_RC_ERROR      (-1)
14. #define PLUGIN_RC_CONTINUE    0
15. #define PLUGIN_RC_RESULT_SENT 1
16.
17. /* Macros to take care of platform specifics */
18. #ifdef _WIN32
19. #define PLUGIN_STRCASECMP      stricmp
20. #define PLUGIN_EXPORTED_FUNCTION __declspec(dllexport)
21. #else
22. #define PLUGIN_STRCASECMP      strcasecmp
23. #define PLUGIN_EXPORTED_FUNCTION
24. #endif
25.
26. /* Static variables to hold the plug-in name and description */
27. static Slapi_PluginDesc pdesc = { PLUGIN_NAME,
28.                                 "Howes/Smith/Good", "2.0", PLUGIN_NAME " plugin" };
```

```

29.

30. /* Static variables to hold attribute constraint configuration */

31. static const char *attr_to_check = "employeeType";

32. static const char *valid_values[] = {

33.     "Contractor", "Employee", "Intern", "Temp", NULL

34. };

35.

36. /* Function prototypes */

37. static int valueconstraint_preadd( Slapi_PBlock *pb );

38. static int valueconstraint_premodify( Slapi_PBlock *pb );

39. static int valueconstraint_is_one_of( Slapi_PBlock *pb,

40.         const char *attrname, const char *val,

41.         const char *allowed[] );

42.

```

Lines 1 through 25 contain comments, `include` statements, and some simple macro definitions. The main header file for the Netscape Directory Server plug-in interface is called `slapi-plugin.h`; it is included on line 9. A plug-in description structure is defined on lines 26 through 28; later that structure is registered with the directory server so that descriptive information about the plug-in is available through Netscape Console and elsewhere.

In this example the attribute and the set of valid values allowed are hard-coded into the `valueconstraint.c` file (a good alternative would be to store this information in a directory entry such as the plug-in's own configuration entry). Lines 30 through 34 define an `attr_to_check` variable (`employeeType`) and a NULL-terminated `valid_values` array. Prototypes for the static functions that are internal to the plug-in implementation appear on lines 36 through 41.

[Listing 4.18](#) shows the `valueconstraint_init()` function that is called by the directory server during server startup. The purpose of this function is to perform any required initialization and to register plug-in callback functions with the directory server.

Listing 4.18 The `valueconstraint_init()` Function

```

43. /* Function valueconstraint_init(): initialization (called during
44. *   server startup).
45. */
46. PLUGIN_EXPORTED_FUNCTION int
47. valueconstraint_init( Slapi_PBlock *pb )
48. {
49.     int      rc = PLUGIN_RC_CONTINUE;
50.

```

```

51.     slapi_log_error( SLAPI_LOG_PLUGIN, PLUGIN_NAME, "=> init\n" );
52.
53.     /*
54.      * Register the plug-in version, plug-in description,
55.      * and two preoperation functions.
56.     */
57.
58.     if ( slapi_pblock_set( pb, SLAPI_PLUGIN_VERSION,
59.                           SLAPI_PLUGIN_VERSION_01 ) != 0 ||
60.         slapi_pblock_set( pb, SLAPI_PLUGIN_DESCRIPTION,
61.                           &pdesc ) != 0 ||
62.         slapi_pblock_set( pb, SLAPI_PLUGIN_PRE_ADD_FN,
63.                           (void *)valueconstraint_preadd ) != 0 ||
64.         slapi_pblock_set( pb, SLAPI_PLUGIN_PRE MODIFY_FN,
65.                           (void *)valueconstraint_premodify ) != 0 ) {
66.
67.         slapi_log_error( SLAPI_LOG_FATAL, PLUGIN_NAME,
68.                         "init: a slapi_pblock_set() call failed\n" );
69.
70.         rc = PLUGIN_RC_ERROR;
71.
72.     }
73. }
74.

```

The Netscape Directory Server plug-in API is sometimes referred to as the SLAPI API (which loosely stands for "standalone LDAP server API"), and most of the functions provided by Netscape have names that begin with "slapi". On line 51 in [Listing 4.18](#), a call is made to `slapi_log_error()`, which is a utility function that writes a message to the directory server's error log if the debug level provided in the first parameter (`SLAPI_LOG_PLUGIN`) is enabled in the server's configuration. By default, `SLAPI_LOG_PLUGIN` messages are not logged, but `SLAPI_LOG_FATAL` messages (used on line 65) are.

Information is passed back and forth between plug-ins and the core of the Netscape server through a parameter block data structure named `Slapi_PBlock` that can hold a variety of items. In fact, the only parameter passed to the `valueconstraint_init()` function is a pointer to a `Slapi_PBlock` data structure. The following two functions are used to get and set items within a SLAPI parameter block:

1. `slapi_pblock_get()`. Retrieves the value of an item from a `Slapi_PBlock` data structure. The item is identified by a macro whose name begins with `SLAPI_PLUGIN`, and the value of the item requested is returned via the last parameter, which is a pointer to an area to hold the value.

2. `slapi_pblock_set()`. Sets the value of an item in a `Slapi_PBlock` data structure. This function takes the same parameters as `slapi_pblock_get()`, except the last parameter is the value to set (not a pointer to the value).

The `slapi_pblock_get()` and `slapi_pblock_set()` functions are used often in the writing of SLAPI plug-ins. The code on lines 53 through 68 in [Listing 4.18](#) includes four calls to the `slapi_pblock_set()` function to register four things: the API version used by the plug-in (`SLAPI_PLUGIN_VERSION`), the plug-in description structure (`SLAPI_PLUGIN_DESCRIPTION`), and two callback functions that will be called before each add or modify operation is executed in the server (`SLAPI_PLUGIN_PRE_ADD_FN` and `SLAPI_PLUGIN_PRE MODIFY_FN`). A preoperation plug-in such as the Value Constraint plug-in can register one preoperation callback function for each kind of LDAP operation (bind, add, modify, delete, and so on). In this plug-in, there are only two such callback functions:

1. `valueconstraint_preadd()`. Called before the server executes each LDAP add operation but after the entire request has been received from the LDAP client and the request parameters have been parsed and placed in the `Slapi_PBlock` data structure. This function enforces the desired constraints on the `employeeType` attribute.
2. `valueconstraint_premodify()`. Called before the server executes each LDAP modify operation but after the entire request has been received from the LDAP client and the request parameters have been parsed and placed in the `Slapi_PBlock` data structure. This function also enforces constraints on the `employeeType` attribute.

Preoperation callback functions return an integer value that is examined by the Netscape server to determine if the operation should continue to execute or if processing should halt. In the latter case, the plug-in is responsible for sending an LDAP result message to the client. The two return values typically used are

- 0. The server should continue executing the operation.
- 1. The plug-in sends a result to the LDAP client, and the server should stop execution.

The macros defined on lines 14 (`PLUGIN_RC_CONTINUE`) and 15 (`PLUGIN_RC_RESULT_SENT`) of [Listing 4.17](#) capture these special return values.

[Listing 4.19](#) shows the code for the `valueconstraint_preadd()` function. It, too, is passed only one parameter, a `Slapi_PBlock` pointer. On lines 82 and 83, some additional Netscape-defined data types are used: `Slapi_Entry` (to represent a directory entry) and `Slapi_Attr` (to represent an attribute within an entry). Most of the data types defined by the Netscape SLAPI API, including these two and the `Slapi_PBlock` data structure introduced earlier, are *opaque* structures. This means that you can declare pointers to them, but you can't define them, determine their size, or look at their fields. Accessor functions are provided to manipulate these opaque data structures. This technique gives Netscape freedom to change the underlying implementation of its server without changing the SLAPI plug-in API.

Listing 4.19 The `valueconstraint_preadd()` Function

```

75. /* Function valueconstraint_preadd(): called by the directory
76. * server before an add operation is processed.
77. */
78. static int
79. valueconstraint_preadd( Slapi_PBlock *pb )
80. {

```

```

81.     int          rc = PLUGIN_RC_CONTINUE;
82.
83.     Slapi_Entry *entry = NULL;
84.
85.     Slapi_Attr  *attr = NULL;
86.
87.     /* Retrieve the entry that was sent in the add operation */
88.     if ( slapi_pblock_get( pb, SLAPI_ADD_ENTRY, &entry ) != 0 ) {
89.
90.         slapi_send_ldap_result( pb, LDAP_OPERATIONS_ERROR, NULL,
91.                               PLUGIN_NAME ": unable to retrieve entry", 0, NULL );
92.
93.         rc = PLUGIN_RC_RESULT_SENT;
94.
95.     } else if ( slapi_entry_attr_find( entry, attr_to_check,
96.                                         &attr ) == 0 ) {
97.
98.         /* Check the attribute values to make sure they are valid */
99.
100.        int           hint;
101.
102.        Slapi_Value  *value;
103.
104.        const char   *strvalue;
105.
106.
107.        for ( hint = slapi_attr_first_value( attr, &value );
108.              hint != -1;
109.              hint = slapi_attr_next_value( attr, hint, &value ) ) {
110.
111.            strvalue = slapi_value_get_string( value );
112.
113.            if ( !valueconstraint_is_one_of( pb, strvalue,
114.                                            attr_to_check, valid_values ) ) {
115.
116.                rc = PLUGIN_RC_RESULT_SENT;
117.
118.                break;
119.
120.            }
121.
122.        }
123.
124.    }
125.
126.    slapi_log_error( SLAPI_LOG_PLUGIN, PLUGIN_NAME,
127.                      "<= preadd (%d)\n", rc );
128.
129.    return rc;
130.
131. }

```

The `valueconstraint_preadd()` function first retrieves the entry that is being added (lines 87–91), and then it calls a `slapi_entry_attr_find()` `Slapi_Entry` accessor function to locate the attribute whose name is in the `attr_to_check` variable (which is `employeeType`). If that attribute is found, a `for` loop (lines 99–108) is used to examine each of the values and check that they are valid. Within the loop, the `slapi_attr_first_value()` and `slapi_attr_next_value()` functions are used to step through the values contained in the `Slapi_Attr` data structure, and the `slapi_value_get_string()` function is used to retrieve the string value. Finally, a utility function named `valueconstraint_is_one_of()` is used to determine if `strvalue` is in the set of valid values. If not, the function return code is set to `PLUGIN_RC_RESULT_SENT` (line 105). The `valueconstraint_is_one_of()` function sends an error result to the LDAP client if the value is not valid; the code for `valueconstraint_is_one_of()` is shown in [Listing 4.21](#).

[Listing 4.20](#) shows the code for the `valueconstraint_premodify()` function, which is quite similar to `valueconstraint_preadd()`. The code on lines 128 through 133 retrieves from the parameter block the list of modifications submitted by the LDAP client. The modifications are returned as a NULL-terminated array of pointers to `LDAPMod` structures. The `LDAPMod` structure is one of the few structures used in the Netscape plug-in API that is not opaque; it is actually part of the LDAP C API that is commonly used to write LDAP client applications.

Listing 4.20 The `valueconstraint_premodify()` Function

```

116. /* Function valueconstraint_premodify(): called by the directory
117. * server before a modify operation is processed.
118. */
119. static int
120. valueconstraint_premodify( Slapi_PBlock *pb )
121. {
122.     int          rc = PLUGIN_RC_CONTINUE;
123.     LDAPMod      *mp, **mods = NULL;
124.     int          i, j;
125.
126.     slapi_log_error( SLAPI_LOG_PLUGIN, PLUGIN_NAME, "=> premodify\n" );
127.
128.     /* Retrieve the modifications sent in the modify operation */
129.     if ( slapi_pblock_get( pb, SLAPI_MODIFY_MODS, &mods ) != 0 ) {
130.         slapi_send_ldap_result( pb, LDAP_OPERATIONS_ERROR, NULL,
131.                                 PLUGIN_NAME ": unable to retrieve modifications",
132.                                 0, NULL );
133.         rc = PLUGIN_RC_RESULT_SENT;
134.     } else {
135.         /*

```

```

136.         * Check the modify suboperations to make sure the values
137.         * they contain are valid. Delete suboperations are
138.         * ignored (any value may be deleted).
139.     */
140.     for ( i = 0; rc == PLUGIN_RC_CONTINUE && mods[i] != NULL;
141.             ++i ) {
142.         mp = mods[i];
143.         if ( 0 == slapi_attr_type_cmp( mp->mod_type,
144.                                         attr_to_check, SLAPI_TYPE_CMP_BASE )
145.             && LDAP_MOD_DELETE !=
146.                 ( mp->mod_op & ~LDAP_MOD_BVALUES ) ) {
147.             for ( j = 0; mp->mod_bvalues[j] != NULL; ++j ) {
148.                 if ( !valueconstraint_is_one_of( pb,
149.                     mp->mod_bvalues[j]->bv_val,
150.                     attr_to_check, valid_values ) ) {
151.                     rc = PLUGIN_RC_RESULT_SENT;
152.                     break;
153.                 }
154.             }
155.         }
156.     }
157. }
158.
159. slapi_log_error( SLAPI_LOG_PLUGIN, PLUGIN_NAME,
160.                   "<= premodify (%d)\n", rc );
161. return rc;
162. }
163.
```

The code on lines 135 through 156 loops through the modifications and checks each one to see if it is an "add value" or a "replace value" operation on the `employeeType` attribute (recall that `attr_to_check` was initialized to the constant string `"employeeType"`). The Netscape-provided `slapi_attr_type_cmp()` utility function is used to compare the attribute type name in the `LDAPMod` structures with `attr_to_check`. The `for` loop on lines 147 through 154 steps through the values, calling the `valueconstraint_is_one_of()` function for each one to check for invalid values.

[Listing 4.21](#) shows the code for the `valueconstraint_is_one_of()` function, which returns an indication

of whether a string value is in the array of allowed values. This function sends an error result to the LDAP client if an invalid value is detected.

Listing 4.21 The `valueconstraint_is_one_of()` Utility Function

```
164. /* Function valueconstraint_is_one_of(): return a nonzero value
165. * if "val" is in the NULL-terminated "allowed" array and 0 if not.
166. * Comparisons are case insensitive. If "val" is not present, a
167. * "constraint violation" error is sent to the LDAP client.
168. */
169. static int
170. valueconstraint_is_one_of( Slapi_PBlock *pb, const char *val,
171.                           const char *attrname, const char *allowed[] )
172. {
173.     int          i;
174.     char        *msg;
175.     const char  *fmt = "invalid value \"%s\" for %s";
176.
177.     if ( val == NULL ) {
178.         return 1; /* ignore NULL values */
179.     }
180.
181.     for ( i = 0; allowed[i] != NULL; ++i ) {
182.         if ( PLUGIN_STRCASECMP( val, allowed[i] ) == 0 ) {
183.             return 1; /* found it */
184.         }
185.     }
186.
187.     /* Not found: send back a "constraint violation" error */
188.     msg = slapi_ch_malloc( strlen(fmt) + strlen(val)
189.                           + strlen(attrname) + 1 );
190.     sprintf( msg, fmt, val, attrname );
191.     slapi_send_ldap_result( pb, LDAP_CONSTRAINT_VIOLATION,
192.                            NULL, msg, 0, NULL );
193.     slapi_ch_free_string( &msg );
194.     return 0; /* not found */
```

```
195. }
```

The `valueconstraint_is_one_of()` function is the last function in `valueconstraint.c`. The code on lines 181 through 185 looks for `val` (the value) within the `allowed` array using a case-insensitive comparison function (`strcasecmp()` or `stricmp()`), abstracted away by the `PLUGIN_STRCASECMP()` macro). If the value is in the set of allowed values, a nonzero value is returned by line 183.

The code on lines 187 through 194 constructs a human-readable error message and uses the `slapi_send_ldap_result()` Netscape plug-in API utility function to send that message along with an LDAP "constraint violation" result code to the client. The `LDAP_CONSTRAINT_VIOLATION` macro is defined by the `ldap.h` header file that is part of the LDAP C API. The `ldap.h` file is included from `slapi-plugin.h`.

Compiling and Installing the Value Constraint Plug-In

On Microsoft Windows, execute the following three commands at the command prompt to compile the two `.c` files that make up the plug-in (using the Microsoft Visual C++ `cl` command) and create a DLL named `valueconstraint.dll` (using the Visual C++ `link` command):

```
cl -IC:\Netscape\Servers\plugins\slapd\slapi\include /c valueconstraint.c

cl -IC:\Netscape\Servers\plugins\slapd\slapi\include /c dllmain.c

link /dll /def:valueconstraint.def /out:valueconstraint.dll /DEFAULTLIB:kernel32.lib
    ↵ user32.lib gdi32.lib winspool.lib comdlg32.lib advapi32.lib shell32.lib ole32.lib
    ↵ oleaut32.lib uuid.lib odbc32.lib odbccp32.lib wsock32.lib C:\Netscape\Servers\plugins\
    ↵ slapd\slapi\lib\libslapd.lib C:\Netscape\Servers\plugins\slapd\slapi\lib\libnspr4.lib
    ↵ valueconstraint.obj dllmain.obj
```

Yes, the `link` command is very long. On Solaris, use these two commands to compile the `valueconstraint.c` file and create a shared library named `valueconstraint.so`:

```
cc -I/export/ds6/plugins/slapd/slapi/include -D_REENTRANT -KPIC -c valueconstraint.c

ld -G -o valueconstraint.so valueconstraint.o /export/ds6/lib/libslapd.so /export/ds6/lib/
    ↵ libnspr4.so
```

The commands for other platforms are similar; consult the Netscape documentation for details.

To install the plug-in, shut down the directory server using the `stop-slapd` command. Next, edit the `dse.ldif` file located in the server `config` directory within the file system to add a configuration entry for the plug-in. [Listing 4.22](#) shows the correct entry for a Solaris installation, assuming the Value Constraint shared library is located in a directory named `/usr/netscape-server-plugins`.

Listing 4.22 The Value Constraint Plug-in Configuration Entry

```
dn: cn=valueconstraint,cn=plugins,cn=config
objectClass: top
objectClass: nsSlapdPlugin
objectClass: extensibleObject
cn: valueconstraint
nsslapd-pluginPath: /usr/netscape-server-plugins/valueconstraint.so
nsslapd-pluginInitfunc: valueconstraint_init
nsslapd-pluginType: preoperation
nsslapd-pluginEnabled: on
nsslapd-plugindepends-on-type: database
nsslapd-pluginId: valueconstraint
nsslapd-pluginVersion: 2.0
nsslapd-pluginVendor: Howes/Smith/Good
nsslapd-pluginDescription: valueconstraint plugin
```

For a Microsoft Windows installation, change `nsslapd-pluginPath` to point to your `valueconstraint.dll` file. After saving your changes to the `dse.ldif` file, restart the server. You can also add the plug-in configuration entry over LDAP while the directory server is running and then restart the server to load the plug-in.

Note

Adding new plug-ins always requires a restart of Netscape Directory Server. Through the use of Netscape Console or modification of the `nsslapd-pluginEnabled` attribute within a plug-in's configuration entry, plug-ins that were present when the server was last started can be disabled or enabled while the server is running. The value of `nsslapd-pluginEnabled` may be `off` or `on`.

Using Netscape Console to examine the list of active plug-ins, you can verify that your Value Constraint plug-in was recognized by the server and correctly loaded.

The Resulting Server Behavior

Verify that the plug-in is working correctly by trying some add and modify operations. Any LDAP client that supports those operations may be used.

[Listing 4.23](#) shows an LDIF file that contains two entries. The first includes an `employeeType` value of `unknown`, which should be rejected by the Value Constraint plug-in as invalid. The second entry includes a valid value (`contractor`).

Note

The commands shown in this section assume that the entries from Netscape's `Example.ldif` file have been loaded into your server. If in doubt, execute a command like this to load the `Example.ldif` file:

```
./ldif2db -n userRoot -i - <ldif/Example.ldif
```

The `ldif2db` command must be executed from the `\Netscape\Servers\ slapd-example` directory on Windows or from the `/export/ds6/slapd-example` directory on Solaris. The other commands in this section rely on the presence of some of the entries and `aci` attributes from the `Example.ldif` file.

Listing 4.23 Entries with Different `employeeType` Values

```
version: 1

# this add operation should fail with constraintViolation

dn: uid=cjones,ou=People,dc=example,dc=com

cn: Christina Jones

sn: Jones

givenName: Christina

objectClass: top

objectClass: person

objectClass: organizationalPerson

objectClass: inetOrgPerson

ou: Accounting

ou: People

L: Sunnyvale

uid: cjones

mail: cjones@example.com

userPassword: secret

employeeType: unknown

# this add operation should succeed

dn: uid=cjones,ou=People,dc=example,dc=com

cn: Christina Jones

sn: Jones

givenName: Christina
```

```
objectClass: top  
objectClass: person  
objectClass: organizationalPerson  
objectClass: inetOrgPerson  
  
ou: Accounting  
  
ou: People  
  
L: Sunnyvale  
  
uid: cjones  
  
mail: cjones@example.com  
  
userPassword: secret  
  
employeeType: contractor
```

Place the contents of [Listing 4.23](#) in a file named `testadds.ldif` and use the `ldapmodify` command to test the Value Constraint plug-in's behavior when processing LDAP add operations. [Listing 4.24](#) shows the command and the result. The `-a` option (add entries) and the `-c` option (continue even if an error occurs) are passed to `ldapmodify` so that the command will try to add the second entry even if the first add fails. The result indicates that the test was a success; the plug-in returned a "constraint violation" error.

Listing 4.24 Testing the Value Constraint Plug-in with Add Operations

```
./ldapmodify -ac -D "uid=kvaughan,ou=People,dc=example,dc=com" -w bribery < testadds.ldif  
  
adding new entry uid=cjones,ou=People,dc=example,dc=com  
ldap_add: Constraint violation  
ldap_add: additional info: invalid value "unknown" for employeeType  
  
adding new entry uid=cjones,ou=People,dc=example,dc=com
```

Next, test the behavior of the Value Constraint plug-in when it is confronted with LDAP modify operations. [Listing 4.25](#) shows an LDIF file that contains a series of modify operations. Comments are included in the file to indicate whether the operation should succeed or not.

Listing 4.25 Modify Operations that Use Different `employeeType` Values

```
version: 1  
  
# this modify operation should fail with constraintViolation  
  
dn: uid=cjones,ou=People,dc=example,dc=com  
changeType: modify  
  
replace: employeeType
```

```

employeeType: fulltime
-
# this modify operation should succeed
dn: uid=cjones,ou=People,dc=example,dc=com
changeType: modify
replace: employeeType
employeeType: employee
-
# this modify operation should succeed
dn: uid=cjones,ou=People,dc=example,dc=com
changeType: modify
delete: employeeType
employeeType: employee
-
# this modify operation should fail with noSuchAttributeValue
dn: uid=cjones,ou=People,dc=example,dc=com
changeType: modify
delete: employeeType
employeeType: vice president
-
# this modify operation should fail with constraintViolation
dn: uid=cjones,ou=People,dc=example,dc=com
changeType: modify
add: employeeType
employeeType: employee
employeeType: vice president
-
```

Use the `ldapmodify` command one more time to verify that modify operations containing invalid `employeeType` values are rejected by the plug-in (and therefore by the directory server) and that those containing valid values are allowed. [Listing 4.26](#) shows the command and the result. This time, the `-c` (continue if an error occurs) and the `-v` (verbose) options are used.

Listing 4.26 Testing the Value Constraint Plug-in with Modify Operations

```
./ldapmodify -c -v -D "uid=kvaughan,ou=People,dc=example,dc=com" -w bribery <  
➥ testmodifies.ldif  
  
ldapmodify: started Tue Aug  2 22:02:54 2002  
  
ldap_init( localhost, 3389 )  
  
Processing a version 1 LDIF file...  
  
replace employeeType:  
  
      fulltime  
  
modifying entry uid=cjones,ou=People,dc=example,dc=com  
  
ldap_modify: Constraint violation  
  
ldap_modify: additional info: invalid value "fulltime" for employeeType  
  
  
replace employeeType:  
  
      employee  
  
modifying entry uid=cjones,ou=People,dc=example,dc=com  
  
modify complete  
  
  
delete employeeType:  
  
      employee  
  
modifying entry uid=cjones,ou=People,dc=example,dc=com  
  
modify complete  
  
  
delete employeeType:  
  
      vice president  
  
modifying entry uid=cjones,ou=People,dc=example,dc=com  
  
ldap_modify: No such attribute  
  
  
add employeeType:  
  
      employee  
  
      vice president  
  
modifying entry uid=cjones,ou=People,dc=example,dc=com  
  
ldap_modify: Constraint violation
```

```
ldap_modify: additional info: invalid value "vice president" for employeeType
```

The plug-in worked as expected. Now perform one more quick test using a different LDAP client. Start Netscape Console and use the **Directory** tab to navigate to the **People** container. The `employeeType` attribute is not listed on any of Netscape's built-in tabs within the entry editor. Follow the steps described here to try to add an invalid `employeeType` value to an entry:

Step 1. Open an **Edit Entry** window for the person entry named `awhite` (Alan White) by double-clicking on the entry name in the right-hand pane of the **Directory** window. Alan White's entry is the first one listed in the **People** container.

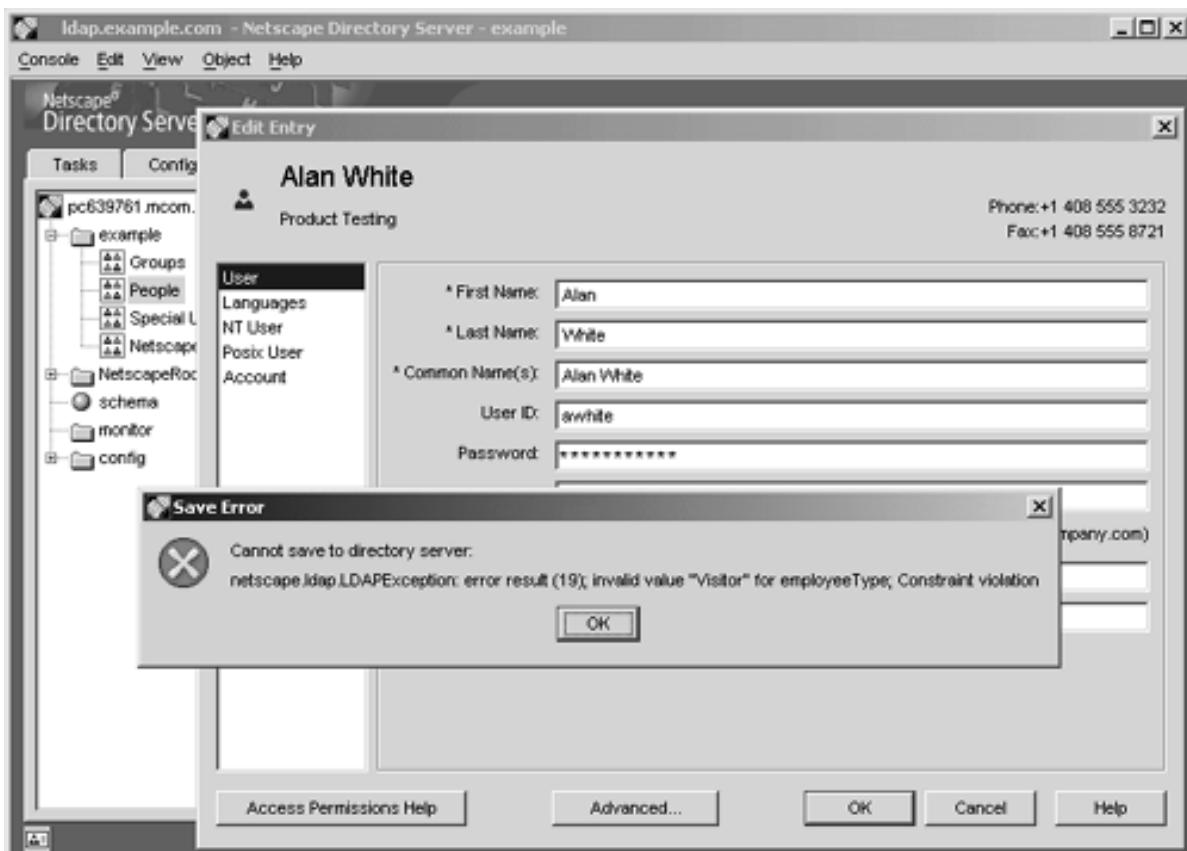
Step 2. Click the **Advanced...** button to open a **Property Editor** window that provides access to all of the available attributes and object classes.

Step 3. Click the **Add Attribute...** button and select `employeeType` from the list that appears. An empty attribute labeled **Employee category** is added to the list of attributes in the **Property Editor** window. This is the descriptive name used by Netscape Console for the `employeeType` attribute.

Step 4. Type "Visitor" in the **Employee category** field, click the **OK** button to close the **Property Editor** window, and click the **OK** button on the **Edit Entry** window.

Netscape Console will try to write your changes to the directory server. [Figure 4.11](#) shows the resulting error alert.

Figure 4.11. A Failed Attempt to Add an Invalid `employeeType` Value



Although this was not an exhaustive test, you should now be convinced that the Value Constraint plug-in is working as designed. The ability to extend Netscape Directory Server using plug-ins is one of its strengths.

Ideas for Improvement

The Value Constraint plug-in could be enhanced in many ways. Here are a few ideas:

- Remove the hard-coded attribute name and the list of valid values from the `valueconstraint.c` file, and change the plug-in to read the necessary configuration information from the Value Constraint plug-in's own entry.
- Support regular expressions or a similar, flexible pattern-matching scheme when checking for valid attribute values.
- Modify the plug-in so that it enforces constraints for only those entries that contain a specific object class value. Although `employeeType` attributes typically appear only in person entries, you may want to enforce constraints on an attribute such as a person's `cn` (common name). Because the `cn` attribute is used in many other kinds of entries, in this case the Value Constraint plug-in needs to be intelligent enough to ignore add and modify operations for nonperson entries.

Your customized directory server is better able to meet your needs. As with a car that you own for many years, with experience you can make specific directory products perform in ways that better match your own needs. Now park the car and shut off the ignition:

```
./stop-slapd
```

This driving lesson is finished.

Team LiB

◀ PREVIOUS

NEXT ▶

Further Reading

Definition of the inetOrgPerson LDAP Object Class (RFC 2798). M. Smith, 2000. Available on the Web at <http://www.ietf.org/rfc/rfc2798.txt>.

Netscape Directory Server Administrator's Guide. Available on the Web at <http://enterprise.netscape.com/docs/directory>.

Netscape Directory Server Configuration, Command, and File Reference. Available on the Web at <http://enterprise.netscape.com/docs/directory>.

Netscape Directory Server Deployment Guide. Available on the Web at <http://enterprise.netscape.com/docs/directory>.

Netscape Directory Server Installation Guide. Available on the Web at <http://enterprise.netscape.com/docs/directory>.

Netscape Directory Server Plug-in Programmer's Guide. Available on the Web at <http://enterprise.netscape.com/docs/directory>.

Netscape Directory Server Schema Reference. Available on the Web at <http://enterprise.netscape.com/docs/directory>.

Looking Ahead

[Part I](#), Introduction to Directory Services and LDAP, has laid the groundwork for the rest of this book by providing an overview of directory services, a comprehensive introduction to the LDAP protocol, and a hands-on look at a leading directory server product. [Part II](#), Designing Your Directory Service, will describe directory design from the ground up. [Chapter 5](#), Directory Design Road Map, will provide an overview of the directory design process, and each of the remaining chapters of [Part II](#) will cover a major directory design topic in detail.

Part II: Designing Your Directory Service

- [5. Directory Design Road Map](#)
- [6. Defining Your Directory Needs](#)
- [7. Data Design](#)
- [8. Schema Design](#)
- [9. Namespace Design](#)
- [10. Topology Design](#)
- [11. Replication Design](#)
- [12. Privacy and Security Design](#)

Chapter 5. Directory Design Road Map

- The Directory Life Cycle
- Directory Design Checklist
- Further Reading
- Looking Ahead

This chapter introduces the three phases of the directory life cycle, briefly describing the role of each. The chapters in the remainder of [Part II](#), Designing Your Directory Service, focus on the first phase of the life cycle—design—and provide a road map to each aspect of this task. Subsequent parts focus on the other two phases of the life cycle.

The Directory Life Cycle

The life cycle of your directory can be broken down into three general phases: design, deployment, and maintenance. It will come as no surprise that the life cycle is not actually this simple, nor is it so easily segmented. In practice, you may design for a while (until you think you've got it right) and then during the deployment phase discover problems with your design that send you back to the design phase. New features introduced at any phase can necessitate a new, miniature design-deploy-maintain cycle. Requirements can also change, even in the maintenance phase, making it necessary to redesign of parts of the service.

View the life cycle of your directory, then, as a series of design, deployment, and maintenance tasks for the original service itself, for new applications that use the directory, and for new features of your directory service. The tasks may appear distinct, or they may blend together.

Nevertheless, it is still helpful to think of the three phases as being separate, especially for discussion purposes. Each phase has distinct tasks:

1. **Design phase.** During the design phase you will gather data about your environment and data sources, your users, and your directory-enabled applications. Using this data, you will design a service that fits your needs.
2. **Deployment phase.** During the deployment phase you will pick a directory vendor; pilot your service in a test configuration; and do stress, scale, and performance testing. You will also test the reliability, redundancy, and fault tolerance capabilities of the system. This phase is also where you first expose the system to a relatively small group of users to get feedback. After the pilot is complete, you will put your directory service into production.
3. **Maintenance phase.** During the maintenance phase you will continually update the data in your directory, keep the service running smoothly, accommodate new applications, and continue to improve the service.

As your service grows and matures, you may often find yourself in all three of these phases at once. For example, you may be designing a new aspect of the service, deploying the latest addition to the service, and maintaining the current service all at the same time.

The following sections delve into each phase of the life cycle in more detail. Design is first, followed by deployment and maintenance. Our goal in this chapter is to give an overview, or road map, of each topic, but not a complete treatment. Subsequent chapters cover each topic in more detail.

Design

During the design phase of your directory service, the most important task is to understand the requirements that will be placed on your directory. Armed with a clear understanding of these requirements, the rest of your design task will be much easier. The directory will not exactly design itself, of course, but your decisions will be relatively uncomplicated if you fully understand your requirements. The result of the design phase is most likely a document or set of documents describing your directory service, the applications it will serve, the major design decisions made during the design process, and the trade-offs those decisions represent.

There are many ways to segment your design process. We've chosen to break things up in the manner described in this chapter. In our experience, this approach is good because it helps you organize and divide your design decisions. Breaking down the task in this manner also helps provide a framework for discussion. Keep in mind, however, that other methodologies may be equally valid. Feel free to explore different approaches, but make sure that you cover all the bases mentioned here. You will find in practice that design decisions you make in one area affect the design of other areas as well.

The design phase is aimed at designing a directory to suit your environment in the following areas:

- **Directory needs.** Your directory service will need to meet the needs of the directory-enabled applications you plan to deploy and the needs of the people who will use those applications. Understanding your applications' needs is covered in [Chapter 6](#), Defining Your Directory Needs.
- **Data.** Whatever type of directory service you want to create, you can be assured that it will involve some kind of data. Chances are good that you have in your organization many existing sources of data that may prove useful in creating your service. Cataloging these data sources, identifying their owners, and establishing an ongoing relationship to your directory is important. Designing specifically for your organization's data is covered in [Chapter 7](#), Data Design.
- **Schema.** Your directory exists to support one or more directory-enabled applications. These applications have certain requirements concerning the data contained in the directory, its format, and how the data is interpreted. These characteristics are determined by the directory schemas, as discussed in [Chapter 8](#), Schema Design.
- **Namespace.** After you determine what data you want to put in your directory, you need a way to organize and reference that data. This is the purpose of a directory namespace. As you will see in [Chapter 9](#), Namespace Design, the choices you make here can have wide-ranging implications for other aspects of the service.
- **Topology.** Topology design involves determining the number of servers you will need, how your directory data will be split among those servers, and where those servers need to be located. The directory topology that determines these factors is discussed in [Chapter 10](#), Topology Design.
- **Replication.** Directory-enabled applications often place severe performance and reliability requirements on the directory. Replication is the means by which the same directory data is maintained in multiple directory servers, leading to a more reliable service and more servers that can answer application queries—thus increasing performance. This subject is the topic of [Chapter 11](#), Replication Design.

Note

Namespace design, topology design, and replication design are interconnected topics. The decisions you make in one area will constrain your options in the other two areas. Completing the design process for these three areas will involve an iterative approach in which you make assumptions in one area, see how they work out in the other two, and make adjustments until you have arrived at an optimal solution.

- **Security.** The security aspect of your design cuts across all other design areas. You must plan to protect the data in the directory itself, as well as design the other aspects of the service to meet the security and privacy requirements of your users and their applications. Many directory design decisions have these security implications. This topic is covered in [Chapter 12](#), Privacy and Security Design.

After you've answered the questions posed by these design areas, you will have the beginnings of a complete directory service. The next phase in the development of your directory service is to deploy the service you've designed.

Deployment

The directory deployment phase is where things start to get real. Until now, you've been collecting data and creating paper designs. In this phase of the project, you actually try things out. You'll work the kinks out of your design, select appropriate directory software, and perform tests to ensure that your design can handle the anticipated load. Finally, you'll roll the directory service into production.

Although the deployment phase varies somewhat depending on your design and your user community, there are certain things you should be sure to do. The following list describes some tips for picking the right directory software and the importance of piloting your directory, doing performance and scale testing, and getting user feedback during the deployment phase:

- **Choosing directory software.** Choosing the proper directory software is important. If you choose software that cannot satisfy your design requirements or cannot scale or perform up to your requirements, it's back to the design drawing board—or time to choose some new directory software. Either way, you are in for a costly, unpleasant experience that will delay final deployment of your service.

Evaluating and choosing directory products can be complicated. You may have requirements not reflected in your ideal design, such as an organizational preference for a particular vendor. Directory products that are technically acceptable might be disqualified for other, nontechnical reasons, such as cost.

Try to consider all factors—technical and practical—that affect your choice of vendor. Don't choose a vendor without some real hands-on experience with its software. This topic is discussed in more detail in [Chapter 13](#), Evaluating Directory Products.

- **Piloting.** The purpose of the deployment phase is to validate your design decisions (or prove them invalid, which calls for a redesign) and determine whether your directory service is functioning properly. The best way to do this is by piloting your directory service.

A directory *pilot* requires you to set up, usually on a smaller scale, the directory you have designed and expose it to a limited user community to get feedback. Piloting your directory service can tell you whether the software you have chosen is appropriate. Piloting can also tell you whether the decisions you've made concerning schema, namespace, topology, and replication fulfill the needs of your directory-enabled applications, and whether the data-gathering techniques you've designed are feasible. Finally, in the pilot phase you will test your directory's performance and scaling characteristics to understand better how you will scale the service to meet future needs. More information on piloting your directory service can be found in [Chapter 14](#), Piloting Your Directory Service.

- **Analyzing costs.** It is also crucial to look at the costs of your directory service from all angles; you need to analyze not only implementation costs, but also maintenance and other costs. This topic is discussed in [Chapter 15](#), Analyzing and Reducing Costs.
- **Obtaining user feedback.** Whatever type of service you're deploying, it's important to expose it to some users. They are, after all, the reason you're doing all this work in the first place. The service may look great to you but still not meet the needs of your users. Getting user feedback is one of the most important steps in validating your design. It is imperative to get feedback from a cross section of users representing a wide variety of requirements. Without it, you won't know whether you've done a good job meeting your objectives with the service.
- **Moving to production.** Finally, during the deployment phase you need to develop and execute a plan for taking the directory service from a pilot to production. Think about how you can do this smoothly and incrementally, if possible. Moving into production is discussed in [Chapter 16](#), Putting Your Directory Service into Production.

A more complete treatment of directory deployment is covered in [Part III](#), Deploying Your Directory Service.

Maintenance

The maintenance of your directory service is the longest-lasting phase, but this important aspect of the service often receives the least amount of attention during the design phase. Designing up front for an easy-to-maintain directory can really pay off in lower maintenance costs and happier users and administrators. Not thinking about maintenance until your directory is deployed can result in a fragile system strung together with various kludges. This kind of system leads to higher administrative overhead, higher costs, and, ultimately, unhappy users.

For example, consider a directory service design that does not take into account data maintenance. Other aspects of the service may be designed well, and users may be happy with the service initially. But what happens in the weeks and months that follow when the data contained in the directory becomes out-of-date? The service becomes less useful, user satisfaction plummets, calls to the Help Desk skyrocket, and the service designer goes from hero to zero quickly.

When designing for a maintainable directory service, think about the everyday or other periodic tasks you will need to perform. How will users be added to the directory? How will data obtained from other sources be kept up-to-date? How often will entries change names, and how painful a process will this be? It's OK to recognize as exceptions some tasks that don't happen often and therefore don't need to be made all that convenient. But keep in mind that your definition of exceptions may change as the number of users in your directory changes. For example, an annoying administrative task that needs to be done once a month on average for every 1,000 users may be perfectly acceptable for a 1,000-user directory. But if your directory grows to 100,000 users, you'll be doing this task more than three times a day!

Consider the following aspects of directory maintenance:

- **Data backups and disaster recovery.** An important aspect of maintaining a reliable service is performing regular backups of your directory data. It's also vitally important to verify your backups to be sure that the data can be restored in the event of a disaster. Finally, if your directory service falls victim to a disaster, having a complete disaster recovery plan is crucial. We discuss these topics in [Chapter 17](#).

- **Data maintenance.** Maintaining the data is often the most important task involved in maintaining your directory. If the data in your directory is not up-to-date, the whole service is much less useful. Make sure that you have in place maintainable and scalable procedures for keeping your directory data updated. The topic of data maintenance is addressed in [Chapter 18](#), Maintaining Data.
- **Monitoring.** Naturally, you'll want to monitor your directory service. You need to know when it goes down so that you can bring it back up quickly. You also need to ensure that it's performing well for your users and then tune it if it is not. You may have an existing monitoring infrastructure in your organization that you want to plug into. The topic of monitoring your directory service is covered in [Chapter 19](#), Monitoring.
- **Troubleshooting.** Even the best-designed directory service occasionally has problems. Effective troubleshooting of problems is critical to your directory service's success. In [Chapter 20](#), Troubleshooting, we discuss strategies for identifying and remedying problems.
- **Changing requirements.** Part of your maintenance plan should include contingency plans for handling changing requirements beyond the normal expansion already described. Think about what you would do if any of your fundamental design assumptions were to change, or if additional directory-enabled applications needed to be deployed. Any change in requirements might require that you kick off a small design cycle to accommodate the new requirements, revisiting your data, schema, namespace, topology, replication, and security design.

Directory Design Checklist

To review, here are the major tasks involved in designing your directory service:

- Determine your directory needs and what applications will use the directory.
- Determine your data needs.
- Design your schema.
- Design your namespace.
- Design the topology of your directory.
- Design your directory replication scheme.
- Design your directory for security and privacy.

Further Reading

Building an X.500 Directory Service in the US (RFC 1943). B. Jennings, 1996. Available on the World Wide Web at <http://www.ietf.org/rfc/rfc1943.txt>.

Introduction to White Pages Services Based on X.500 (RFC 1684). P. Jurg, 1994. Available on the World Wide Web at <http://www.ietf.org/rfc/rfc1684.txt>.

Netscape Directory Server Deployment Guide. Netscape Communications Corporation, 2001. Available on the World Wide Web at <http://enterprise.netscape.com/docs/directory>.

Windows 2000 Active Directory Design and Deployment. G. Olsen and T. Carlson, New Riders Publishing, 2000.

Check with your directory vendor for any product-specific documentation available about directory design, deployment, and maintenance.

Looking Ahead

Now that you've studied the road map presented in this chapter and mapped out your route, it's time to get out there on the road! The rest of [Part II](#), [Chapters 6](#) through [12](#), will take you through a detailed design process for each of the design areas mentioned in this chapter. How to identify your directory needs will be tackled first. [Parts III](#) and [IV](#) will cover the topics of directory deployment and maintenance, respectively.

Chapter 6. Defining Your Directory Needs

- Overview of the Directory Needs Definition Process
- Analyzing Your Environment
- Determining and Prioritizing Application Needs
- Determining and Prioritizing Users' Needs and Expectations
- Determining and Prioritizing Deployment Constraints
- Determining and Prioritizing Other Environmental Constraints
- Choosing an Overall Directory Design and Deployment Approach
- Setting Some Goals and Milestones
- Defining Your Directory Needs Checklist
- Further Reading
- Looking Ahead

Directory needs encompass all the reasons for designing and deploying a directory service. Before beginning work on any task, it is a good idea to understand why you're working on it and what you hope to accomplish. Ask yourself why you decided to deploy a directory service. These are some possible answers:

- To support a few important directory-enabled applications
- To save money by consolidating a multitude of proprietary directories into one standards-based service—that is, to solve the N+1 directory problem described in [Chapter 1](#), Directory Services Overview and History
- To help system administrators manage information with less effort and greater automation
- To make it easier for end users to locate resources within your organization

No matter what the initial impetus, your directory service ultimately must serve the needs of applications and people, and it should be designed with those needs in mind. Your design will likely be evaluated on the basis of how well it serves these needs; therefore, it is important to understand the needs up front.

It is also important to choose a directory design and deployment approach that fits your situation and your philosophy. A variety of organizational and environmental constraints, along with the preferences of the design and deployment team itself, determine what approach works best. A successful approach also balances short-term needs with the need to lay a good foundation for future use of the directory service.

This chapter helps you gather and organize information on all the different needs and constraints that affect the design of your directory service. After working through the material in this chapter, you should be able to produce a good first draft of a directory requirements document. You will also be able to begin to form a project plan for your directory design and deployment effort.

Experienced project managers will find much of the general material presented in this chapter to be familiar. We begin with an overview of the needs definition process, and then we explore each part of this process in detail in the rest of the chapter.

Overview of the Directory Needs Definition Process

Your boss, coworkers, and customers (users) will measure the success of your directory service on the basis of how well it meets their needs and the needs of the entire organization. Whether this evaluation is formal (for example, part of a yearly performance review) or informal (for example, a casual conversation that takes place near the cappuccino machine), it *will* take place. The best way to meet everyone's directory-related needs is to gather and understand as many of them as possible up front, and to keep them in mind throughout the stages of directory design and deployment.

Step 1: Analyze Your Environment

The first step in gathering your directory-related needs is to understand the environment in which your directory service will be deployed. *Environment* is a broad term that covers a wide range of topics, including organizational structure and geography, computer systems, networks, application software, users, the directory deployment team, other system administrators, and any other people the directory serves. Later in this chapter you will learn how to analyze your own environment.

Step 2: Determine and Prioritize Needs

When you have a good understanding of the overall environment, the needs and constraints that come from each area should be gathered and prioritized. The order in which you do this doesn't matter, as long as no stone is left unturned. Later in this chapter you will learn how to accomplish this task. In brief, you will want to look at each of these areas:

- **Application needs.** Application needs include all the things the directory service must do to help directory-enabled applications perform their own tasks correctly and efficiently. Applications are usually the primary force behind deployment of a directory service. For example, a modern messaging system typically relies on a directory service for its knowledge of e-mail users and groups of users. It may be impossible to deploy the messaging system without first deploying a directory service that holds the required information. Gathering information about the needs of applications can be complex and time-consuming, but it is probably the most important task and, therefore, the one you should spend the most time on.
- **User needs and expectations.** User needs and expectations include all the things that people who use your directory service expect or desire from it. For example, end users may expect to always find accurate and up-to-date telephone numbers in the directory. Or they may have privacy concerns about the personal data held in the directory service. Your directory service design should consider all the users' desires, although it is often difficult to know what your end users really need (especially if the concept of a directory service is new to them). One job of a good directory project planner is to listen to what users say they want but deliver what they actually need.
- **Deployment constraints.** Deployment constraints arise from the organizational situation or the characteristics of the people charged with designing and deploying the directory service. Resource constraints, personal and organizational philosophies, the needs of those who will administer the directory service, and other realities affect your directory design and largely determine your overall approach. Because you are probably a member of the deployment team yourself, you should not have to go far to gather these deployment constraints, but you need to be careful to examine your situation objectively.

- **Other environmental constraints.** Other constraints may arise from the environment in which the directory service is deployed. These include everything from system- and network-related constraints to any constraints imposed by the other data sources and directories with which your directory must coexist. For example, the computers and operating systems already being used in your organization may limit your choice of directory service software. Also security needs vary depending on the overall audience for your service; if your directory service lives inside a firewall, it probably needs less protection than if it is accessible to everyone on the Internet.

For each of these areas, you must gather information, produce a list of needs, and assign priorities to each item on the list.

Step 3: Choose an Overall Directory Design and Deployment Approach

After you have gathered and prioritized all the different directory-related needs and constraints, it is important to choose an overall approach to directory design and deployment that fits your needs and situation. In this book we guide you through a design and deployment process that has served us well, but your personal philosophy and organizational realities may lead you to adopt a different approach. The amounts of time spent in the design stage, the piloting stage, and in deployment of the initial directory service all vary widely from organization to organization and from project to project. The most important thing is to choose an approach that will help you succeed with your directory service deployment. Later in this chapter you will learn how to make a good choice.

Step 4: Set Goals and Milestones

The fourth and final step in the needs definition process is to set some goals and milestones that will measure the progress of your directory deployment. This seemingly simple task is often overlooked, but it is well worth spending time up front to define goals and milestones. Most projects go more smoothly when people working on them have good targets to aim for. Good goals and milestones are easily understood, realistic, and significant enough to be worth celebrating. Later in this chapter you will learn how to set good goals and milestones for your directory project.

The results of the directory needs definition process feed into the rest of the directory design process discussed in subsequent chapters. The remainder of this chapter is devoted to exploring each of the needs definition tasks in more detail.

Analyzing Your Environment

To determine your directory needs accurately, it is essential to understand the environment in which your directory service will operate. Unless you are new to your organization, you probably already know quite a bit about the overall environment. Spending some time during the early stages of design to record what you do know, and additional time on research to fill in any missing details, will pay off later in the directory design process. Explore the four areas described in this section—organizational structure and geography, computer systems, the network, and application software—to help you produce a complete portrait of the environment in which your directory service will be deployed.

Organizational Structure and Geography

Create a list of the major units within your organization and all the physical locations your directory service must serve. Also include information about organizations outside of yours that the directory will serve (if any). Note significant differences in the environment at each location and refer to them as you proceed with the remaining environment topics. For example, some locations will have more users than others, different kinds of computer systems, better network connectivity, and so on.

Organizational structure and geography will influence everything from how many physical servers you need to deploy to how you maintain the data stored in your directory service. The needs of a small organization located in a single building are generally simpler than those of a multinational organization with offices located in many different time zones. In the latter case, seemingly simple decisions, such as choosing a time of day to perform system maintenance, are difficult because of the multitude of time zones in which the users of the system reside. The same issue applies to an extranet directory project in which the users may encompass a worldwide audience.

The amount of independence that individual departments within an organization have varies widely from organization to organization as well. In a decentralized organization, delegating responsibility for directory content to each department or location may be appropriate or perhaps even required. In a centralized organization, management of the directory may be centralized for better efficiency.

Computer Systems

Create an inventory that characterizes the different types of computers in your organization. Also make a record of approximately how many of each different type are in use and the role of each system. [Table 6.1](#) shows a sample computer system inventory.

Also consider the following system-related topics:

- Whether machine upgrades are likely to occur soon
- Whether you will be able to purchase new systems for the directory servers themselves to run on
- Which machines will need to reach the directory servers
- How you will distribute directory service software to machines that need it
- How much control you have over the machines

Table 6.1. A Sample Computer System Inventory

Role	Operating System	Processor	Speed (MHz)	RAM (MB)	Quantity
Low-end desktop	Windows 98SE	Pentium II	350	32	75
Typical desktop	MacOS 9.1	PowerPC G3	350	64	25
Typical desktop	Windows 98	Pentium II	600	128	100
CAD workstation	Windows 2000	Pentium III	800	256	50
Intranet server	Solaris 8	2xUltraSPARC-II	400	512	18

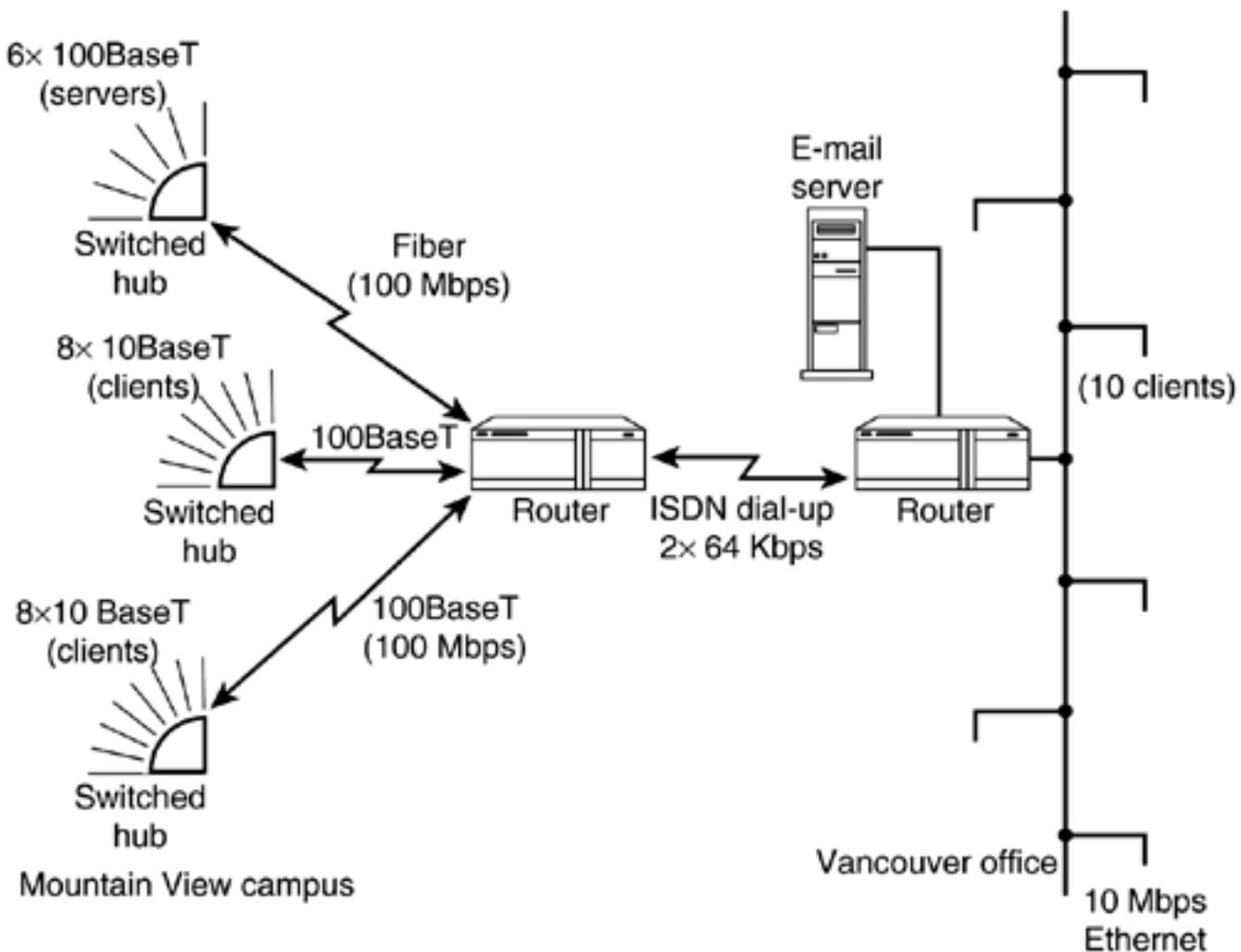
If you are a system designer, administrator, or other information services (IS) professional, you already know how important it is to understand which computer systems are used in your organization. The variety of systems that must be supported by your directory service constrains your choices for directory server and client software.

The Network

Obtain or create a map of your organization's network (or the portion of it on which your directory service will be deployed). This map should show all the backbone and branch networks that exist in the part of your organization that your directory service will serve. Each network link should be labeled with information about its bandwidth, latency, and reliability, along with any use-based costs. In large organizations, usually a central group oversees the network; hopefully the supervisory group will readily provide this kind of information. If necessary, you can measure the characteristics of the link or simply estimate them on the basis of the technology employed.

[Figure 6.1](#) shows a sample network map. Except for the intermittent ISDN line that links the Vancouver office to the main campus in Mountain View, this network is generally good. When designing your directory service topology, you should give special consideration to any part of the network that is particularly weak. For example, it may be necessary to place a directory replica in the Vancouver office to provide good service there.

Figure 6.1. A Sample Network Map



All directory services depend on a computer network to exchange data with applications and for communication between the directory servers themselves. The composition, speed, and reliability of your network heavily influences how many servers you need and what the best location is for each one. The network may partially determine how many users and applications your directory can serve. Finally, in some cases the characteristics of the network may limit the types of directory applications you can support.

For example, because most messaging systems employ store-and-forward designs, a messaging server can usually tolerate some delay in receiving responses from a directory service. In contrast, people who use an online phonebook application are unlikely to tolerate delays that exceed a second or two. To the messaging server, high throughput is important, but to end users low latency (that is, fast response time) is important. The network between the directory server and the applications or end users is one factor that influences the overall throughput and latency of a directory service.

Application Software

In most organizations, typically many different application software packages are in use. For planning purposes, concern yourself with only the most popular, the most critical, and the most directory-enabled applications. Much of the design of your directory service will be driven by the needs of the directory-enabled applications you plan to deploy. These needs are discussed in more detail in the next section of this chapter and in [Chapter 7](#), Data Design.

Determining and Prioritizing Application Needs

It is important to focus first on LDAP-enabled applications when pondering your directory needs because in most organizations one or possibly two important applications are the driving force behind the initial deployment of the directory service. If this is true in your organization, you can focus most of your energy on meeting the needs of these high-profile applications. If you succeed in deploying a directory service that makes these important applications look good, your coworkers, employees, and users will label the directory service a success as well.

Note that you should avoid creating a directory service that is so focused on the needs of a small number of applications that it must be heavily redesigned later to accommodate other applications that come online. As you work through the various stages of designing your directory, remember to consider the broader, long-term picture. As an example, consider the schema for your directory service, which, as you will learn in more detail in [Chapter 8, Schema Design](#), determines what information you can place in your service. Choose general-purpose schemas over extremely specialized schemas whenever possible, so that you can accommodate more applications without changing the schema.

The directory-related needs of applications generally fall into one of these categories: data, performance, level of service, security, or priorities. These are described in the following sections.

Data

All directory-enabled applications access data stored in a directory service. For each application you plan to support, consider in general terms how it will use the directory and what data it requires to accomplish its mission.

Does the application need to store a lot of data? A directory service may have limited capacity for data or may simply slow down as the number of entries and attributes stored in the service increases. It is important to understand the data capacity needs of each application so that you can plan appropriately.

For example, a network printer location service may impose modest data scalability requirements; each printer entry will probably be small, and most organizations do not own millions of printers. In contrast, a calendar or scheduling application that creates an entry for every meeting and task that appears on users' calendars may need to store a dozen or more attributes with each entry—creating literally millions of complex entries in the directory. Meeting the needs of a scheduling application may require partitioning entries among several servers to achieve the required level of performance.

How flexible is the application in terms of the schemas used in the directory? For example, a directory-enabled workflow system may require access to much information about people and their organizational roles and relationships. Such an application may also use the directory for its security needs, so the directory may also be required to store passwords, public key-based certificates, and access control information. Depending on how the workflow system is designed, it may be flexible or rigid about what schemas are used in the directory.

Does the application have any special data needs? For example, does it need to store unusual data types beyond the basics of character strings, phone numbers, and integers? Is the information used by the application so dynamic that it does not necessarily need to be written to persistent storage? Special data needs such as those implied by these questions

may severely limit the choices of directory service software.

Performance

Most applications expect the directory service to meet certain standards for performance. Two aspects of directory performance are of special interest: latency and throughput. *Latency* refers to the elapsed time between when an application makes an LDAP request and when it receives a response. Typically, low latency is most important for applications that use the directory service as part of a larger task, or when an end user is waiting for a response. For example, telephone operators who handle directory assistance inquiries require fast response time from a directory because they cannot move on to another call until they receive a response and pass it on to the current caller.

Throughput refers to the total sustainable operation load that the directory can handle. It is possible for a directory service to have high latency (for example, each search takes 2 seconds to complete) and still have high throughput (for example, the server can process 1,500 searches every second). For applications that make heavy use of the directory, such as an e-mail delivery system, throughput is the most important performance metric. You will need to think about how much work each application expects to be able to accomplish in a given time period. Be sure to distinguish read throughput requirements from write throughput requirements.

[Figure 6.2](#) shows an example of the throughput needs of a hypothetical e-mail delivery system. The bottom line is that the e-mail delivery application requires directory throughput of 133 searches per second in order to support the three message transfer agents (MTAs).

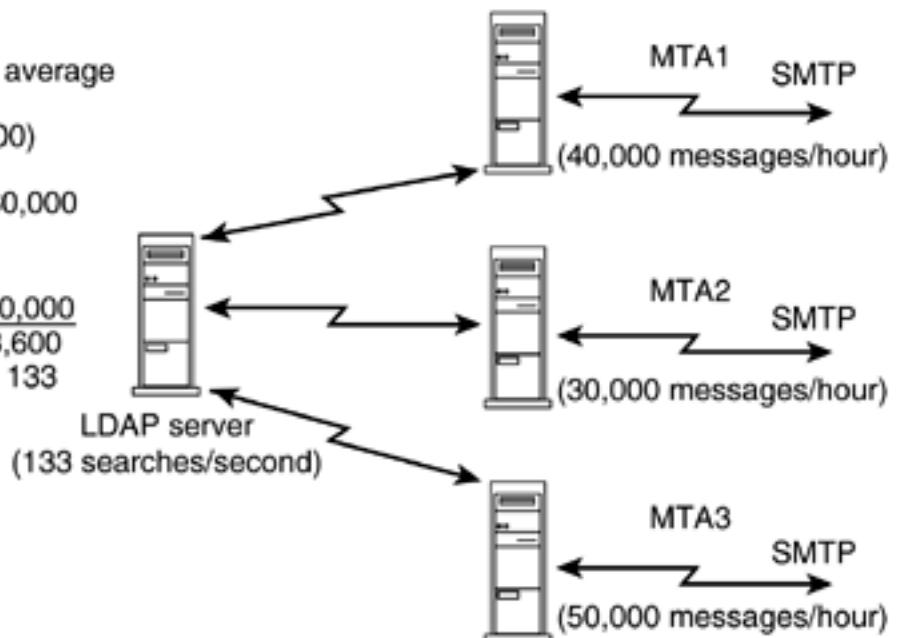
Figure 6.2. Throughput Needs of a Busy E-Mail Service

Calculations:

LDAP searches/message = 4 on average
Messages/hour = 120,000
That is, $(40,000 + 30,000 + 50,000)$

LDAP searches/hour = 480,000
That is, $(4 \times 120,000)$

LDAP searches/second = $\frac{480,000}{3,600}$
divided by 3,600 seconds/hour = 133



Getting a handle on how well your directory service must perform helps produce a good design and ultimately results in a production directory service that enables the applications to perform well. It is always good to avoid a situation in which your directory service becomes a bottleneck for an important application.

Level of Service

Level of service typically encompasses availability (percentage of time the service is up and running), robustness (that is, whether the directory data store is transactional and self-repairing), service monitoring, and disaster recovery procedures. If the application itself provides an experimental service that is not expected to be available at all times, it is acceptable to build the application on top of an experimental directory service. Much of the characterization of service level has to do with the expectations of the people who rely on a directory-enabled application. It is important to know what level of service each application expects so that you can design a service that meets this expectation.

Tip

Over time, your directory service will probably need to support a wide range of service levels, from experimental to production to mission-critical. Keep this in mind as you define needs, set requirements, and design the directory service. Sometimes it is difficult and expensive to raise the service level after a directory is deployed, so you need to make sure that the software and hardware you select can meet your future needs. For example, if you eventually need to support 24x7 access to your directory service, you need to choose LDAP server software and hardware that supports features such as online backup of the directory database.

Security

Think about the security requirements of your directory-enabled applications. Some application security requirements impose security requirements on your directory service as well. Consider these topics:

- **Privacy requirements.** How secret is the data that will be transferred between the application and the directory servers? Should encryption be used to protect the information? For information that needs protection, what strength of encryption is required?
- **Authentication requirements.** Are simple passwords sufficient? Should one-time passwords be used instead? Must public key-based certificates be used to authenticate applications and users? If so, must the certificates and keys be stored securely on smart cards?
- **Access control requirements.** Does the application need to be able to restrict access to directory information? What special needs does the application have?
- **Auditing requirements.** Must a log of all directory activity be maintained so that a security audit can be performed? How long must the activity log be kept?

These topics are covered in detail in [Chapter 12](#), Privacy and Security Design.

Prioritizing Application Needs

The final step in examining any set of needs is to assign a set of priorities to each one. In principle this is a simple task, but if the applications that your directory service is expected to support have a wide range of conflicting needs, setting priorities can be a challenge. A good technique is to make several passes through all the needs.

In the first pass, just assign each application need to one of these three categories:

1. Must have for the application to function
2. Would help the application do its job better
3. Would be nice to have, but is not critical

For example, the capability to customize the directory may be absolutely required to support an inflexible application, so this requirement should be assigned to the first category. In contrast, high directory throughput may help a heavily directory-dependent application work well, but there is probably some room for negotiation on the performance requirements; therefore, this requirement should be assigned to the second category.

In the second pass, think about how important each application is and sort the needs within each category you assigned in the first pass. Because it is unlikely that you will be able to satisfy all the application requirements from the start, ordering the list of requirements is an important step that shows you where to focus your attention.

Finally, make one more pass to check your work. This time, just look at the needs in the bottom half of your ordered priorities list and make sure that none of them seem particularly important. If any do, move them up and reorder the list.

Determining and Prioritizing Users' Needs and Expectations

It is essential to know how many users you're serving, who they are, where they're located, what expertise they have, and what their expectations are. Successful computer professionals know that their primary mission is to meet and exceed users' expectations.

A great way to measure the overall success of the directory service is to ask its users whether the service meets their needs and expectations. Even if you don't measure your success by user satisfaction, someone else (such as your boss!) will. Thus you should make an effort to determine users' needs and expectations in advance of the directory service deployment.

Because most end users are probably not familiar with the concept of an open, general-purpose directory service, you may need to educate them before you can extract useful information about their directory needs and expectations. If you already have some application-specific directories or an existing general-purpose directory in your organization, the task of determining user needs will be easier.

This section discusses some ways you can determine and prioritize your users' needs and expectations.

Asking Your Users

One of the best ways to determine users' needs is to ask them. If it is unrealistic at this point in your directory-planning process to conduct a general survey of your users, begin with an unscientific sample that includes your coworkers and friends and expand your survey later. Some good questions to ask users to help deduce their directory-related needs are these:

- How would you use a directory if you had access to one?
- What kind of data about other people do you absolutely need to perform your job? What kind of data would you like to have access to?
- What kinds of information do you expect to find in a directory?
- What data about yourself would you like to see published in a directory for others to use?
- What applications could benefit from being directory-enabled? How do you expect such applications to behave?
- How could a directory help you get your own work done more efficiently?
- How often will you search for or read information from the directory?
- How often will you make changes to directory information?
- How would you like to access directory information?

Keep in mind that you will need to provide more context and some additional explanations for users unfamiliar with the concept of a directory service, and that some of the preceding questions will not be relevant if your directory service will be mainly behind the scenes and not directly visible to end users. It is often useful to ask yourself these questions first (putting yourself in the users' shoes) and think about how you expect to take advantage of the directory service in the future. As someone in the process of becoming a directory expert (or who is one already), you almost certainly have ideas that your users do not.

In addition to or instead of directly asking users about their needs, you can do the following:

- Look at the existing directories that people are using, such as printed (offline) directories, documents maintained by administrators (for example, Excel

spreadsheets with names and telephone numbers), manually maintained lists of partner contact information, vendor contact lists, and so on.

- Ask administrators of existing systems about the needs of the users they support.
- Talk to managers and directors of specific departments to find out about the needs and expectations of the people who work for them.
- Talk to people who support or manage users (for example, system administrators). This is an especially valuable approach if your directory deployment will replace one or more existing proprietary directories.

Accuracy and Completeness of Data

One area where users often have great expectations (even if they are not mentioned explicitly) is the completeness and accuracy of the data stored in the directory service.

Many people expect the data to be complete and up-to-date, and they may become disillusioned with your service if they discover otherwise. For example, one of the most common reasons users access a directory service is to look up contact information about other people, including voice and fax telephone numbers, e-mail addresses, and postal mailing addresses. If the information in your directory is incomplete or stale, people will quickly learn not to trust your directory service.

Because users may not volunteer these data-related expectations, be sure to ask them in a way that makes sense to them. For example, a question like, What is the maximum tolerable propagation delay when replicating directory data? is much more intimidating than one like, If you change your telephone number, how soon do you expect the change to appear in all the online systems? Even better is to provide choices for the user, such as "Within five minutes," "Within one hour," and "The same day."

Privacy

Another interesting and sometimes contentious set of user needs and expectations is centered on personal privacy. Because your directory service will probably store some data about each person in your organization or each person who visits your e-commerce Web site, it is important to ask your users about their privacy expectations. Whereas some users will be relatively unconcerned about who has access to personal information, others will be very concerned.

When asking people about their privacy concerns, be sure to explain what data you plan to store and to whom you plan to grant access. A lack of understanding of the needs and expectations surrounding personal privacy can lead to some unpleasant political problems as you begin to deploy your directory, so be sure to put some thought into it during the design process. The topic of privacy is discussed in detail in [Chapter 12](#), Privacy and Security Design.

Audience

Next consider how broad the audience for your directory service might become. For example, will the service be accessible only to people inside your organization, or do you plan to replicate some information to a server outside your firewall? If your directory lives outside the relatively safe confines of a firewall, the invited and uninvited audiences for your directory service potentially include everyone on the Internet. For example, you might be designing a directory deployment to service an extranet site visited by partners and suppliers. In such a situation, access control rules would clearly be needed to restrict the audience. It is up to you to decide who the important members of your user base are and design an appropriate service. Access control rules may be used, for example, to limit the

access granted to anonymous users of the directory.

Also consider any special needs of your users. For example, in a multinational corporation or for an e-commerce site, the directory service must serve people who come from a variety of cultural backgrounds; thus, differences in language, privacy expectations, privacy laws, and other areas may be important.

The Relationship of User Needs to Application Needs

User needs and application needs are often tied together. Most directory-enabled applications ultimately serve a set of users, so some of your user needs come to your service indirectly via the needs of applications. Work with those responsible for deploying the applications to understand how the needs of the users of each application affect the design of your directory service.

It is also important to consider the needs of the system administrators of your directory-enabled applications and all the other administrators within your organization. The directory service itself and the tools provided for its maintenance must ultimately meet the needs of both end users and all the different kinds of administrators. For example, you may need to develop a special tool that Help Desk personnel can use to easily reset an end user's directory password. The needs of system administrators often lead to some deployment constraints, as discussed shortly, in the section Determining and Prioritizing Deployment Constraints.

Prioritizing Your Users' Needs

As for all other directory needs, try to order the list of users' needs by importance. The process of setting priorities is based partly on fairness (for example, "Many people mentioned this need") and partly on politics (for example, "The director of my department thinks this is important"). Try to be realistic about what you can accomplish, but also be careful not to place a lower priority on a user's need just because you don't agree that it is important.

Determining and Prioritizing Deployment Constraints

Additional constraints are imposed by the organizational situation and the characteristics of the directory system designers and administrators. If you are one of the system designers or administrators, some of these constraints arise from your personal views. You can rarely alter the organizational situation or people's personal characteristics, but understanding these constraints can help you achieve an optimal design.

Resources

One important set of constraints involves the quantity of resources—money, time, and people—available for the directory deployment effort. When you're creating any new service, it is important to know how much money is available to spend, how much time you have for planning and deployment, and how much effort can be applied to the project. In turn, the leader of the directory deployment effort should provide detailed information to the project sponsors about what resources will be needed.

Resource-rich organizations tend to do things on a larger scale; of course, expectations of those working in such an environment are high as well. In contrast, if you work in a resource-poor organization, typically you are held accountable for all expenditures, and you may have to do more work up front to show that your directory service project will pay for itself. No matter what kind of organization you are associated with, your directory deployment project will be expected to show a good return on investment, so it is important to use resources wisely.

Be sure to set realistic goals. For example, it may be OK to spend three months planning before beginning a directory service pilot, but not if a workflow application that the directory must serve is expected to be fully deployed within that same time.

Openness of the Process

Another set of deployment constraints relates to the openness of the directory design and deployment process. If the directory project is officially blessed and well funded by a centralized information services organization, it probably makes sense to have a fairly open process in which you publish and solicit comments on your preliminary directory design.

It is especially helpful to get feedback from administrators of applications, people who manage other databases within your organization, and end users themselves. However, if the directory project is being conducted in secret on time borrowed from other projects, the design process should be closed. Although in some organizations a secret "skunk works" project of this sort may not be tolerated, in others it is an acceptable approach that can be used to make progress in the face of organizational opposition or indifference.

Skills of the Directory System Designers

We use the term *system designer* to refer to a person who designs and plans for the deployment of information systems. When you're planning your directory service, the most important system designers whose needs must be considered are those who design and plan the deployment of the directory service itself. Information systems professionals with many years of experience assigned full-time to a directory project are more likely to succeed with a "from scratch" approach than people who are asked to work on a directory project in their spare time. Also some organizations and individuals have a tendency to look for turnkey solutions that require less design and experimentation. Alternatively, some people are

inclined to learn everything there is to know about the problem and design a solution from start to finish. Understanding the skills and experience of your team will help you set realistic goals.

Skills and Needs of System Administrators

Consider the skills and needs of the people who will maintain your directory service and associated applications after they are deployed. A *system administrator* is someone responsible for the care and maintenance of a production service. Most system administrators prefer to eliminate or at least automate boring, repetitive tasks. In large organizations, system administrators typically spend much of their time and effort managing organizational data—an area in which your directory service might be able to help make their job easier. By talking to a group of system administrators, you can generate many good ideas for directory management tools, application management aids, and process improvements.

The most important system administrators whose needs must be considered are those charged with running the directory service. The characteristics of the system administrators impose a set of deployment constraints that may affect your approach; ask for and listen to all of their input.

The Political Climate

Geographical boundaries and organizational structure often lead to differences in thinking among groups. Political climate is an important but often misunderstood aspect of an organization's environment. Because political differences often arise as a result of poor communication between groups, they tend to be more pronounced in large, mature, or hierarchically managed organizations.

Tip

You will probably find it difficult to paint an accurate picture of the political climate within your own organization. The best approach is to ask some experienced employees from a variety of organizational units for their own views and look for common themes and complaints. You will still need to sort out fact from fiction, of course.

Political disagreements are usually centered on how to use resources (people, money, time), the direction of the organization as a whole, or philosophical differences between managers. These disagreements may hurt your directory service deployment efforts if you are caught in the middle. If you have been asked to succeed in designing and deploying a fully functional directory service even though several other groups have failed in the past, a short time to market may be an important requirement. Showing some early, useful results will help gain support for continuing work on the directory service.

You may also find that you need to spend considerable time and resources to defuse political conflicts with other parts of your organization. As a directory expert, part of your job is to sell everyone on the benefits of a general-purpose, standards-based directory service. Remember that good communication is the best weapon you have to help reduce the impact of political conflicts.

For example, if at some point during your directory deployment effort you are blocked because of a lack of cooperation from another group, you may be able to defuse the

situation by gaining the support of the other group's management. An effective way to gain support is to prepare a presentation on the benefits of the directory deployment and then sell Management on the idea. It is important not to let your project lose too much momentum or collapse because of political conflicts.

Prioritizing Your Deployment Constraints

As with the application and user needs discussed earlier, list all the deployment constraints you can think of and then assign priorities to each one. Because many deployment constraints come out of organizational or personal characteristics, it is important to be as objective as possible and make sure that the most important constraints overall are near the top of your list. Keep in mind that no matter what you do, you will probably not satisfy everyone. For example, it may be more important to make the director of the Personnel department happy than to make yourself happy.

Determining and Prioritizing Other Environmental Constraints

The final set of constraints to consider is those related to the environment in which the directory service is to be deployed. There are many of these potential constraints, and the list varies depending on the situation, but this section touches on a few of the most important areas.

Hardware and Software

A lot of information about existing systems was discussed earlier in this chapter in the section *Analyzing Your Environment*. When thinking about your directory service, consider the computer hardware and operating systems that it must support. The use of a standard protocol for directory access such as LDAP largely insulates you from the characteristics of any specific computer platform; as a practical consideration, however, your directory service may need to accommodate a variety of existing systems.

Some directory or directory-enabled application software is available for only one or two platforms. Ask yourself the following questions to make sure you take into account these system-related constraints:

- Can you purchase new hardware and software to deploy the directory servers, or do they need to run on hardware you already own?
- What kind of computer hardware and operating systems do your users have?
- What kind of computer hardware and operating systems will your directory-enabled applications need to run on?
- What up-and-coming computing platforms do you need to worry about?

Be sure to note any trends away from certain existing systems and toward others. For example, suppose that use of the Linux operating system is on the rise and Microsoft Windows 98 is on its way out. By the time your directory service is fully deployed, support for Linux may be much more important than support for Windows 98.

Another example is the trend toward wireless and handheld devices. The constraints imposed by these kinds of devices present interesting challenges to the directory designer. For example, connecting a wireless device to your directory infrastructure may require specialized directory access software (and perhaps custom development) because of the network bandwidth and hardware limitations of the device.

The Network

Another set of environmental constraints relates to your organization's network. As discussed earlier, the physical deployment of directory servers may be governed largely by the cost, reliability, and performance characteristics of the network that your directory service relies on. Even worse, if the network within your organization is managed by a completely separate group, you may be powerless to effect much change in the network. In that case you will need to design your directory service to fit the existing network constraints. If changes can be made to the network to accommodate your directory needs, so much the better. The physical deployment of directory servers throughout your network is covered in depth in [Chapters 10](#), Topology Design, and [11](#), Replication Design.

Criticality of Service

Next think about whether your directory service must provide mission-critical services or whether things are a bit more relaxed. For example, if you're deploying a directory service that will be used in a hospital emergency room, the standards that your directory service will be held to may be very high indeed. In contrast, it may be acceptable for a software company's internal directory service to fail occasionally, because the users are familiar with and may be more tolerant of such failures.

Security

Some security and privacy constraints come from the directory users (as previously discussed), but some arise from the environment in which the service operates. For example, if a directory service operates in an open network environment such as the Internet, it is more likely to be attacked by malicious intruders. You do not want a breach of security within your directory service to occur and cause your e-commerce site to be mentioned on the front page of the *Wall Street Journal*, so appropriate steps must be taken. Similarly, if your organization is secretive by nature or deals with sensitive data, the value of breaking the directory service's security may be higher—and the security-related expectations of users may be higher as well. Because privacy and security are so important, we devote all of [Chapter 12](#), Privacy and Security Design, to those topics.

Coexistence with Other Databases and Directories

Finally, a directory service typically must coexist with a variety of directories and database systems. These external systems may be completely independent of your directory service or tightly integrated with it. In the latter case, these systems will impose additional constraints on your directory service, covering a wide range of areas. You can find more information about integrating with other directories and databases in [Chapter 7](#), Data Design, and [Chapter 23](#), Directory Coexistence, where we cover the topic of integration with other data sources.

Prioritizing Your Environmental Constraints

As you did for all the needs and constraints discussed earlier, list the remaining environmental constraints and assign priorities to each of them. Try to distinguish between absolute constraints that you cannot overcome and constraints that you may be able to work around or avoid. For example, it may be difficult and expensive to upgrade all the computer hardware and operating systems that users have, but you may be able to lobby successfully to purchase new systems on which to run the directory servers themselves.

Choosing an Overall Directory Design and Deployment Approach

After you have identified your directory needs, choose an overall design and deployment approach that fits your way of doing things and your specific situation. Your choice will help you succeed with your service deployment—and in the end, that's what matters the most.

Match the Prevailing Philosophy

It is important to choose an approach that fits your philosophy, as well as that of your team and your organization. Some people prefer to spend a lot of time in the design and piloting phases before moving a directory service into production. Others prefer to jump in with both feet and deploy a pilot service with the knowledge that iteration and redesign will take place in the future.

The organization or project for which you are deploying the directory service may also have expectations that are important to meet. For example, if most deployed computing services are 24x7 services that are heavily financed and staffed, it makes little sense to deploy a directory service that is not of similar quality.

Take Constraints into Account

Another factor in choosing an approach that works for you is to consider the constraints under which the deployment must operate. Earlier in this chapter we listed some of these deployment and environmental constraints. Resource constraints (time, money, and people) usually affect your approach the most because they tend to limit in a very real way what you can do and how fast you can do it. The authors know of several successful directory deployment projects that began as spare-time activities, but those projects did take longer to develop into high-quality production services than some other well-funded and well-staffed projects did.

Political constraints are also worth considering, especially if they imply a specific schedule or a specific way of doing things. For example, suppose that there is strong political opposition to or just strong skepticism about the directory project. If so, it may make sense to deliver a basic, useful service as soon as possible instead of taking time to design a be-all and end-all directory service.

Favor Simple over Complex

Another important point is that simple solutions generally work better than complex ones. Combined with a deep knowledge of the subject matter and an incremental approach to design, a "keep it simple" approach is probably your best bet. This book provides in-depth coverage of directory services, but we encourage you to adopt an incremental approach to design and deployment to go with good knowledge of all the issues.

In an incremental approach to design, you begin by choosing simple solutions that address your most important needs and then revisit design decisions later as necessary. In this way, you can feed real deployment experience gained during pilot or production phases of the directory service back into the design process. Finding and deploying a simple solution is not necessarily quicker than finding and deploying a complex one, but simple solutions do tend to be easier to debug, easier to understand, and more flexible.

Focus on the Most Important Needs

Finally, it is best to focus initially on the most important directory-related needs, perhaps the top five. If, instead, you try to meet all the needs you are aware of, you may end up not meeting any of them well. Using the process outlined in this chapter to locate and prioritize application and user needs should help you find the most important areas to focus your efforts. Having focus will also make it much easier to measure your progress, as discussed in the next section.

Team LiB

◀ PREVIOUS

NEXT ▶

Setting Some Goals and Milestones

Although setting goals and milestones is not specific to the topic of directories, good goals and milestones are important for your directory design and deployment efforts. This task is often overlooked, even though everyone knows that goals and milestones are part of any good project plan. In this section we discuss the purpose of goals and milestones and explain how to choose them. Think of this section as Project Management 101 for directory project leaders.

Goals

Good goals help motivate people by providing a target to aim for. They help set and communicate expectations both inside and outside the directory deployment team. Goals are used to measure accomplishments and recognize success. For your directory service effort, try to choose some goals early so that you have something to shoot for throughout the project.

A good goal tells both *what* is expected and *when*, and it is easily understood by everyone involved in a project. Goals should be realistic, and the people who are asked to meet them should believe that they are achievable. Goals should also be measurable; that is, it should be clear to everyone when a goal has (or has not) been reached.

Some examples of good directory service design and deployment goals might include

- Requirements document completed and published for other groups to comment on
- Initial directory design completed and published
- Directory service up and running in a pilot environment
- Revised directory design completed and published
- Production-quality service deployed with one directory-enabled application relying on the service
- During every week, 8 out of 10 employees directly or indirectly using the directory service
- A 20 percent reduction in the time it takes to create all the necessary accounts and prepare the computing environment for a new hire
- A 40 percent reduction in the time and effort required by system administrators to manage data

Of course, you should choose target dates for these goals. Assigning accurate dates is usually difficult unless you have worked on a similar directory design and deployment project in the past. For some of the more distant goals, you should avoid trying to set a target date initially and just plan to set it as the project progresses. In some situations, a schedule may be imposed on you, and you won't have that luxury!

Milestones

Use a set of milestones to measure progress toward your goals. Milestones are especially important when the time frame in which you might achieve a particular goal is well into the future (more than a few months away); this way, you can use the milestones to check whether you are on track to meet the goal. Milestones also help you know when to adjust your overall schedule. If you achieve a milestone early, it might be possible to pull your entire schedule in and shorten the project plan. If you're late in reaching a milestone, the overall schedule will probably need to be lengthened. Finally, showing advancement against a set of milestones demonstrates real progress, which can help you when you need additional funding or buy-in from other groups.

Like good goals, good milestones help motivate people by providing intermediate targets they can aim for. As someone once said, "If not for deadlines, nothing would ever get done." If you're managing a directory design and deployment project, be sure you set achievable milestones and encourage individuals and the team to celebrate when those milestones are reached.

Choosing milestones is often easy after you pick your goals. The milestones should simply fall out of the goal as a series of tasks that must be accomplished to achieve the goal. For example, some reasonable intermediate milestones for the second sample goal mentioned earlier ("initial directory design completed and published") might include the following:

- Data and schema design completed
- Namespace design completed
- Server topology and replication design completed
- Privacy and security design completed
- Initial draft of complete design shared with your immediate colleagues to solicit feedback

Keep in mind that the purpose of milestones is to help motivate you and your team and to mark progress. If creating and tracking a large list of detailed milestones does not fit with your work habits and you feel you can succeed without them, feel free to ignore our advice here. For most people, though, milestones are a useful tool for marking progress as they move through a large project. It is usually more rewarding to complete small subprojects at regular intervals rather than to wait for the entire, long project to be completed.

Recommendations for Setting Goals and Milestones

We recommend that you select two or three short-term goals (achievable within the next three to six months) and two or three long-term goals for your directory design and deployment effort. Then for each goal set some intermediate milestones. The milestones can be as frequent as you want, but a good rule of thumb is to create milestones that you will hit every couple of weeks.

If you have a team of people working on the directory service, be sure to involve them in the goal- and milestone-setting process and make sure that they know what is expected of them individually. People who are involved in setting the goals and milestones will have a better understanding of them—and they'll be more motivated to achieve them.

Tip

Share status information about goals and milestones with all who are involved or interested in your directory service efforts. Actively use the milestones to track progress toward your goals. By advertising exactly what you hope to accomplish and when, you implicitly enlist the aid of others and set reasonable expectations for the directory service deployment process.

Defining Your Directory Needs Checklist

You should perform the following tasks to determine your directory needs:

- Analyze your environment.
- Determine and prioritize needs:
 - Application needs
 - Users' needs and expectations
 - Deployment constraints
 - Other environmental constraints
- Choose an appropriate directory design and deployment approach.
- Set some achievable short- and long-term goals and milestones.

Further Reading

Building an X.500 Directory Service in the US (RFC 1943). B. Jennings, 1996. Available on the World Wide Web at <http://www.ietf.org/rfc/rfc1943.txt>.

IBM LDAP Implementation Cookbook. H. Joiner, M. Melot, H. Strandén, and P. Widhiasta, International Technical Support Organization, 1999. Available on the World Wide Web at <http://www.redbooks.ibm.com/redbooks/SG245110.html>.

Microsoft Active Directory planning and deployment guides. Available on the World Wide Web at <http://www.microsoft.com/windows2000/technologies/directory/ad/default.asp#section10>.

Netscape Directory Server Deployment Guide. Available on the World Wide Web at <http://enterprise.netscape.com/docs/directory>.

Designing Your Novell eDirectory Network. Available on the World Wide Web at <http://www.novell.com/documentation/lg/edir87/edir87/data/a2iiido.html>.

Looking Ahead

In [Chapter 7](#), Data Design, we will discuss data design for your directory service. We will use the list of directory-enabled applications you made in this chapter to seed the process of determining what data must be stored in your directory service.

Chapter 7. Data Design

- Data Design Overview
- Common Data-Related Problems
- Creating a Data Policy Statement
- Identifying Which Data Elements You Need
- General Characteristics of Data Elements
- Sources of Data
- Maintaining Good Relationships with Other Data Sources
- Data Design Checklist
- Further Reading
- Looking Ahead

Because a directory service is (among other things) a specialized database, it should not come as a surprise that some of the most important directory design considerations involve data. It is critical that you put some effort up front into thinking about what kind of information you will store in your directory service and how you will obtain and manage that information.

In this chapter we first examine the major issues surrounding directory data and describe some common data-related problems. We also explain how to define a data policy statement that is appropriate for your deployment. Next we explore some techniques for identifying what data should be stored in your directory service. Then we take a step back and look at some general characteristics of data. Finally, we discuss some potential sources for data and tips for maintaining good relationships with all the data sources that exist within your organization.

Data Design Overview

The term *data* refers to the entire collection of information stored in your directory service. We also use *data* as a general term to refer to information of any kind regardless of where it is stored. Data is what makes a directory service interesting; after all, your directory service is only as useful as the data it holds. Furthermore, the issues surrounding data sometimes evoke strong emotional and political responses because people care about their personal information, their department's information, customer information, and Web site visitor information. In addition, people who write or deploy directory-enabled applications have their own data-related needs. Some of the different needs will inevitably conflict, so data design is an important and potentially complex area.

Some other data-related topics are explored in other chapters. For example, when designing a schema (see [Chapter 8](#), Schema Design), we will use the results obtained in this chapter to choose LDAP object classes and attributes to hold your data. The design of your directory's namespace (see [Chapter 9](#), Namespace Design) will also build on the material in this chapter as you strive to organize your directory data in a logical structure to optimize administration and operation of the directory service. In [Chapter 18](#), Maintaining Data, we will explore topics related to keeping your directory data up-to-date and accurate over the lifetime of your directory service.

In [Chapter 6](#), Defining Your Directory Needs, we examined directory needs in detail, focusing on the applications that will use your directory. As already noted, the applications deployed against your directory greatly influence the design and deployment of the entire directory service. Applications are the strongest driving factor in identifying which pieces of data should be stored in your directory and how you should manage them.

The term *data element* refers to the pieces of data, and *data source* refers to any system that stores a collection of data elements. Data sources are sometimes called *repositories*. In LDAP terms, data elements correspond closely to *attribute types*. Examples of data elements include a person's full name, a printer's paper capacity, or a computer's processor type. Specific instances of data elements are called *data element values* or just *data values* (for example, the full name "Babs Jensen"). In LDAP terms, data values are called *attribute values*.

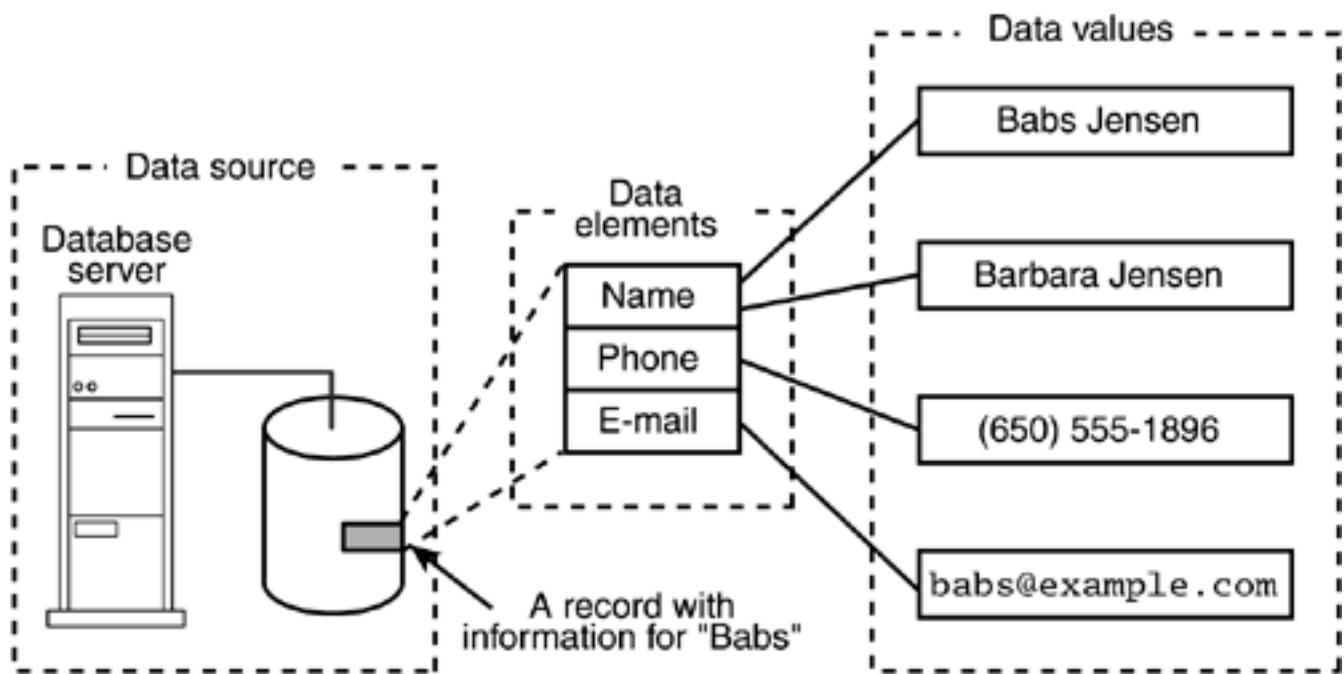
Note

In this chapter we use less specific, non-LDAP terms such as *data element* so that we can direct all our attention to the issues surrounding the data itself. When we tackle schema design in [Chapter 8](#), we will return to using the LDAP terms as we focus on how data elements are mapped specifically onto the LDAP information model.

Within your organization, data probably is stored in many places. For example, the Personnel or Human Resources department may manage a data source that includes information about all people at the company who receive a paycheck. Such information is typically managed with human resources management software, which typically stores data in a relational database management system (RDBMS).

Of course, a directory service is also a data source. [Figure 7.1](#) shows how data sources, data elements, and data values are related.

Figure 7.1. Data Sources, Data Elements, and Data Values



One important question you will need to answer is, How does the data stored in the LDAP directory service relate to the data held in the other important data sources within my organization? To answer, you will want to learn as much as you can about the other data sources that exist within your organization.

Common Data-Related Problems

Although we live in the information age, rarely is the right information available at the right time. As individuals, typically we are affected by one or more of the following problems:

- Too much information (it is difficult to find exactly what we're looking for)
- Not enough information (the information we need is not available)
- Poor-quality, improperly formatted, or just plain wrong information (we may be misled)
- Out-of-date information (we may be acting on yesterday's news)

Applications that use your directory service will suffer from the same problems unless you give careful consideration to which data elements you include in your directory service, where to obtain the data, and how to manage it. For example, applications that rely on the directory as a source of contact information such as phone numbers or e-mail addresses will succeed, in their users' eyes, only if the information is accurate and up-to-date.

In addition, even the most accurate information may be hard to use if it is not stored in a consistent way. For example, a phone number might be stored as 555-1212, (650) 555-1212, +1 650 555-1212, 1.650.555.1212, or yet another form. The key to understanding and avoiding data-related problems is to develop a good understanding of how both end users and applications use your directory service, and then to put in place policies, procedures, and software to smooth out the problems.

Another potential problem is especially common within large deployments: data redundancy without coordination. *Data redundancy* refers to multiple copies of data elements and values kept in more than one data source. Problems arise when two or more systems store the same data element but do not coordinate changes to the element. If you are fortunate enough to be designing a new directory service to support a new application, you should be able to avoid data redundancy problems entirely. Most of the time, however, you will need to address some data redundancy problems.

For example, if your home postal address is stored both in a centralized human resources database and in a database maintained by the corporate travel office, you may need to contact both offices when you change your address. In large organizations, such personal information commonly is stored in several uncoordinated data sources. Often these different sources are difficult and expensive to manage because they all use different computer systems and software packages to manage the same kind of information.

Don't be surprised to find data redundancy problems that do not involve people-related data. For example, within an organization's Information Services division, the Networking group and Help Desk group may maintain separate data sources that include information about the computers and other devices connected to the network.

The best solution to data redundancy problems is to eliminate redundant data sources and synchronize the remaining ones. This challenging task requires good communication and cooperation among all the data owners. This topic is discussed in detail later in this chapter in the section titled *Maintaining Good Relationships with Other Data Sources*, and in [Chapter 23](#), *Directory Coexistence*.

Creating a Data Policy Statement

Before you begin identifying and characterizing the data elements that you plan to store in your directory service, it is important to develop some general guidelines about directory data. These guidelines should be collected in a written data policy statement. The purpose of such a statement is to help everyone affected by your directory service to understand in general terms how data will be handled. Because this group includes you, your directory deployment team, data source owners, application authors, and end users, you should widely publish your data policy statement throughout your organization and perhaps outside (if some or all of the end users are outside your company, for example).

Your data policy statement should cover the following topics:

- **Guidelines for determining what data will and will not be stored in your directory service.** For example, the general guideline could be that any data element that is likely to be shared by more than one application will be stored in your directory. You might decide that large values (greater than 10K) will never be stored in your directory.
- **Guidelines for access to directory data.** This topic is especially important if you plan to store any sensitive information in your directory service. You should also include general guidelines on the kind of authentication and encryption required for access to directory data.
- **Guidelines for modification of directory data.** This topic might include information about whether you expect end users to be allowed to update their own entries, the capability of applications to modify entries, and other "data ownership" issues. You should also include general guidelines on the kind of authentication and encryption required when changes are being made to directory data.
- **Legal considerations.** Because of privacy laws, employment contracts, or other legal considerations, there may be certain kinds of information you may simply not be able to store in your directory service or allow people to access. It is best to involve your organization's legal staff when you're formulating this aspect of your policy.
- **Guidelines for maintaining data stored in more than one location.** Typically, data elements will be stored in your directory service, as well as in one or more external data sources. Topics such as how to handle data flow between the sources and which source will be authoritative should be covered by a general data policy.
- **Guidelines for handling exceptions to your general policies.** Because no policy can cover all possible situations, you should define a simple process for handling exceptions.

Your data policy statement should be a fairly stable document. However, you will inevitably need to evolve your policy as your mission changes, as you learn more about managing your directory service, and as external factors such as privacy laws change.

Because your data policy statement will cover a lot of ground, it is essential that you involve other groups within your organization in the process of creating and reviewing the policy. In many cases the data policy will actually be defined mainly by people outside your directory team. For example, the owners of important data sources and your legal department will undoubtedly have a lot to say about how you should handle data.

Here are some specific groups to enlist when defining your directory data policy:

- Your directory design and deployment team
- People who maintain other important data sources within your organization, if applicable to your directory deployment (for example, the human resources department)
- Authors and deployers of important directory-enabled applications
- The end users who will use your directory service and the directory-enabled applications
- Your legal department
- Upper management, including your chief information officer (CIO) or even the office of your chief executive officer (CEO)

Now that you have a good start on creating a data policy statement, it is time to examine the specific data elements you will store in your directory. Looking at specific examples of data elements will also help you firm up your data policy.

Team LiB

◀ PREVIOUS

NEXT ▶

Identifying Which Data Elements You Need

As we saw in [Chapter 6](#), Defining Your Directory Needs, much of the design of your directory service will be driven by the applications that use it. Now examine each directory application to determine what data it will use. In the first pass, just look at each application and list all the data elements it will use. To obtain this information, you will either need to become an expert on the application itself or enlist the help of others.

For commercial software that supports LDAP, a list of data elements in the form of attribute types should be provided somewhere in the documentation. If not, contact the company that produced the software and request the information. For custom applications created in-house, a design document should be available that includes the information. As the directory expert, you may want to help adjust the design so that the application uses the directory wisely. [Table 7.1](#) shows some examples of applications and the data elements they might require.

Table 7.1. Applications and Their Required Data Elements

Application	Vendor	Class of Information	Required Data Elements
Policy management agent for a Web server	Netegrity	People	User ID, password, department
		Groups	Group name, description, list of members, owner
Electronic mail system	sendmail	People	User ID, password, e-mail address, mail host, vacation message, delivery options, forwarding address
		Groups	Group name, description, list of members, owner

Company phone book	Developed in-house	People	Name, e-mail address, phone numbers, user ID, password, mailing address, department, manager, home page URL, car license plate number
Organizational chart generation utility	Oblix	People	Manager, employee type, job title
Asset management application	Developed in-house	Computers	Owner, make, model, CPU, speed, storage capacity, IP address, host name

After you have compiled a list of the data elements used by each application, locate the data elements that are used by more than one application. It is almost always better to simplify and use as few data elements as possible. Failure to do so can create data redundancy problems within your directory service itself ! For the applications shown in [Table 7.1](#), several data elements can be shared between the applications. For example, the first three applications require access to a person's user ID and password.

Often one or two applications drive the initial directory service deployment, and it is appropriate to focus only on the needs of those applications. Keep in mind that you will probably need to revisit decisions made now when other directory-enabled applications come online, especially if the new applications use some of the same data elements as the first round of applications. Directory services software is generally designed so that it is relatively easy to add new data elements incrementally. However, be careful not to combine data elements that may need to be separated later; it may be difficult to determine how to divide an existing data value (for example, reliably extracting a person's surname from his full name).

Try not to let your thinking become too specialized during the data design phase of directory planning. Always consider what will happen if your directory is asked to support new applications. It makes sense to do a significant amount of up-front data planning if one or more of the following apply:

- You are replacing a directory service that is already deployed and in use.
- You plan to serve a wide variety of applications with your directory service (as opposed to providing a specialized service focused on just one or two applications).
- Your service must interact with many other data sources.
- For political or practical reasons, you need to involve people outside your team in the data design process.

When you need to get a handle on the data elements already in use, it is helpful to create a *data sources inventory*. Simply put, this is a list of all the data sources (databases and directories) in use within an organization that are relevant to your directory deployment. It should include a fair amount of detail about the database tables in use, including information on each field that appears in each table. Similarly, it should show all the attributes and object classes in use in the directory service. The data sources inventory should also include

pointers to supporting documentation, when available, and may include contact information for those responsible for each database.

Creating a data sources inventory can be costly and time-consuming for large organizations, but it should be valuable when the time comes to ponder the relationship between the data elements you plan to store in your directory service and those held in other data sources. [Table 7.2](#) shows a sample data sources inventory.

Table 7.2. A Sample Data Sources Inventory

Data Source	Software/Contact	Class of Information	Data Elements
Human resources management system	PeopleSoft (jeffw@hr.example.com)	People	Name, address, phone, employee number, and so on
E-mail system	Microsoft Exchange (leif@is.example.com)	People	Name, user ID, e-mail address, preferences
		Groups	Group name, list of members
Product development phone list	Excel spreadsheet (johnb@pd.example.com)	People	Name, e-mail address, phone extension, home phone

In smaller organizations and for simple directory deployments, there is invariably less need to involve people from other groups in your directory design. There may not even be any other groups!

Tip

Be careful to plan for the future, especially if you work in a growing organization. If you hear that the two-person team that currently produces the paychecks by hand is hiring six more people and meeting with PeopleSoft to discuss software and support needs, you should be prepared to adjust your directory plans accordingly.

After you have completed your data sources inventory, you may find that you have a long list of data elements. This is not something to be too concerned about—directories are designed to handle many data elements without much overhead. The next step is to look at the characteristics of each data element in more detail.

General Characteristics of Data Elements

All data elements share some general characteristics:

- Format
- The size of each data value
- The number of distinct data values
- Data ownership and restrictions
- Consumers
- Frequency of changes in values (dynamic versus static)
- Range of applicability (shared versus application-specific)
- Relationships with other data elements

Each characteristic mentioned in the preceding list is discussed in more detail in the following sections.

Tip

Before you design your directory schema (a topic we will tackle in [Chapter 8](#)), you should characterize each element using the guidelines included in the following sections. Add this information to the list of data elements you created when you examined your application's needs.

Format

Data elements can be grouped according to the natural format of the information. For example, people's names are always textual data, but telephone numbers consist primarily of digits. [Table 7.3](#) shows some of the more common data formats and provides sample data elements for each.

If your textual data is written in more than one character set or language, be sure to note that as well. As you will see in [Chapter 8](#), Schema Design, each LDAP attribute type is assigned a *syntax* and a set of *matching rules* that precisely define the rules for interpreting the stored values. For example, the `cn` (common name) attribute is of the syntax `DirectoryString` with a `caseIgnoreMatch` matching rule. This means that Unicode characters can be stored in a `cn` attribute and that the case of letters that make up a name is not significant in comparisons of one `cn` value with another.

The Size of Each Data Value

Knowing the approximate size of each value in bytes will help with the more directory-specific design work that we will tackle in subsequent chapters. Although it is sometimes difficult to assign hard limits for the size of a data element, it is usually relatively easy to come up with a range that encompasses the typical data values that will be used. For example, the elements of a North American telephone number combine to require approximately 16 characters of storage (1 character for the North American country code of 1, 3 for the area code, 7 for the local number, and 5 for optional spaces and punctuation).

At the other end of the scale, if you choose to store images in your directory service, the

size of each value will be much larger (for example, 50K or larger). Be sure to check your directory server software to see whether it places any restrictions on the size of the data values.

Table 7.3. Common Data Formats

General Format	Common Variations	Sample Data Elements
Text string	Case sensitive, case insensitive	Person's name, printer's name, URL
Multiline text string	Case sensitive, case insensitive	Postal address, description
Phone number	Local, international	Work phone number, fax number
Numeric	Integer, floating point	Employee number, cost
Multimedia	Image, sound, movie	Photograph, music sample
Binary	(Many variations)	Digital certificate, preferences data

When considering how large text data values will be, do not forget to take into account the character set if you have any data that is not plain ASCII. As explained in [Chapter 2](#), Introduction to LDAP, the UTF-8 encoding of the Unicode character set is used to represent international data in LDAP directories. UTF-8 is a variable-length encoding: Each UTF-8 character requires between 1 and 4 bytes, and each ASCII character requires only 1 byte.

Tip

In some cases it makes more sense to store a pointer to a data value in the directory service instead of storing the data value itself. This pointer-based approach is especially useful when the data value is large but related to another data element that you plan to store in your directory.

For example, if you are storing information about all your department's current projects, you may want to include HTTP URLs that point to the detailed project plans. The project plans themselves should remain on Web servers because they are probably large, complex documents, but users and applications will still be able to locate these documents by consulting the directory service.

The Number of Distinct Data Values

For each data element, you should answer the question, How many data values will this

element typically have? For example, a person will usually have only one user ID but may have several phone numbers. This information will help in directory capacity planning and may also be useful if you replicate or synchronize with other data stores that may have different characteristics. It will be important to know, for example, if a data store can store only one value for a person's name, whereas your directory can store many values.

Data Ownership and Restrictions

Dealing with data ownership issues is one of the more challenging aspects of directory design. When you're thinking about access control, privacy, and security issues, it is important to know exactly which people and applications should be allowed to view or modify a data element. Data ownership also affects whether you will allow a data element to be changed by directory clients and whether and how a data element is kept in sync with other data sources. Often various restrictions will be imposed on a data value by the data owner or another interested group.

In addition, data owners may associate business rules with critical data elements, and these rules may restrict who can view or update the data element. For example, the human resources department probably has a well-defined process that restricts who can assign employee numbers to new hires, and it may also impose restrictions on who can see employee numbers.

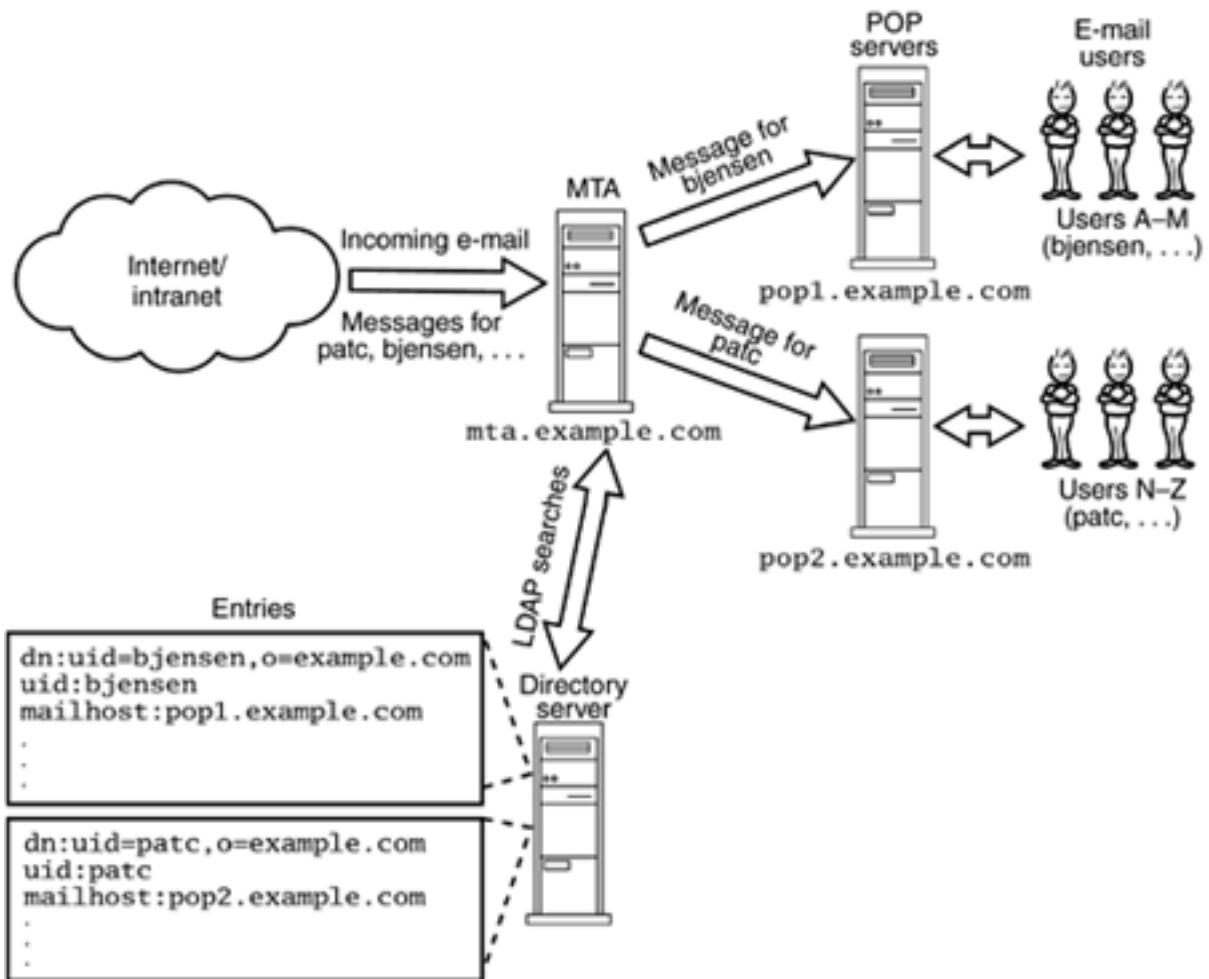
Some questions to ask yourself include, Who should be notified when this data element is modified? If this data element is stored in more than one data source, which system has final authority over the data element? and Who has money or another stake riding on the accuracy of this data value?

Consumers

The consumers of a data element are the directory-enabled applications and external data sources that use it. When you're planning directory replication and topology, and managing the relationships between your directory service and other data sources, it is helpful to know about the consumers of each data element.

For example, a message transfer agent (MTA) is a piece of application software whose job is to route e-mail messages to their correct destinations. When processing e-mail, LDAP-enabled MTAs may look at an attribute in a user's entry called `mailHost`, which gives the host name of the server that holds the user's e-mail (see [Figure 7.2](#)).

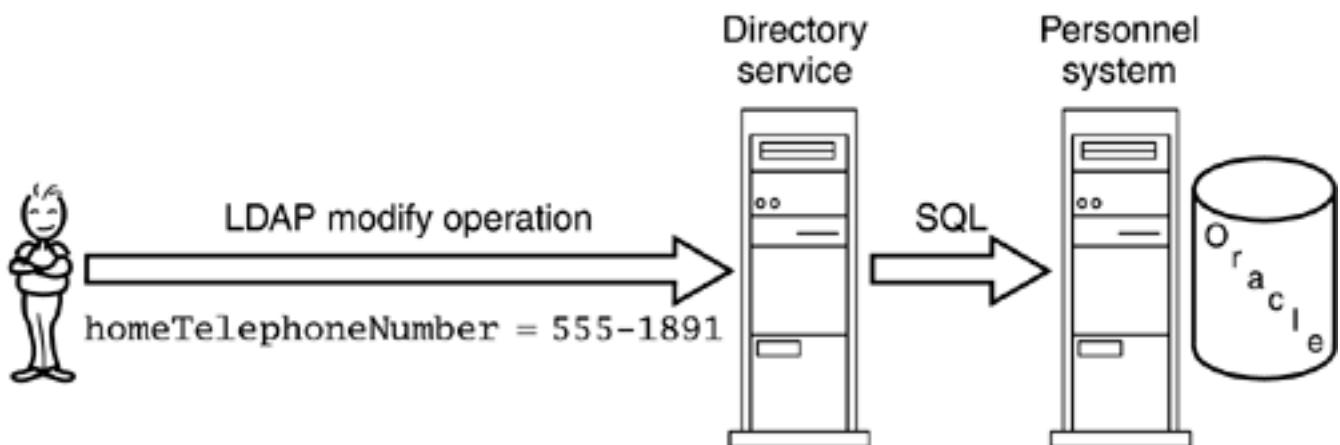
Figure 7.2. An MTA and the `mailHost` Attribute



The MTA is thus an important consumer of the `mailHost` attribute, so it may be important for the MTA always to get the most up-to-date copy of the `mailHost` attribute that is available. E-mail client software might also use the same `mailHost` attribute to determine the location of a user's mail drop, in which case `mailHost` becomes a shared attribute.

Similarly, users may be allowed to change their home telephone numbers in the directory service, and these changes may be propagated to a personnel system that stores its data in an Oracle database. [Figure 7.3](#) shows this scenario.

Figure 7.3. Home Phone Number Propagated to Personnel Database



In this scenario the personnel system is a consumer of the home telephone number, but it is also a data source for other, non-LDAP applications that may access the phone number directly from it. If the home phone number is not a piece of data critical to the personnel

system, it may be OK for updates between it and the directory service to be fairly infrequent. However, if the phone number is critical to both the personnel system and the directory service, a process that accomplishes frequent synchronization may need to be developed.

If you compile detailed information about all the consumers of a data element, you can aggregate all the information into an estimate of how often a given data element will be accessed. Again, this information is useful when you're doing capacity planning for your directory service.

Frequency of Changes in Values: Dynamic or Static?

It is also helpful to know which data elements are *dynamic* (that is, have values that change often) and which are *static* (that is, have values that change infrequently). You will need this information for design of your directory server topology, as well as for capacity planning. For example, suppose you use a replicated directory service that allows writes for a given entry to occur on only one server (a single master system). If you have many attributes whose values change often, you may need to partition your data to avoid overwhelming any one master server with the write traffic.

One way to characterize the dynamic or static nature of attributes is by estimating the ratio of reads to writes for each data element. For example, a user ID may be written once when a student joins a university but read dozens of times each day as e-mail is delivered; this attribute is static because it has a read-to-write ratio that is effectively infinite. In contrast, if a Web browser stores a user's personal bookmarks in the directory service, they may be changed once a day or more often; these attributes are dynamic because they have a read-to-write ratio that may be close to 1:1.

Range of Applicability: Shared or Application-Specific?

Some data elements are used by many applications; others may be used by only one application. Shared data elements require careful planning so that the needs of all the applications are met adequately. On the other hand, if a data element is used by only one application and the data values are large or accessed frequently, consider keeping the values outside your directory service to avoid performance problems when accessing the values. In your data policy statement you may want to include some guidelines for making this kind of decision (as discussed earlier in this chapter).

Tip

One thing to watch for if you conclude that a data element is application-specific is that, over time, new applications may come online that also use the data element. When in doubt, assume that a data element will be shared.

Note that even if data elements are not shared by more than one directory-enabled application, it may make sense to store them together to ease manageability or improve the availability of the information (through directory replication). For example, it may be desirable to delete a person's e-mail-related data elements when the person is deleted from the corporate phonebook. The easiest approach is to store the e-mail-related elements in the directory service along with all the person's contact and other information. That way, you won't need to delete the information in both the directory and the mail system to delete a user's record.

Relationships with Other Data Elements

When you're selecting a schema and laying out the namespace for your directory service, it is useful to know how your data elements are related. Because directory entries typically represent real-world objects, it is important to know which data elements relate to the same kind of object.

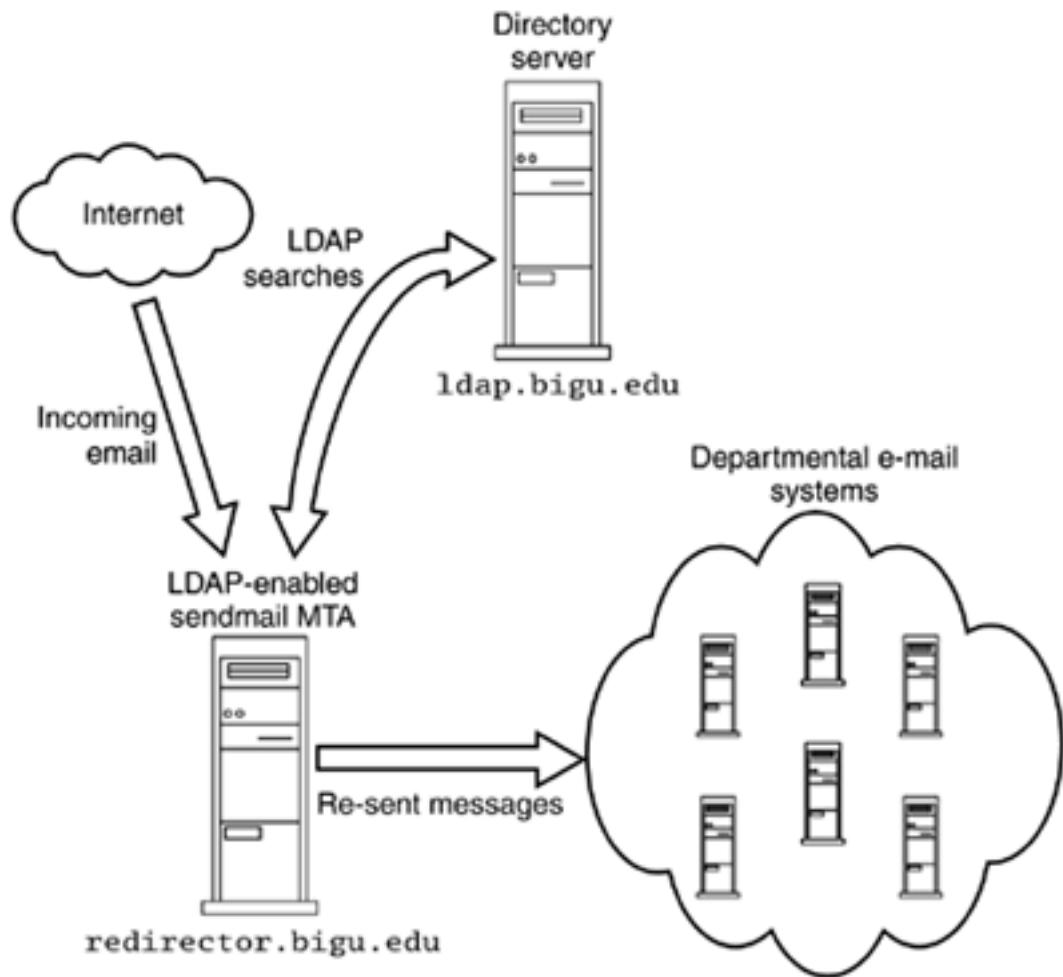
For example, if you have an entry in your directory service for each printer attached to your network, you will want to make it easy for an application to find all the printer-related data elements. You can accomplish this by choosing a schema that defines an all-inclusive printer object (see [Chapter 8](#), Schema Design, for more information on schemas).

Some relationships between data elements are subtler and may be easily overlooked. For example, if you use the directory service to determine who used a printer in the previous 24 hours, you will need to relate information about some users to the printer's entry. You could address this need by including in each printer entry a set of data elements that point to the entries of the people who have recently used the printer.

A Data Element Characteristics Example

Suppose that you work at a large university with a great variety of installed e-mail systems. E-mail is often a major factor affecting the expenditure of information technology dollars, and you want to show your boss the value of your new directory service. To do this, you decide to develop as your first directory-enabled application a service that reroutes all e-mail entering the university from the Internet to the correct system. [Figure 7.4](#) shows the basic setup.

Figure 7.4. The Business Card E-Mail Service



By configuring your Domain Name System (DNS) correctly, you arrange for all e-mail messages sent to `string@bigu.edu` to arrive on the machine called `redirector.bigu.edu`. This machine runs a copy of the sendmail software that has been configured to perform mail routing using an LDAP directory service. Using criteria constructed from `string`, the MTA searches the directory service running on `ldap.bigu.edu` for a user's entry. For example, if a message is addressed to `babs.jensen@bigu.edu`, a search with an LDAP filter of `cn=babs jensen` is performed. If an entry is found, the e-mail message is re-sent to the user's mail delivery address, which is typically a mail server in the individual's department or school within the university.

We christen this service the Business Card E-mail Service, or BCES, because now people can safely put one centrally managed e-mail address on their business cards that will not change even if they switch departments within the university.

One of the first design questions we need to answer is, What data elements do we need and what are their essential characteristics? [Table 7.4](#) provides one potential answer (the real answer depends on exactly which version of the sendmail software is used and how it is configured).

Note that some characteristics are missing from [Table 7.4](#). We don't include any information on how dynamic each data element is because we believe that none of the data elements hold data values that change often. Also, because we are focused on just one application, we have not yet needed to think about which data elements will be shared with other directory-enabled applications.

Table 7.4. Sample Analysis of Data Element Characteristics

Element (Example)	Format	Size/ Number of Values	Owner	Consumers	Related to
Full name (John Jones)	Text	<128 chars. /1 or a few values	Personnel dept.	Users; BCES	User's entry
User ID (jjones)	Text	<8 chars. /1 value	IS dept.	BCES	User's entry
E-mail address (jjones@bigu. edu)	Text (Internet mail address)	Many chars. /1 or a few values	IS dept.	Users; BCES	User's entry
Delivery address (jjones@math. bigu.edu)	Text (Internet mail address)	Many chars. /1 value	User and system admins.	BCES	User's entry

Analyzing Data Elements

After you have compiled a fairly complete list of data elements, examine each data element you plan to include in your own directory to determine which characteristics it shares with others. The goal is to eliminate redundant elements and to develop a clear understanding of how each element will be used so that you can ensure that it makes sense to include it in your directory. By doing this analysis up front, you will save time during the schema and namespace design stages and avoid deployment problems.

For example, suppose that a certain dynamic data element will be modified on average many times each minute, but the data element is used by only one application. Because most directory implementations are optimized for read operations (and update operations are relatively slow), it may make more sense to avoid storing such a dynamic, application-specific data element in your directory service altogether.

Sources of Data

Now you should have a list of all the data elements you want to include in your directory service. It is time now to take a slightly broader view and determine where you will obtain the values for the data elements and how you will manage the relationships with other data sources.

By now you probably have a pretty good idea what data elements exist in electronic form in the other data sources within your organization. Some data elements are unique to a new application you plan to deploy; thus the data values themselves may not exist anywhere yet. The following are common data sources:

- Other directory services
- Network operating systems
- Databases
- Simple files or spreadsheets that hold data in electronic form
- Applications
- Administrators
- End users

You may need to obtain data from any or all of these sources. For each data element you plan to include in your directory, you should decide the following:

- What data source will be consulted to obtain the initial data values?
- Will the data element be updated from other data sources on an ongoing basis?
- Will changes made in the directory be propagated back to any other data sources?

The information you have already gathered about each data element should help with these questions. You may not have all the answers when you initially deploy your directory service, but it is important to give some thought to these issues in advance. You do not want your directory service to hold out-of-date information, nor do you want your directory service to be accused of trying to supplant other important data sources such as the Human Resources department's database (unless that is part of your charter). Common data sources and surrounding issues are discussed in more detail in the following sections.

Tip

When surveying the data sources present in your organization, be sure to ask yourself whether each one will be important a year or two from now. If a system will be decommissioned in the near future, you may not want to invest much effort in studying it (unless of course your directory service is destined to replace it).

Other Directory Services and Network Operating Systems

Your organization may already have a large, central directory service. If not, you probably have departmental local area networks (LANs) that include some kind of network operating system such as Novell NetWare (with data held in Novell eDirectory servers), Microsoft Windows 2000 Server (with data held on Active Directory domain controllers), or Sun's Network Information Service (with data held in NIS servers). You probably also have data in DNS servers.

If the kind of data held by these systems is relevant to your directory project, find out what data these systems contain and how it is being used. Think about whether the LDAP directory service you deploy can take the place of some of these existing directories. Also consider replicating data between the various systems; replication should prevent the problem of having different values for a data element present in several systems.

Databases

Your organization probably has several database systems (relational, hierarchical, or other) that hold interesting data. Examples include customer databases, the human resources or personnel database, student databases, and databases used to manage network resources. Another example is your in-house private branch exchange (PBX) telephone system (if you have one); its database probably contains the most up-to-date telephone number information.

In general, you will view these existing databases as gold mines when populating your directory service with an initial set of data. Do not be surprised if these systems are hard to work with, though. Often they are old, expensive to run, difficult to make changes to, and difficult to tie into other systems. Some databases may impose unreasonable restrictions on certain data fields, making the databases fairly useless as data sources. For example, a PBX database may be able to store only six characters for a login ID, whereas most other systems allow eight. These kinds of restrictions also make it difficult to propagate information from your LDAP directory back to the database if you ever want to do so.

You will need the cooperation of whoever runs these databases to get the data you want. Thus you may need to spend time lobbying to convince them that your new directory service is valuable to them and to the organization as a whole. But persist in your quest; existing databases often hold valuable information.

Files

In smaller organizations or within departments, it is common for an administrative assistant to maintain a complete phone list in a simple file or spreadsheet. If the information is maintained at the department level, it can be a good source for the most up-to-date contact information on people. Of course, the data may not be in a form that is easily pulled into a structured data system such as an LDAP-based directory service.

Applications

Many network-based or collaborative applications have a built-in database or directory that stores information about users of the application and other objects the application needs to know about. These application-specific directories tend to be proprietary, closed systems that may not even provide application programming interfaces (APIs) for accessing the data. Examples include e-mail systems that are not LDAP aware, some Web servers, and most electronic calendar systems.

One of the most important reasons to invest in an enterprise directory service based on LDAP is to replace many of these application-specific directories, or at least to coexist with them. If you are working on such a project, you need to develop a coexistence strategy or actively manage a transition to the replacement service (this may be a lengthy strategy). The existing application data stores are the best source for the application-specific data elements you plan to store in your directory service, so try to work *with* them and not around them.

Administrators

The administrators charged with caring for the computer systems, networks, and telephone systems in an organization manage a lot of interesting data. If the data these administrators manage is not available in electronic form or another manageable format, you will still want to directly enlist the aid of these administrators in setting up and maintaining the data held in your directory service.

End Users

Last, but certainly not least, end users are often the best source of data about themselves. This is especially true with personal contact information (such as home telephone numbers and addresses) and preferences.

Because you probably do not have time to ask each person for information, you should provide an easy-to-use interface to your directory service so that individuals can update their own information. Some data elements come from other databases or are maintained by administrators or applications, so be prepared to explain to end users why they can't change those data elements themselves. It is a good idea to publish information about the origin and owner of each data element held in your directory service. Such information is usually included in your data policy statement (as discussed earlier in this chapter).

Maintaining Good Relationships with Other Data Sources

As mentioned earlier, it is important to think about how the data held in your directory service relates to data stored in other sources within your organization. Many organizations today suffer from data redundancy problems in which data elements are stored in multiple, uncoordinated databases and directory systems. Your directory service could help solve this problem, and you should certainly strive to avoid making the problem worse!

The following sections provide an overview of some techniques you can use to make your directory service a success in an environment where multiple data sources hold the same data elements. [Chapter 23](#), Directory Coexistence, includes a more extensive discussion on integrating with other data sources.

Replication

If you are working with directory products that come from the same vendor or use the same protocols for replication—for example, the emerging LDAP Duplication/Replication/Update Protocol group Internet Engineering Task Force (LDUP IETF) standard—you should be able to use the built-in replication features to maintain consistent values for data elements across many servers. Third-party software also provides LDAP-to-LDAP replication for servers that support some common LDAP extensions, such as the changelog mechanism.

Replication has many potential benefits, including the possibility of spreading your directory application load across many servers, providing for redundancy in the face of server failures, and so on. See [Chapter 11](#), Replication Design, for more information on replication design considerations.

Synchronization

Synchronization is a process in which changes made in one system are propagated to another. It differs from replication in that the protocols, schema, and data formats may vary widely among the data sources involved. Synchronization is typically done frequently (every hour, day, or week), but consistency of the data is usually not as tight as with replication. Synchronization can be performed in one or both directions and between two or more data sources.

For example, if employee name changes are always handled by the Human Resources department, it may be appropriate to propagate those changes to your directory service—a one-way synchronization. If you allow telephone numbers to be changed both in the human resources database and in your directory service, you may want to set up two-way synchronization between the systems. If two-way synchronization is used, you will need to carefully consider what the outcome is if the same data element is changed in two different systems.

Synchronization tools are available from a variety of vendors, either bundled with their directory products or as standalone tools. For example, Microsoft Metadirectory Services can synchronize data between Microsoft Active Directory and many other data sources, including other LDAP servers such as Netscape Directory Server, Microsoft Exchange Server, various relational databases, Lotus Notes, and flat files.

Note

As LDAP directory service products mature, you can expect more synchronization tools to become available from the major vendors because nearly all customers are asking for them.

Good synchronization tools allow you to hook into the synchronization process and cause other actions as a result of changes to data. For example, when a person is added to a human resources database, an entry might be created in your central directory service. The same event could also be used to trigger actions outside your directory service, such as creating operating system accounts or granting access to network devices such as printers and file servers.

If you do set up synchronization between data sources, be prepared for some bumps along the way. Often a lot of tuning and some in-house software development are needed to smooth over the differences between the data sources. In many cases, the synchronization software is produced by one vendor, the directory software by another, and database software used by other data sources by yet another vendor. Clearly there is some system integration work to be done, but synchronization is a good solution if you can overcome these issues.

Batch Updates

Batch updates are really a kind of "loosely coupled" synchronization. Typically, batch updates are done less often (for example, once a month) and may involve the merging of data that comes from radically different sources. With few exceptions, the data-merging process must be developed in-house. This can be an expensive prospect because it usually takes several iterations working with real data before all the bugs are worked out.

When the authors were at the University of Michigan, a complicated C language program called munge was developed to merge data from the human resources database, student database, and a central directory service. The actual "munging" was done once or twice a month and usually took about a day. During that time no modifications could be made to the data held in the directory service, and a system administrator typically had to keep a close eye on the munge process. If anything went wrong during the data-merging process, the system administrators had to fix the data by hand or restart the entire munge process. The poor system administrators were often grumpy.

The key to successful use of batch updates is to streamline the process and make it as automatic and foolproof as possible.

Political Considerations

As the new kid on the block, your directory service may be viewed by your colleagues as something that creates more work for them rather than as the liberating tool it can become. This attitude is especially common in large organizations in which the keepers of data sources have limited resources, are focused on their own problems, or are just set in their ways.

Hopefully this is not the situation with your organization, but if it is, the best solution is to convince your peers who maintain other important data sources that your directory service will help them in some way. For example, if by working together you can eliminate four or five redundant change-of-address forms and replace the entire collection with a single change-of-address Web page, you will all be recognized as heroes.

If you are one of the people who manage non-LDAP directories or databases, you have our congratulations; you have already proven yourself to be a forward-thinking individual by reading this book. Please try to be helpful and friendly to the person trying to design and deploy the LDAP directory service (if you are not lucky enough to own that task yourself).

Team LiB

◀ PREVIOUS

NEXT ▶

Data Design Checklist

The following list summarizes the necessary steps to create a good data design:

- Develop an understanding of which applications will use your directory initially and in the future.
- Create a data policy statement.
- Create a data source inventory for your entire organization.
- Create a list of the data elements needed by each directory-enabled application.
- Characterize each data element according to
 - Format
 - Size of each value
 - Number of distinct values
 - Data ownership and restrictions
 - Consumers
 - Frequency of changes in values (dynamic versus static)
 - Range of applicability (shared versus application-specific)
 - Relationships with other data elements
- Examine the application-specific lists to locate overlap among data elements.
- Create a plan for managing the relationships between data sources.

Further Reading

Netscape Directory Server Deployment Guide: Chapter 2, How to Plan Your Directory Data.
Available on the World Wide Web at <http://enterprise.netscape.com/docs/directory>.

Looking Ahead

The first few chapters of [Part II](#) have provided an overview of the directory design process, helped you understand your directory-related needs, and helped you identify issues surrounding the data to be stored. The next few chapters will look at topics more specific to the design of LDAP directories. First up, in [Chapter 8](#), is directory schema design, a topic that is closely related to the material covered in this chapter.

Chapter 8. Schema Design

- The Purpose of a Schema
- Elements of LDAP Schemas
- Directory Schema Formats
- The Schema-Checking Process
- Schema Design Overview
- Sources of Predefined Schemas
- Defining New Schema Elements
- Documenting and Publishing Your Schemas
- Schema Maintenance and Evolution
- Schema Design Checklist
- Further Reading
- Looking Ahead

If you have some background in database systems, you are already familiar with the concept of a schema. Simply put, a *schema* is a set of rules that determines what data can be stored in a database or directory service. Schemas are important because they help maintain the integrity and quality of the data stored in a directory service. Schemas also help reduce duplication of data and provide a well-documented, predictable way for directory-enabled applications to access and modify the collection of directory objects.

In this chapter you will learn what role schemas play, why they are important, how to locate predefined schema elements, and how to design your own custom schema. First we discuss the purpose of a schema, and then we provide a detailed description of the elements that make up LDAP schemas. We continue by illustrating two formats commonly used to describe schemas, and we explain the process that servers go through to check directory entries against the schema rules. In the second part of the chapter, we describe a schema design process that you should follow for your directory service. We cover locating and selecting predefined schemas, designing and documenting your schema, and planning for schema maintenance and evolution.

The Purpose of a Schema

A *directory schema* is a set of rules that determine what can be stored in a directory service and how directory servers and clients should treat information during directory operations such as searches. Before a directory server stores a new or modified entry, it checks the entry's contents against the schema rules. Whenever directory clients or servers compare two attribute values, they consult the schema to determine what comparison algorithm to use.

[Chapter 7](#), Data Design, covered the importance of combining redundant data elements (that is, those needed by more than one application) into as few data elements as possible. One of the main purposes for a schema is to ensure that poorly behaved applications play by the rules and do not store redundant data in the directory service. Imagine the consequences if every directory-enabled application stored a person's name in a different directory attribute. The result would be wasted storage space and values that should be the same but are different, and ultimately a lot of confusion on the part of applications and end users.

Schemas can also be used to impose constraints on the size, range, and format of data values stored in the directory. For example, according to the Internet mail standards, e-mail address values should use a restricted set of characters and should conform to a specific format (`addr@domain`). In many cases, schema rules impose simple restrictions such as "this value must be an integer." Ensuring that the data values in the directory service conform to a collection of simple rules increases the quality of the data.

Finally, directory schemas can help slow the effects of directory entropy. Although they are not a substitute for appropriate access control rules (as described in [Chapter 12](#), Privacy and Security Design), schema rules do help a bit in preventing chaos within your directory service.

Suppose that you allow end users to modify directory entries, but there is no schema enforcement; you should not be surprised when your directory servers become overburdened with a lot of information that does not belong there. Some users may store a lot of information that is of interest to only themselves, some may store very large values, and others may be silly or even malicious. For example, somebody might try to use an LDAP-based directory as a file system backup service for his PC, although most people would agree that this is inappropriate and should be discouraged!

Tip

If you have a lot of experience with traditional databases, you probably can't imagine a data store that does not impose schema rules. However, keep in mind that many users of your directory service may be novices with no directory service, database, or schema experience. Part of your job as the directory architect is to educate your users and developers and help them understand that schemas improve the directory service by increasing its reliability and the quality of the data.

Elements of LDAP Schemas

In LDAP-based directories, a schema consists of attribute types, attribute syntaxes, matching rules, and object classes. These terms were introduced in [Chapter 2](#), Introduction to LDAP, when we described the LDAP information model, but they are described in greater detail here.

Attributes

Recall that directory entries contain a collection of attribute types and values. *Attribute types* (or simply *attributes*) hold specific data elements such as a name, business phone number, or printer's rated speed in pages per minute. In LDAP, the definition of an attribute type includes the following:

- A name that uniquely identifies the attribute type
- An object identifier (OID) that also uniquely identifies the attribute
- A textual description
- An associated attribute syntax
- A set of matching rules that govern comparisons and searches
- A usage indicator (whether for applications or for operation of the directory service itself)
- An indication of whether the attribute is multivalued or single-valued
- An indication of whether the attribute can be modified by regular applications
- Restrictions on the range or size of the values that may be stored in the attribute

Attribute names are usually fairly short and somewhat cryptic, although they do not have to be. Attribute names have the following properties:

- They are not case sensitive; for example, `cn` and `CN` both refer to the same attribute.
- Characters used within them are limited to ASCII letters, digits, and the hyphen character; and they must begin with a letter.
- They must be unique across the entire directory service because LDAP applications generally refer to attributes using their names.

Examples of valid attribute names include `cn`, `telephoneNumber`, `postalAddress`, `one-way`, `faxPhone2`, and `pagesPerMinute`. Some examples of invalid attribute names are `last#`, `2for2`, `my.boss`, and `favorite_drink`.

Some standard attributes have historically been known by both a longer name and a shorter name (for example, `commonName` and `cn`), but in most cases the shorter name is the standard that LDAP clients and servers use. Confusion about which attribute name to use caused LDAP client and server interoperability problems in the past. Some implementations, such as Netscape Directory Server, support longer names as aliases or synonyms for the shorter, standard names to increase compatibility with older LDAP applications.

An attribute's OID is a unique numerical identifier usually written as a sequence of integers separated by dots. For example, the OID for the `postalAddress` attribute is `2.5.4.16`. OIDs are required by X.500 directory implementations because they are used within the X.500 family of protocols to identify attribute types. Although LDAP clients and servers can use OIDs in place of attribute names, names are almost always used in practice because they are much easier to work with than OIDs.

In an entry stored in a directory server database, attributes have one or more *attribute values* associated with them. These values correspond to the data element values discussed in [Chapter 7](#), Data Design. The attribute type provides the semantic meaning for a set of values. For example, the value `12` by itself is not very meaningful; but if you know that it is a value for the `pagesPerMinute` attribute, you can begin to imagine several uses for the value. The definition of the attribute type includes an indication of whether the type is allowed to hold only one value (single-valued) or several values (multivalued). Most attribute types are multivalued, which is the default.

The *attribute usage indicator* is usually omitted from the definition of an attribute type because it defaults to user applications, which means that the attribute is of general purpose and can be used by any directory application. The other type of usage is operational (note that X.500 systems support several subcategories of operational usage). *Operational attributes* are used by the directory service itself for administrative or system-related purposes and are usually maintained by the directory servers themselves. These attributes are not visible to directory clients unless specifically requested, and a client typically cannot modify them. [Table 8.1](#) shows some examples of operational attributes.

Clients can retrieve operational attributes by explicitly asking for them by name. Work is also under way in the Internet Engineering Task Force (IETF) to define an LDAP control that allows a client to request the return of all operational attributes.

Tip

Once the forthcoming LDAPv3 control to retrieve all operational attributes has been widely implemented, it will be easy to retrieve all the attributes of an entry using one LDAP search command. In the meantime, you can perform an LDAP search command and ask for * (which stands for all regular attributes) plus a list of desired operational attributes. If you want to retrieve all the operational attributes that are present, you must first generate a list of the ones supported by your server. You can do this by consulting the server's documentation or by reading the server's schema over LDAP and using a scripting language such as awk or Perl to extract the attribute types that are marked as operational. See the section titled The LDAPv3 Schema Format later in this chapter for more information about how LDAP servers store schemas and how they can be retrieved over LDAP.

An Attribute Type Example

People generally like names and are pretty good at remembering them. Almost all real-world objects have a name of some kind, and we have already mentioned the `cn` (common name) attribute several times. Sometimes, though, it is helpful to associate longer text with an LDAP entry that describes the entry in more detail. LDAP defines a handy attribute called `description` for just that purpose. [Listing 8.1](#) shows its definition.

Table 8.1. Examples of Operational Attributes

Attribute	Where Found	Description
<code>modifyTimeStamp</code>	Any entry	Date/time an entry was last modified
<code>modifiersName</code>	Any entry	Distinguished name (DN) of the entry that made the last modification
<code>namingContexts</code>	LDAPv3 root DSE	Partitions of the directory held in a server
<code>supportedLDAPVersion</code>	LDAPv3 root DSE	Versions of the LDAP protocol supported by the server
<code>aci</code>	Any entry	Access control information (Netscape-specific)

Listing 8.1 The `description` Attribute

```
( 2.5.4.13 NAME 'description' EQUALITY caseIgnoreMatch SUBSTR caseIgnoreSubstringsMatch  
→ SYNTAX 1.3.6.1.4.1.1466.115.121.1.15{1024} )
```

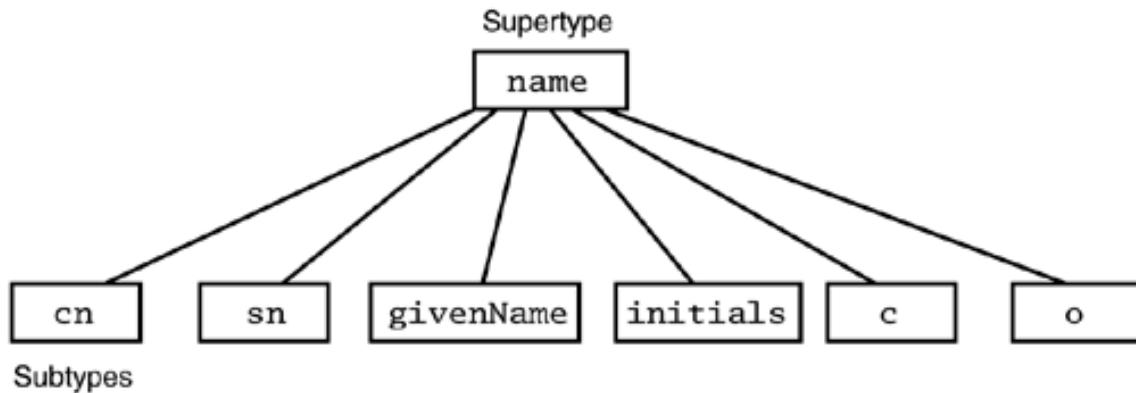
We will explain this format in more detail later in this chapter when we talk about using the LDAPv3 format from RFC 2252 to describe schemas. Translated into English, this definition says that the attribute named `description` is a string that can hold up to 1,024 characters. This attribute type uses the `caseIgnore` family of matching rules; therefore, the case of letters and leading and trailing space characters are ignored in comparisons of values. The OID of this attribute type is `2.5.4.13`, a sequence assigned by the X.500 standards committee.

Attribute Hierarchies

Some LDAP implementations, notably those closely aligned with the most recent X.500 standards, support *attribute subtypes*. Subtypes define attribute hierarchies in which general attribute types can be used to construct more specific types. For example, X.500 defines the `cn` attribute as a subtype of an attribute called `name`. Similarly, the `sn` (surname) attribute is also a subtype of `name`. The attribute called `name` is said to be the *supertype* of `cn`, and `cn` is said to be *derived from* `name`.

Attribute subtypes inherit the characteristics of their supertype. In addition, a supertype can be used to simplify searches and retrieval of attributes derived from it. For example, because both the `cn` and `sn` attributes are subtypes of the attribute `name`, a search request that asks for all values of the `name` attribute type will get back all values of `name`, `cn`, `sn`, and any other subtypes of `name`. [Figure 8.1](#) shows a portion of the attribute hierarchy for the `name` attribute type. Note that several other attribute types not shown in [Figure 8.1](#) are also subtypes of `name` (it is a popular supertype!).

Figure 8.1. The Attribute Hierarchy for the `name` Attribute Type



Attribute subtypes are an interesting but potentially confusing feature. Most LDAP implementations do not even support attribute subtypes. Even if you like the idea of subtypes, think twice before you design a schema that relies on them.

Attribute Syntaxes

As mentioned earlier, each attribute type has an associated *attribute syntax* that specifies exactly how data values are represented. Just as each attribute type has an OID, each syntax has one too. Although in LDAP most data values are carried as text strings (unlike in X.500's Directory Access Protocol, or DAP), additional rules may apply to the string. For example, dollar signs (\$) are used to separate the pieces of information in a postal mailing address in LDAP. [Table 8.2](#) shows some standard syntaxes.

The great majority of attribute types have the `DirectoryString` syntax. Note that some LDAP server software packages, such as Netscape Directory Server, allow additional syntaxes to be defined through the use of software plug-ins, but generally the set of supported syntaxes is limited by the directory server software.

Matching Rules

Another piece of the puzzle is that LDAP clients and servers must use special rules when comparing the values of different attribute types. For example, when comparing filenames on a Unix system, the case of the letters must be taken into account; on a Microsoft Windows system, however, filenames aren't case sensitive (for example, `Windows` would match `WINDOWS`).

The rules used for making attribute value comparisons are called *matching rules*. [Table 8.3](#) shows some standard matching rules.

Some directory service implementations still use the term *syntax* to refer to both the data representation itself and the matching rules because in the early X.500 standards no distinction was made. Also note that most directory server implementations support only a predetermined set of matching rules. Adding support for new matching rules usually requires code to be written, or it may be impossible to do at all.

Table 8.2. Standard Syntaxes

Syntax	OID	Description
Binary	1.3.6.1.4.1.1466.115.121.1.5	Data encoded by Basic Encoding Rules (BER) or Distinguished Encoding Rules (DER)—for example, an X.509v3 certificate
Boolean	1.3.6.1.4.1.1466.115.121.1.7	TRUE or FALSE
CountryString	1.3.6.1.4.1.1466.115.121.1.11	Text string that holds a two-digit country code—for example, US
DirectoryString	1.3.6.1.4.1.1466.115.121.1.15	International Organization for Standardization (ISO) 10646 text string in UCS Transformation Format 8 (UTF-8) format (ISO 10646 is a superset of Unicode)
DN	1.3.6.1.4.1.1466.115.121.1.12	Distinguished name (pointer to another entry) in string (RFC 2253) format
GeneralizedTime	1.3.6.1.4.1.1466.115.121.1.24	Date and time in X.208 format—for example, 20010911134600Z
IA5String	1.3.6.1.4.1.1466.115.121.1.26	ASCII text string
INTEGER	1.3.6.1.4.1.1466.115.121.1.27	Integer numeric value, represented as a string
OctetString	1.3.6.1.4.1.1466.115.121.1.40	Arbitrary binary data (an array of 8-bit bytes)
PostalAddress	1.3.6.1.4.1.1466.115.121.1.41	Multiline postal address (lines separated by "\$" characters)
PrintableString	1.3.6.1.4.1.1466.115.121.1.44	Text string; characters are from a restricted set that includes ASCII letters, digits, and a few punctuation marks (such as a comma)
TelephoneNumber	1.3.6.1.4.1.1466.115.121.1.50	Telephone number; should be in E.123 format—for example, +1 800 555-1212. E.123 is a standard format in which all telephone numbers begin with a plus sign that is followed by the complete, internationally dialable number, which begins with a country code (the country code for all North American countries is 1)
URI	1.3.6.1.4.1.4401.1.1.1	Uniform Resource Identifier—for example, a Uniform Resource Locator (URL)

Table 8.3. Standard Matching Rules

Matching Rule	Description
<code>booleanMatch</code>	Comparisons follow the rules for comparing Boolean values; only equality match is supported.
<code>caseIgnoreMatch</code>	Case of letters and leading, trailing, and multiple spaces are ignored during comparisons.
<code>caseExactMatch</code>	Case of letters is significant during comparisons; leading, trailing, and multiple spaces are ignored.
<code>distinguishedNameMatch</code>	Comparisons follow special rules for comparing DNs (number of relative distinguished names [RDNs] must be the same, each RDN must have the same number of values, and each type and value must match).
<code>integerMatch</code>	Comparisons follow the rules for comparing integers.
<code>octetStringMatch</code>	A binary (byte-by-byte) comparison is performed on the attribute values.
<code>telephoneNumberMatch</code>	This rule is like <code>caseIgnoreMatch</code> , except hyphen and space characters are also ignored during comparisons.

Object Classes

In LDAP, *object classes* are used to group related information. Typically, an object class models a real-world object such as a person, printer, or network device (although this is not required). Each directory entry belongs to one or more object classes. The names of the object classes to which an entry belongs are always listed as values for a special multivalued attribute called `objectclass`. The set of object classes associated with an entry serves the following needs:

- It determines which attribute types *must* be included in the entry.
- It determines which attribute types *may* be included in the entry.
- It provides a convenient way for directory clients to retrieve a subset of entries during search operations.

For example, an object class designed to hold information about people might require that a name be present and allow specific optional attributes to be included to hold personal and contact information about a person. If the object class is called `person`, a directory client could provide a way to search for only people entries by adding a component such as `objectclass=person` to the LDAP search filter it constructs.

The definition of an LDAP object class includes all the following pieces of information:

- A name that uniquely identifies the class
- A textual description
- An OID that also uniquely identifies the class
- A set of mandatory attribute types
- A set of allowed attribute types
- A kind (structural, auxiliary, or abstract)

The *name* of an object class is usually mnemonic and easily understood by humans, but good directory clients try to hide these names from end users. A name is the most common way for humans, clients, and servers to refer to the object class. As for attribute types, both the name and OID must be unique throughout the directory service. Some examples of object class names include `person`, `printer`, `groupOfNames`, and `applicationEntity`.

The set of *mandatory* (required) attribute types is usually fairly short or even empty. The set of *allowed*

(optional) attribute types is often quite long. It is the job of each directory server to enforce attribute type restrictions of an object class when an entry is added to the directory or modified in any way. The one exception is that read-only directory replicas do not need to check for schema violations; they can simply rely on the updatable server that supplies them to do the checking.

The *kind* of object class indicates how the class is used. *Structural classes*, for example, describe the basic aspects of an object. In some directory server implementations, structural classes can be used to place restrictions on where an entry can be stored within the directory information tree (DIT). Most object classes are structural classes, and all entries should belong to exactly one structural object class. Examples of structural object classes include `person` and `groupOfNames`.

Auxiliary classes place no restrictions on where an entry may be stored, and they are used to add a set of related attributes to an entry that already belongs to a structural class. As many auxiliary classes as desired can be "mixed in" to an entry; thus, auxiliary classes are sometimes called *mix-in classes*. Examples include `simpleSecurityObject`, `mailRecipient`, and `cacheObject`.

Abstract classes, the third and last kind of object class, are rare and are used only for classes needed to support LDAP's basic information model. Two examples of abstract classes are `top` (discussed later in this chapter in the section titled Object Class Inheritance) and `alias` (which is used to create entry aliases, as described in [Chapter 2](#), Introduction to LDAP).

Some LDAP directory server implementations, including Netscape Directory Server, do not rigidly enforce restrictions based on the kind of object classes used in a schema (for example, they do not prevent you from creating an entry with two structural classes). However, the distinctions implied by the different kinds of classes are useful when you're designing your own schema.

An Object Class Example

One of the most common uses of a directory system is to store information about people. To model these real-world "objects," we might define an object class called `person`. The information about people that is typically stored in a directory service includes name, contact information, department, job title or position, and any information needed to support software applications. For each piece of information, we would define an attribute type. For naming entries and to help directory clients, we may want to require that all `person` entries include a name. Other attributes would be allowed in an entry but would be optional.

Not surprisingly, the X.500 designers had many of these same thoughts and defined a standard `person` object class that was adopted by the LDAP designers as well. It is shown in [Figure 8.2](#). The `person` class is a structural object class used for entries that represent people. Some name-related attributes are mandatory; all others are optional.

Figure 8.2. The Standard LDAP `person` Object Class

<u>Represents:</u>	<u>Requires:</u>	<u>Allows:</u>
 (A person)	<code>sn: Jensen</code> <code>cn: Babs Jensen</code> <code>objectclass: top</code> <code> person</code> (Naming information)	<code>description: directory manager</code> <code>telephoneNumber: +1 650 555-1234</code> <code>userPassword: secret</code> <code>seeAlso: cn=Fred Jensen</code> (Descriptive and contact information)

The Presence of Multiple-Object Classes

There is no limit to the number of different object classes a single entry can belong to, as long as only one of the classes is a structural class. If an entry belongs to more than one object class, we determine the set of attributes allowed in the entry simply by computing the union or aggregate of the attributes listed in all the object classes. Similarly, the set of mandatory attributes is the union of all the required attributes listed in all the object classes. This means that if an attribute is required in one of the object classes to which an entry belongs, it is required in the entry. The entire attribute namespace is a flat space with no hierarchy or other structure within the names.

For example, suppose that the `printer` and `networkDevice` object classes are defined as shown in [Listing 8.2](#).

(using the LDAPv3 schema format described in more detail later in this chapter; of course, `OIDn` and `OIDp` would need to be replaced with real, numeric OIDs).

Listing 8.2 `printer` and `networkDevice` Object Classes

```
( OIDp NAME 'printer' SUP top STRUCTURAL MUST cn MAY ( description $ pagesPerMinute $  
languages  
↳ ) )  
  
( OIDn NAME 'networkDevice' SUP top AUXILLIARY MUST ipaddress MAY ( cn $ connectionSpeed ) )
```

Given these definitions, the following attributes must be present in any entry that belongs to both classes:

```
cn  
ipaddress
```

The following attributes are optional:

```
connectionSpeed  
description  
languages  
pagesPerMinute
```

Notice that because the `cn` attribute is required in the `printer` class, a `cn` value must always be present in entries that belong to both classes (even though it is merely optional in the `networkDevice` class).

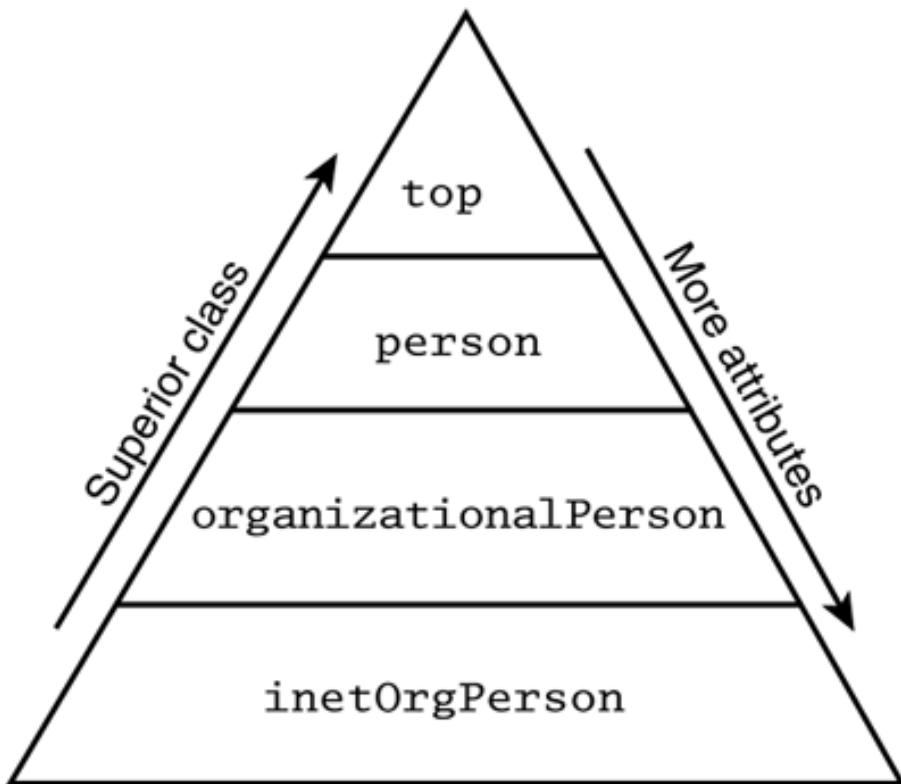
Tip

Even if an attribute type is listed in more than one object class, an entry that belongs to both classes has only one instance of that attribute type. That is, the attribute type space is flat and unrelated to the way in which attributes are assigned to object classes. In the example in [Listing 8.2](#), it is impossible to distinguish between the `cn` attribute included in the `printer` class and the `cn` attribute included in the `networkDevice` class. They are the same attribute, and there is only one set of `cn` values in an entry that belongs to both object classes.

Object Class Inheritance

One object class can be derived from another, in which case it inherits some characteristics of the other class. This is sometimes called *subclassing*, or *object class inheritance*. [Figure 8.3](#) shows an example.

Figure 8.3. Object Class Inheritance



The `inetOrgPerson` object class defined by the directory designers at Netscape is also supported by most other implementations. It extends the `person` class and includes additional attribute types to store data elements commonly used on the Internet and within organizations. It is actually derived from an intermediate class called `organizationalPerson`, which is derived from the `person` class. Therefore, `inetOrgPerson` entries require all the attributes required of `organizationalPerson` entries (and therefore of `person` entries as well). Similarly, `inetOrgPerson` entries are allowed to include any of the optional attributes from the superior classes.

In general, the class from which another class inherits some of its characteristics is called the *superior class*, or *superclass* (that is, `organizationalPerson` is the superior class of `inetOrgPerson`). When one class is derived from another, it inherits the set of required attribute types, the set of optional attribute types, and the kind of object class from its superior. Also, an exception to the rule mentioned previously (each entry may belong to exactly one structural class) is that entries may belong to two or more structural classes if the classes are derived from each other.

All structural object classes are ultimately derived from one special abstract object class called `top`. The definition of the `top` class consists of a single mandatory attribute called `objectclass`, which ensures that all LDAP entries contain at least one value for `objectclass`.

Be careful when comparing LDAP's object class inheritance with the more sophisticated class inheritance supported by many object-oriented programming languages. For example, LDAP implementations do not support multiple inheritance, in which a single object class is derived from two or more superior classes. As we have seen, however, LDAP does support aggregation because an entry can be a member of more than one object class. Also there is no way to "override" any of the schema rules defined by the superior class. For example, if the superior class requires the `cn` attribute to be present, then it must always be present in all subclasses.

The LDAPv3 `extensibleObject` Object Class

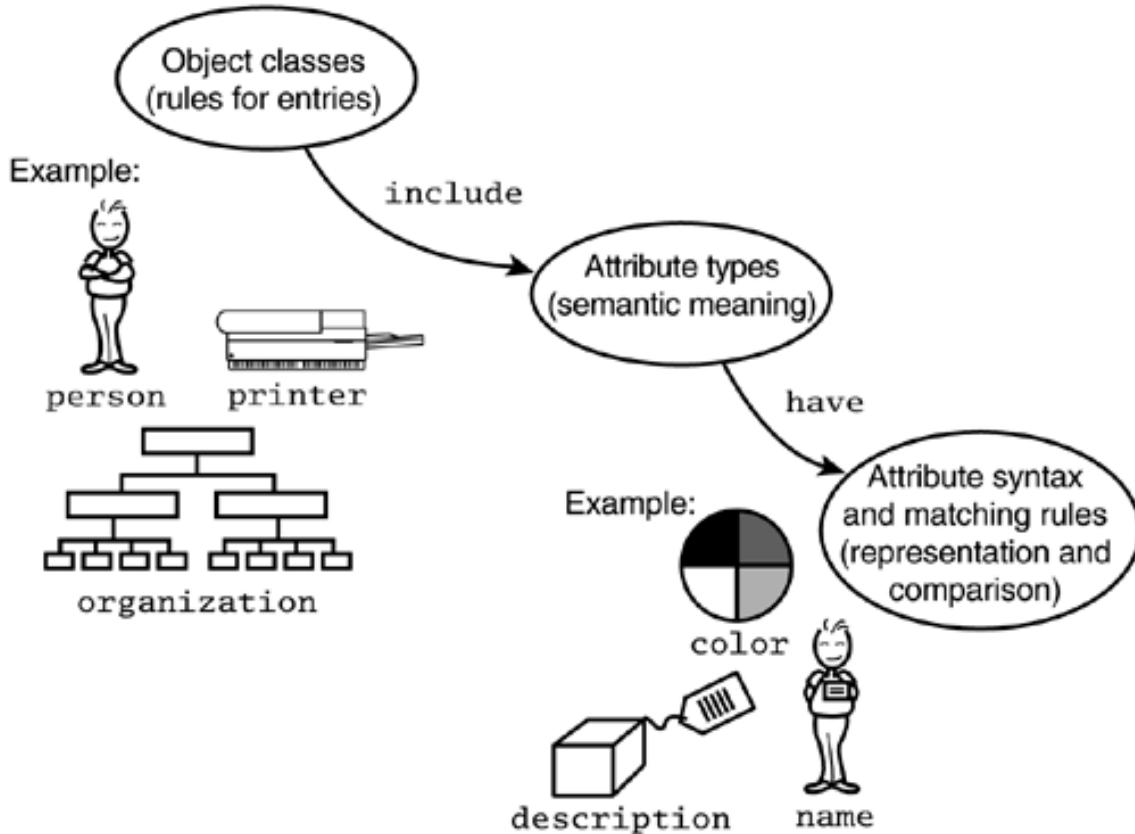
LDAPv3 defines a special object class called `extensibleObject` that is supported by all compliant implementations. This object class allows an open-ended set of attribute types—that is, any valid attribute is allowed in `extensibleObject` entries. Note that the attribute types must be defined in the schema; using `extensibleObject` only eliminates the requirement that the correct object classes be included in an entry.

In general, this class is rarely used, for the same reasons that most administrators do not entirely disable schema checking: It may give users and applications too much freedom to store unstructured data in the directory. This class can be useful when you want to create entries that are largely free from schema constraints, especially if it is used by only a small set of applications that you control.

Schema Element Summary

In summary, LDAP schemas are made up of attribute types, each of which has a syntax and a set of matching rules. Attributes are used to hold specific kinds of data, and they are grouped into logical units through the definition of object classes. It is the job of each directory server to enforce the restrictions imposed by the schemas when entries are created or modified. [Figure 8.4](#) shows all the elements of LDAP schemas and how they relate to one another.

Figure 8.4. Directory Schema Elements



Tip

The schemas used by a directory service typically include dozens of object classes and hundreds of attribute types. Thankfully, the number of attribute syntaxes is usually small (a dozen or fewer), and the number of widely used object classes is also limited. Do not be overwhelmed by the number of available schemas. Focus on your needs and use an incremental approach as you select schema elements for your service.

Directory Schema Formats

When reading vendor documentation or standards documents, or when working with directory service software, you will encounter several different formats used to describe a directory schema. It is useful to be able to read and understand some popular schema formats. Two schema definition formats are described in this section.

The first schema format we describe, the LDAPv3 schema format, is the most popular one, and it is also the one used throughout this book. The other format we describe is the ASN.1 format, which is included to help you read schema documentation that happens to use it. In the first edition of this book, we described a third schema format: the slapd.conf schema format. That format is generally used only by older directory software such as Netscape Directory Server prior to release 6 and by OpenLDAP 1.x, so we do not describe it here.

The LDAPv3 Schema Format

Version 3 of the LDAP protocol requires that directory servers publish their supported schemas through LDAP itself, thereby allowing directory client applications to retrieve the schema programmatically and adapt their behavior to it. For example, a client that allows administrators to add new directory entries can query a directory server to find the complete list of structural object classes it supports and present that list to the administrator. The LDAPv3 schema format is described in RFC 2252, *LDAPv3 Attribute Syntax Definitions*.

Clients locate a set of schemas by reading the values of the `subschemaSubentry` operational attribute that is located in every directory entry. Values of the `subschemaSubentry` attribute are LDAP DNs that point to entries that belong to the `subschema` object class. Such entries are called *subschema entries*. Netscape Directory Server, for example, supports one global set of schemas for the server, which is stored in the subschema entry named `cn=schema`. Therefore every entry has a `subschemaSubentry` operational attribute that points to the `cn=schema` entry; that is, each has the value `cn=schema`.

The `subschema` object class allows the following seven regular attribute types to be present:

1. `attributeTypes`
2. `objectClasses`
3. `matchingRules`
4. `matchingRuleUse`
5. `dITStructureRules`
6. `nameForms`
7. `dITContentRules`

In addition, the following operational attribute type may be present in subschema subentries:

8. `ldapSyntaxes`

We discuss only the first two attributes (`attributeTypes` and `objectClasses`) because they are by far the most important and most widely used. For information on the rest, refer to the *LDAPv3 Attribute Syntax Definitions* document (RFC 2252).

The LDAPv3 specification encourages servers to allow their schemas to be modified over LDAP. If an implementation supports it, a directory client can modify the `attributeTypes` and `objectClasses` attributes of a subschema entry (subject to access control restrictions, of course).

[Listing 8.3](#) shows how to use the Netscape ldapsearch command-line utility to retrieve a set of schemas.

The first `ldapsearch` command finds the DN of the subschema entry that governs Babs Jensen's entry (`cn=schema`), and the second `ldapsearch` command retrieves the subschema entry itself. The entry data returned by the second command is truncated to save space.

Listing 8.3 Retrieving an LDAP Server's Schema

```
ldapsearch -h ldap.example.com. -b "dc=example,dc=com" -s sub cn="Babs Jensen" uid  
↳ subschemaSubentry  
  
version: 1  
  
dn: uid=bjensen, ou=People, dc=example,dc=com  
  
uid: bjensen  
  
subschemaSubentry: cn=schema  
  
  
ldapsearch -h ldap.example.com -b "cn=schema" -s base "objectclass=subschema"  
↳ objectClasses attributeTypes matchingRules matchingRuleUse dITStructureRules nameForms  
↳ dITContentRules ldapSyntaxes  
  
version: 1  
  
dn: cn=schema  
  
objectClasses: ( 2.5.6.0 NAME 'top' DESC 'Standard LDAP objectclass' ABSTRACT  
    MUST objectClass X-ORIGIN 'RFC 2256' )  
  
objectClasses: ( 2.5.6.1 NAME 'alias' DESC 'Standard LDAP objectclass' SUP top  
    ABSTRACT MUST aliasedObjectName X-ORIGIN 'RFC 2256' )  
  
objectClasses: ( 1.3.6.1.4.1.1466.101.120.111 NAME 'extensibleObject' DESC 'LD  
    APv3 extensible object' SUP top AUXILIARY X-ORIGIN 'RFC 2252' )  
  
objectClasses: ( 2.5.6.2 NAME 'country' DESC 'Standard LDAP objectclass' SUP t  
    op STRUCTURAL MUST c MAY ( searchGuide $ description ) X-ORIGIN 'RFC 2256' )  
  
objectClasses: ( 2.5.6.3 NAME 'locality' DESC 'Standard LDAP attribute type' S  
    UP top STRUCTURAL MAY ( description $ L $ searchGuide $ seeAlso $ st $ stree  
    t ) X-ORIGIN 'RFC 2256' )  
  
...
```

LDAPv3 Attribute Type Definitions

The `attributeTypes` attribute within a subschema entry of an LDAPv3-compliant server contains one value for each attribute supported by the server. Each value is as shown in [Listing 8.4](#), broken up into multiple lines for clarity and with optional items in brackets.

Listing 8.4 The General Form of LDAPv3 `attributeTypes` Values

```
( ATOID NAME ATNAME [ DESC ATDESC ] [ OBSOLETE ] [ SUP SUPOID ] [ EQUALITY EQMATCHOID
  ↪ ] [ ORDERING ORDMATCHOID ] [ SUBSTR SUBMATCHOID ] [ SYNTAX SYNOID ] [ SINGLE-VALUE ] [
  ↪ COLLECTIVE ] [ NO-USER-MODIFICATION ] [ USAGE ATUSAGE ] )
```

`ATOID` is the object identifier for the attribute; `ATNAME` is the attribute name (which can be a list of names); `ATDESC` is a text description of the attribute; `SUPOID` is the object identifier of the type it is derived from (if any); `SYNOID` is the object identifier of the attribute's syntax; and `EQMATCHOID`, `ORDMATCHOID`, and `SUBMATCHOID` are the object identifiers for the matching rules associated with the attribute type.

By default, attribute types are assumed to be multivalued unless the `SINGLE-VALUE` phrase is present. Attribute types that are marked with the `OBSOLETE` keyword cannot be used within new entries. Attribute types are shared among a set of entries if the `COLLECTIVE` keyword is present. If the `NO-USER-MODIFICATION` keyword is present, a normal user of the directory cannot modify the attribute type.

`ATUSAGE` indicates how attributes with this type are used in the directory service and must be one of the values shown in [Table 8.4](#). If this parameter is omitted, the default value `userApplications` is in effect. All parameters that are strings should be enclosed in single quotes.

Each attribute type is represented by one value for the `attributeTypes` attribute within a subschema entry. [Listing 8.5](#) shows how the standard `cn` and `name` attribute types would be described in the LDAPv3 schema format. The first attribute type has the OID `2.5.4.3` and two names (`cn` and `commonName`). It is a subtype of the `name` attribute type and therefore inherits other qualities from the attribute type called `name`. The second attribute type has the OID `2.5.4.41` and is called `name`. It has syntax `DirectoryString` with a length limit of 32,768 characters. The `caseIgnore` family of matching rules is specified (which means that the case of letters will be ignored when values are being compared).

Table 8.4. Attribute Usage Indicators

ATUSAGE Value	Description
<code>userApplications</code>	A regular attribute used by directory applications
<code>directoryOperation</code>	An operational attribute shared among replicas
<code>dSAOperation</code>	A server-specific operational attribute
<code>distributedOperation</code>	An operational attribute that supports distributed operations

Listing 8.5 The `cn` and `name` Attribute Types in the LDAPv3 Schema Format

```
( 2.5.4.3 NAME ( 'cn' 'commonName' ) DESC 'the name an object is commonly known by' SUP
  ↪ name )
( 2.5.4.41 NAME 'name' DESC 'the name supertype' EQUALITY caseIgnoreMatch SUBSTR
  ↪ caseIgnoreSubstringsMatch SYNTAX 1.3.6.1.4.1.1466.115.121.1.15{32768} )
```

[Listing 8.6](#) shows some additional examples of attribute type definitions in LDAPv3 format. Notice that the last two attribute type definitions include the additional keyword **X-ORIGIN**. The LDAPv3 format allows extended keywords to be used in attribute type and object class definitions; these must begin with "X-". The Netscape products use **X-ORIGIN** to indicate the origin of a schema definition.

Listing 8.6 Additional Attribute Type Examples in the LDAPv3 Schema Format

```
( 1.3.6.1.4.1.250.1.57 NAME 'labeledURI' DESC 'Uniform Resource Identifier with optional  
➥ label' SYNTAX 1.3.6.1.4.1.1466.1.15.121.1.26 )  
  
( 2.5.4.34 NAME 'seeAlso' DESC 'DN pointer to a related entry' SUP distinguishedName  
➥ SYNTAX 1.3.6.1.4.1.1466.115.121.1.12 )  
  
( 1.3.6.1.4.1.1466.101.120.15 NAME 'supportedLDAPVersion' DESC 'root DSE operational  
➥ attribute type' SYNTAX 1.3.6.1.4.1.1466.115.121.1.27 X-ORIGIN 'RFC 2252' )  
  
( 2.16.840.1.113730.3.1.559 NAME 'nsMaxMailLists' DESC 'Limit for number of mailing lists  
➥ in a name space' SYNTAX 1.3.6.1.4.1.1466.115.121.1.27 SINGLE-VALUE X-ORIGIN 'Netscape  
➥ Delegated Administrator' )
```

LDAPv3 Object Class Definitions

The **objectClasses** attribute type within a subschema entry of an LDAPv3-compliant server contains one value for each object class supported by the server. [Listing 8.7](#) shows the general form for each value, broken into multiple lines for clarity and with optional items in brackets.

Listing 8.7 The General Form of LDAPv3 `objectClasses` Values

```
( OCID NAME OCNAME [ DESC OCDESC ] [ OBSOLETE ] [ SUP SUPOID ] [ OCKIND ] [ MUST REQATSET  
➥ ] [ MAY ALLOWATSET ] )
```

OCID is the object identifier for the class; **OCNAME** is the name of the object class; **OCDESC** is a text description of the class; **SUPOID** is the object identifier of the class it is derived from; **OCKIND** is **ABSTRACT**, **STRUCTURAL**, or **AUXILIARY** (the default is **STRUCTURAL**); and **REQATSET** and **ALLOWATSET** are the names or OIDs of the required and allowed attribute types, respectively. The attribute sets are enclosed in parentheses, and each attribute type name is separated from the next with a dollar sign. Attribute types that are marked with the **OBSOLETE** keyword cannot be used within new entries. All parameters that are strings should be enclosed in single quotes.

[Listing 8.8](#) shows how the standard **person** object class would appear in LDAPv3 schema format. The object identifier is **2.5.6.6** (a value assigned by the X.500 standards committee), the name of the class is **person**, the class is derived from the **top** abstract class, and the required and optional attribute types are as shown in the **MUST** and **MAY** lists. This is a structural class that is used as the primary object class for an entry.

Listing 8.8 The Standard `person` Object Class in LDAPv3 Format

```
( 2.5.6.6 NAME 'person' DESC 'Standard Person Object Class' SUP 'top' STRUCTURAL MUST (  
➥ objectclass $ sn $ cn ) MAY ( description $ seeAlso $ telephoneNumber $ userPassword ) )
```

[Listing 8.9](#) shows two additional structural object classes: the `inetOrgPerson` object class from RFC 2798 and the `organizationalPerson` class from which it is derived (notice that `organizationalPerson` has `person` as its superior, so an entry with object class `inetOrgPerson` must obey the `person` rules as well).

Listing 8.9 The `inetOrgPerson` and `organizationalPerson` Object Classes in LDAPv3 Format

```
( 2.16.840.1.113730.3.2.2 NAME 'inetOrgPerson' DESC 'Extended organizational person
➥ object class' SUP organizationalPerson STRUCTURAL MAY ( audio $ businessCategory $
➥ carLicense $ departmentNumber $ displayName $ employeeNumber $ employeeType $ givenName
➥ $
➥ homePhone $ homePostalAddress $ initials $ jpegPhoto $ labeledURI $ mail $ manager $
➥ mobile $ o $ pager $ photo $ roomNumber $ secretary $ uid $ userCertificate $
➥ x500uniqueIdentifier $ preferredLanguage $ userSMIMECertificate $ userPKCS12 ) )

( 2.5.6.7 NAME 'organizationalPerson' DESC 'Standard organizational person objectclass'
➥ SUP person STRUCTURAL MAY ( destinationIndicator $ facsimileTelephoneNumber $
➥ internationaliSDNNumber $ L $ ou $ physicalDeliveryOfficeName $ postOfficeBox $
➥ postalAddress $ postalCode $ preferredDeliveryMethod $ registeredAddress $ st $ street $
➥ teletexTerminalIdentifier $ telexNumber $ title $ x121Address ) )
```

Finally, [Listing 8.10](#) shows an example of a simple auxiliary object class. The `labeledURIOBJECT` class is defined in RFC 2079 and can be added to any entry to allow a `labeledURI` attribute to be present in the entry, typically to allow a URL to be stored.

Listing 8.10 The `labeledURIOBJECT` Class in LDAPv3 Format

```
( 1.3.6.1.4.1.250.3.15 NAME 'labeledURIOBJECT' DESC 'object that contains the URI
➥ attribute type' SUP top AUXILIARY MAY labeledURI X-ORIGIN 'RFC 2079' )
```

Although the LDAPv3 schema format is optimized for programmatic access rather than human readability, if you insert newline characters when composing a definition, it is fairly easy to read.

Tip

We recommend that you use the LDAPv3 schema format when designing your own directory's schema. The LDAPv3 format is used in most new RFCs and other standards documents, and it can be used directly as server configuration information for many of the most popular LDAP server implementations.

The ASN.1 Schema Format

Some of the LDAP and X.500 standards documents use Abstract Syntax Notation One (ASN.1) to document

the schema they use, the elements of their protocols, and many other details. Although a complete discussion of ASN.1 is beyond the scope of this book, it is useful to learn to read and understand ASN.1-based schema definitions. If you are not interested in understanding ASN.1-based schema definitions, feel free to skip this entire section. You can always come back and read the material here if you run into ASN.1 when you're searching for schema elements to use for your own directory service.

At its most basic level, ASN.1 is a language that can be used to describe abstract types and values. Like most declarative programming languages, ASN.1 defines a series of simple types and allows them to be combined in various ways to build more complex descriptions. An ASN.1 definition, described in the next section, generally consists of interrelated definitions called *ASN.1 macros*.

ASN.1 Attribute Type Definitions

ASN.1 attribute type definitions are generally of the form shown in [Listing 8.11](#) (all the items enclosed in square brackets are optional).

Listing 8.11 The General Form of an Attribute Type Definition in ASN.1

```
ATNAME ATTRIBUTE ::= {  
    WITH SYNTAX SYNTAXNAME { BOUNDS }  
    EQUALITY MATCHING RULE EQMATCHINGRULE  
    [ ORDERING MATCHING RULE ORDMATCHINGRULE ]  
    [ SUBSTRINGS MATCHING RULE SUBMATCHINGRULE ]  
    [ SINGLEVALUED ]  
    ID OID }
```

ATNAME is the attribute type name, **SYNTAXNAME** identifies the attribute syntax, **BOUNDS** specifies restrictions on the size of the attribute values, **EQMATCHINGRULE** identifies the rules used when testing two attribute values for equality, **ORDMATCHINGRULE** identifies the rules used when performing less-than-or-equal and greater-than-or-equal comparisons between values, **SUBMATCHINGRULE** identifies the rules used when matching substrings to values, and **OID** is the object identifier for the attribute type.

Typically, **SYNTAXNAME** and all the matching-rule identifiers refer recursively to other ASN.1 macros. In most ASN.1 macros, the case of the letters used is not significant; upper- and lowercase letters may be used interchangeably. The following is a simple example:

```
name ATTRIBUTE ::= {  
    WITH SYNTAX DirectoryString { ub-name }  
    EQUALITY MATCHING RULE caseIgnoreMatch  
    SUBSTRINGS MATCHING RULE caseIgnoreSubstringsMatch  
    ID id-at-name }
```

This is the definition of the **name** attribute, which is really a supertype from which other naming attributes are derived. It has the syntax **DirectoryString**, which is one of the standard syntaxes LDAP uses. The parameter **ub-name** following the syntax identifier, which stands for "upper bound on name," is a limit on the size of the values for this attribute type. It is defined elsewhere as the integer value 64. Similarly, the OID **id-at-name** is defined as follows:

```
id-at-name OBJECT IDENTIFIER ::= { id-at 41 }
```

This identifier expands to the complete OID `2.5.4.41` because `id-at` is defined somewhere else as `2.5.4`. The matching rules `caseIgnoreMatch` and `caseIgnoreSubstringsMatch` also have their own ASN.1 definitions, but we will not explore them in detail here.

If an attribute type is a subtype of another attribute type (that is, if it is derived from another attribute), the ASN.1 attribute definition will generally be in the form shown in [Listing 8.12](#).

Listing 8.12 The General Form of a Subtyped Attribute Type in ASN.1

```
ATNAME ATTRIBUTE ::= {  
    SUBTYPE OF SUPERTYPE  
    [ EQUALITY MATCHING RULE EQMATCHINGRULE ]  
    [ ORDERING MATCHING RULE ORDMATCHINGRULE ]  
    [ SUBSTRINGS MATCHING RULE SUBMATCHINGRULE ]  
    [ SINGLEVALUED ]  
    ID OID }
```

`SUPERTYPE` is the type from which `ATNAME` is derived. The matching rules and any other information defined in the type's supertype need not be included. [Listing 8.13](#) shows a simple example of this kind of attribute type definition.

Listing 8.13 The ASN.1 Definition of the `cn` Attribute Type

```
-- upper bound for cn:  
  
ub-common-name INTEGER ::= 64  
  
-- OID branch for X.500 standard attribute types:  
  
id-at OBJECT IDENTIFIER ::= { 2.5.4 }  
  
-- OID for cn:  
  
id-at-commonName OBJECT IDENTIFIER ::= {id-at 3}  
  
-- the cn attribute type itself:  
  
commonName ATTRIBUTE ::= {  
    SUBTYPE OF name  
    WITH SYNTAX DirectoryString {ub-common-name}  
    ID id-at-commonName }
```

Note that the definitions of `ub-common-name` and `id-at-commonName` were included because the `commonName` definition depends on these other ASN.1 elements. A complete ASN.1 schema definition must always define or import all the elements it references, except for some basic elements defined by the ASN.1 specification itself. The lines in [Listing 8.13](#) that start with two hyphens (--) are simply comments included to aid humans who are reading the ASN.1 macros.

ASN.1 Object Class Definitions

ASN.1-based object class definitions use the format shown in [Listing 8.14](#).

Listing 8.14 The General Form of an ASN.1 Object Class Definition

```
OCNAME OBJECT-CLASS ::= {  
    SUBCLASS OF { SUPERCLASS }  
    MUST CONTAIN { REQATTRS }  
    MAY CONTAIN { ALLOWATTRS }  
    ID OID }
```

`OCNAME` is the name of the object class, `SUPERCLASS` is the object class from which it is derived, and `REQATTRS` and `ALLOWATTRS` are the sets of required and allowed attribute types, respectively. If multiple attribute types are in the `REQATTRS` or `ALLOWATTRS` sets, each type is separated from its neighbor by a vertical bar in this manner:

```
cn | sn
```

The order in which the attributes are listed is not significant. Sometimes other ASN.1 macros are used in place of an attribute type to refer to an entire set of attributes.

[Listing 8.15](#) shows a sample object class definition written using ASN.1. This `locality` class, OID 2.5.6.3, is used for entries that represent physical locations of various kinds, such as "the state of California" or "222-B Baker Street." It is a subclass of the class called `top`, which means that, as with all other structural object classes, the `objectclass` attribute is mandatory. The following attributes may optionally be included in `locality` entries: `description`, `searchGuide`, `localityName`, `stateOrProvinceName`, `streetAddress`, and `seeAlso`. Several of these attribute types are defined in the `LocaleAttributeSet` ASN.1 macro so that the set can easily be included in other object class definitions.

Listing 8.15 The ASN.1 Definition of the `locality` Object Class

```
-- OID branch for X.500 standard object classes:  
  
id-oc OBJECT IDENTIFIER ::= 2.5.6  
  
-- OID for locality object class:  
  
id-oc-locality OBJECT IDENTIFIER ::= {id-oc 3}  
  
-- a set of attributes that are useful for describing locales:
```

```

LocaleAttributeSet ATTRIBUTE ::= {
    localityName |
    stateOrProvinceName |
    streetAddress }

-- an object class that represents a physical location

locality OBJECT-CLASS ::= {
    SUBCLASS OF      { top }
    MAY CONTAIN     { description |
                      searchGuide |
                      LocaleAttributeSet |
                      seeAlso }

    ID id-oc-locality }

```

As you can see, reading ASN.1 is complex and often involves peeling away several layers until you get to the information you need. The process can be tedious because there is no way to tell in advance how many layers you will need to remove before you're finished. ASN.1 is a general-purpose tool; unfortunately, general-purpose tools are usually more difficult to use than are those designed to do one simple thing well.

The Schema-Checking Process

When a new entry is added to the directory or an existing entry is modified, the directory server that processes the request goes through a *schema-checking process* before committing the add or modify request. The schema-checking process is done before the directory database itself is altered and ensures that all new or modified directory entries conform to the schema rules. If an entry violates any of the schema rules in effect, the directory server rejects the request and a "constraint violation" error is returned to the LDAP client.

The steps that a server typically performs when checking schemas are as follows:

Step 1. Verify that all new or modified values conform to the syntax rules.

Step 2. Verify that at least one value for the `objectclass` attribute is present.

Step 3. For each object class, make sure that at least one value for each of the mandatory attributes is present.

Step 4. Check each attribute to make sure that it is allowed by one of the object classes and that the attribute has only one value if it is a single-valued attribute.

Note that the specifics and order of execution of some of the steps shown may vary in different directory service implementations. However, the basic idea is always the same: The resulting entry (whether new or modified) is checked for complete conformity with the server's schema rules, and an error is sent to the directory client if any problems are found. If there is a schema violation, the entire add or modify operation is rejected by the server.

Some implementations, such as Netscape Directory Server, provide a way to disable schema checking entirely, in which case entries that are added or modified are not checked against the schema rules at all.

Schema-Checking Examples

Let's assume that the `person` and `labeledURIObject` schemas shown in [Listing 8.8](#) and [8.10](#), respectively, are present in your directory server. If someone attempts to add this entry, the add operation will fail because a required attribute type, `sn`, is missing:

```
dn: cn=Babs Jensen,ou=People,dc=example,dc=com
objectClass: top
objectClass: person
cn: Babs Jensen
```

The following entry will also cause a failure during the schema-checking process, but for a different reason: One of the attribute types in the entry definition, `labeledURI`, is not allowed by any of the object classes present in the entry:

```
dn: cn=Babs Jensen,ou=People,dc=example,dc=com
objectClass: top
objectClass: person
sn: Jensen
cn: Babs Jensen
labeledURI: http://www.example.com/bjensen Babs's Home Page
```

The following entry will pass schema checking because all required attribute types are present and all attribute types are allowed by one of the object classes present in the entry:

```
dn: cn=Babs Jensen,ou=People,dc=example,dc=com
objectClass: top
objectClass: person
objectClass: labeledURIOBJECT
sn: Jensen
cn: Babs Jensen
labeledURI: http://www.example.com/bjensen Babs's Home Page
```

Schema Design Overview

Schema design is the process of selecting and defining schemas to be used in a directory service. Before tackling schema design, you should first follow the data design process described in [Chapter 7](#); to complete your schema design, you will need detailed information on what data elements you plan to store in your directory service.

Schema design involves these steps:

- Step 1.** Locate schemas provided with the applications you plan to deploy.
- Step 2.** Locate standard and directory vendor-provided schemas.
- Step 3.** Choose predefined schema elements to meet as many of your data element needs as possible.
- Step 4.** Develop schema extensions or define new schema elements to meet your remaining needs.
- Step 5.** Plan for schema maintenance and evolution.
- Step 6.** Document your schema design.

Overall, it is best to use existing schema elements whenever possible, and preferably widely published and well-documented schemas. This strategy is the easiest way to choose a schema, and it should enhance compatibility with the directory-enabled applications you will deploy in the future. One of the challenges in schema design is to locate good predefined schemas to use, so we cover that topic in some depth.

If no existing schema meets your needs, you will need to create a custom schema by subclassing an existing object class definition or by defining completely new schema elements. When defining your own schema elements, it is important to use proper techniques to avoid potential problems down the road. You will also want to document your schema design (a simple but important step that many people overlook). Finally, it is a good idea to develop a strategy for maintaining your schemas and to plan for future changes.

A Few Words about Schema Configuration

In most directory server implementations, the administrator of the server determines what schema rules are in effect on a given server. Although we assume that schema customization is possible, the ease and degree to which it can be done vary widely among implementations.

An administration interface that can be used to customize schemas is included with most, but not all, directory service software. In some systems you can change schemas only by writing code that calls vendor-specific APIs. If you have already chosen your directory server software or are considering a specific implementation, check the documentation to make sure that you can and know how to perform any necessary schema customizations.

Implementations also differ in whether all data stored throughout the directory service is subject to the same set of schema rules or whether schema configuration can vary from server to server. In some implementations, different portions of the directory namespace can have their own schema rules, thus allowing the administrator to arrange for different subtrees within the directory to use different schemas. The term *subschema* refers to the schema that is in use for a specific subtree of the directory.

If you are deploying a centrally managed directory service, in most situations you should use the same schema everywhere. On the other hand, if you are deploying more than one service, or if you are an Internet service provider (ISP) that is providing directory service to more than one customer, it may be appropriate to use different schemas in different parts of the service.

Finally, you should check whether the directory server software you plan to use supports automated replication of schema information or whether you will need to take your own steps to ensure that the correct schema rules are installed on all servers you deploy. Not all implementations that support replication of user directory data can replicate schema information.

The Relationship of Schema Design to Data Design

When we tackled data design in [Chapter 7](#), we focused mainly on creating a detailed list of data elements to be included in the directory service. Most of the data design work was driven by the requirements of the applications that will use the directory service. During schema design, each data element is mapped to an LDAP attribute type, and related elements are gathered into LDAP object classes.

If some of the applications you plan to deploy come with a set of recommended schema definitions, the schema design process will probably be easier. You can also choose from many standard schemas for common objects such as people, groups, and departments. However, you may need to design your own schemas or refine the vendor-provided and standard schemas to meet your requirements. In addition, if two different directory-enabled applications recommend the use of radically different schemas for the same type of object, such as a person, you will need to sort that out during the schema design process as well. Harmonizing schemas is important, so look for a high degree of configurability when selecting your directory server and application software.

Let's Call the Whole Thing Off

Some of you may wonder whether you really need schema rules. Sometimes it seems like the schemas just get in the way of applications and users who want freedom when using the directory service. On the basis of our experience, and for all the reasons mentioned earlier in this chapter, it is important to use schemas within a directory service. Depending on how you use your directory, however, it may be appropriate to deemphasize schema rules or eliminate schema checking altogether.

Some implementations, such as Netscape Directory Server, allow you to disable schema checking entirely. Also, as discussed earlier, all LDAPv3-compliant servers support a special object class called `extensibleObject` that, when applied to an entry, allows any defined attribute type to be stored in the entry (effectively disabling the *may* aspect of object class-related schema checking). Most directory service administrators will want to keep schema checking enabled and use the `extensibleObject` class only in special situations (for example, for applications that need to store their own configuration information and do not need the directory server itself to enforce schema rules).

Caution

If you do decide to downplay or disable schema rules, be careful. It is nearly impossible to impose order later on a collection of data that was created free from the constraints of schema checking! We suggest that you proceed with caution and at least begin your directory service deployment with schema

checking enabled, even if you believe you will turn it off eventually.

Team LiB

◀ PREVIOUS

NEXT ▶

Sources of Predefined Schemas

Predefined schemas are provided by application vendors, directory standards documents, and directory service software vendors. By choosing predefined schemas to meet as many of your needs as possible, you avoid reinventing the wheel and help ensure compatibility with as many directory-enabled applications as possible.

Directory-Enabled Applications

All directory-enabled application software should include documentation that describes its schema requirements. If you can't locate the schema information for an application that you plan to deploy, contact the vendor or author of the software and ask for the information. Often this important information is buried in an appendix or is available only on a vendor's Web site.

Whereas some applications have specific requirements, others are more liberal in the schemas they can use. It is important to find out whether an application requires a specific set of object classes and attributes or can be configured to work with a variety of schemas. The directory-enabled application that is the least flexible of those you plan to deploy may in practice dictate many of your schema choices.

For example, routing of e-mail is a fairly exact process. If you plan to deploy a directory-enabled e-mail routing system, you may find that the e-mail software requires a specific object class to be present in all person and group entries and that specific attributes must have values for the mail-routing algorithms to function. Sun ONE Messaging Server is an example of LDAP-enabled mail-routing and delivery software that has fairly strict schema requirements.

In contrast, consider a directory lookup client such as a Web-based phone book or an e-mail client such as the Netscape 7 Mail client. These kinds of applications typically support the standard `person` object class and its associated attributes out of the box. They often support other schemas chosen by the vendor, such as the `inetOrgPerson` extended person object class. A good lookup client also provides some degree of customization so that it can be modified to accommodate organization-specific schemas.

Standard Schemas

Schemas endorsed by more than one vendor and published by a standards body can be good choices for use in your directory service. Usually a schema that has made it into a standards document has been reviewed and agreed to by several different implementers and users of directory services. Standard schemas also have the advantage that they are generally well documented and widely published.

It is especially useful to choose a standard schema when you're caught between two different vendors of directory-enabled applications that are each promoting their own proprietary schemas. If the schema needs of two applications conflict, asking the application vendors to support a standard schema is a good approach. Because everyone claims to believe in open standards, it would be difficult for the vendors to explain why they cannot support the standard schema (unless it simply does not meet the needs of their particular application).

Several sources of standard schemas should be consulted:

- LDAP standards documents
- X.500 standards documents
- Industry consortium standards

Table 8.5. Examples of Standard Schemas

Source	Status	Description
--------	--------	-------------

IETF/RFC 2247	IETF proposed standard	Using Domains in LDAP/X.500 DNs (<code>dc</code> , <code>domain</code> , <code>dcObject</code>)
IETF/RFC 2256	IETF proposed standard	X.500 user schema for use with LDAPv3
IETF/RFC 2587	IETF proposed standard	Internet public key infrastructure (PKI) schema—for example, <code>pkiUser</code>
IETF/RFC 2798	De facto industry standard	<code>inetOrgPerson</code> extended person schema

The Open Group Proposed industry standard Kerberos version 5 schema

Most standard schema information is available free of cost on the World Wide Web. In some cases, you may need to contact the standards body directly to purchase a copy of the standards documents. [Table 8.5](#) shows some examples of standard schemas.

If the schemas you design are of general use, consider submitting them to a standards body as well. Some URLs and other pointers to help you locate standard schemas are included in the Further Reading section near the end of this chapter.

Schemas Provided by Directory Vendors

Last, but certainly not least, most directory software comes with a generous collection of predefined schemas. In most cases, these schemas are a mix of standard schemas, application-specific schemas, and schemas that are being promoted by the directory vendor. One big advantage of directory vendor-provided schemas is that they are more than likely already installed in the directory service software, so they require less work to use.

On the other hand, just the fact that a schema comes preinstalled does not mean that it will meet your needs. As with all other schema-related decisions, read through the documentation and the schema definitions carefully to evaluate a vendor-provided schema against the needs of the applications you plan to deploy.

Adding a Schema to an Installed Directory Server

The procedure for adding a schema to a directory server varies from product to product, but an increasing number of servers support adding LDAPv3-format schemas over LDAP itself. You do this using an LDAP modify operation that targets the subschema entry to which you want to add the schema. A special utility may be provided for this purpose, or you may just need to create the correct LDAP Data Interchange Format (LDIF) file yourself.

With the Netscape Directory Server 6 product, schemas can be manipulated in several ways. The most interactive way is to use the graphical management console. But if you have a new schema that is in the LDAPv3 format, you will probably want to add the schema by dropping a new file into the `INSTALLDIR/slapd-INSTANCE/config/schema/` subdirectory or by adding the schema over LDAP. To add it over LDAP, you need to create an LDIF file that represents an LDAP modify operation on the `cn=schema` entry. [Listing 8.16](#) shows an example of an LDIF file that may be used to add some of the schema from RFC 2587 (*Internet X.509 Public Key Infrastructure LDAPv2 Schema*).

Listing 8.16 An LDIF File to Add Additional Parts of the RFC 2587 Schema

```
dn: cn=schema
```

```
changetype: modify  
  
add: objectClasses  
  
objectClasses: ( 2.5.6.21 NAME 'pkiUser' SUP top AUXILIARY  
    MAY userCertificate )  
  
objectClasses: ( 2.5.6.22 NAME 'pkiCA' SUP top AUXILIARY  
    MAY ( cACertificate $ certificateRevocationList $  
        authorityRevocationList $ crossCertificatePair ) )
```

Note that the two lines that start with `MAY` and the one that starts with `authorityRevocationList` are continued LDIF lines that begin with two space characters. To add attribute type definitions, the procedure is similar: Just include the appropriate `attributeTypes` values in the LDIF file you use as input to an LDAP modify operation. Assuming that the preceding LDIF is placed in a file called `pki-schema.ldif`, an `ldapmodify` command to add this schema will look something like this:

```
ldapmodify -v -D "uid=dsadmin,dc=example,dc=com" -w secret < pki-schema.ldif
```

Here is sample output from such a command:

```
ldapmodify: started Mon Jun 6 16:45:39 2002  
  
ldap_init( localhost, 389 )  
  
add objectClasses: ( 2.5.6.21 NAME 'pkiUser' SUP top AUXILIARY MAY userCertificate ) ( 2.  
    5.6.22 NAME 'pkiCA' SUP top AUXILIARY MAY ( cACertificate $ certificateRevocationList  
    $  
    authorityRevocationList $ crossCertificatePair ) )  
  
modifying entry cn=schema  
  
modify complete
```

Defining New Schema Elements

If your schema needs are not adequately met by existing, predefined schemas, you will need to define your own object classes and attributes. It is fairly easy to define your own schemas, but as with all design tasks, there are pitfalls and trade-offs to be considered. In this section we describe an approach that should produce good results. We do not discuss defining new attribute syntaxes because in most implementations adding new syntaxes requires code to be written to support it, usually by the directory software vendor.

Choosing Names for New Attribute Types and Object Classes

You should choose a naming scheme for the new object classes and attributes you define. All names should be made as meaningful as possible but not too long or cumbersome. Attribute names and object classes are generally hidden from end users, but directory service administrators and applications developers will work with them extensively.

It is also important to make some effort to avoid collisions with the names chosen by other parties (standards committees, directory vendors, other software vendors, and so on). Remember, the entire attribute namespace is flat. The same is true for the object class namespace. A good strategy is to prefix the names of all the schema elements you define with something that resembles your organization's name. If you do that, collisions with other definitions are unlikely.

For example, the ACME Corporation might use the prefix "acme" and create attributes and object classes such as these:

```
acmePerson (object class)  
acmePrinter (object class)  
acmeID (object class)  
acmeHoursAllowedAccess (attribute type)
```

Tip

If you define schema elements that you intend to publish widely and submit to a standards body such as the IETF, there is no need to prefix the names of the attributes and object classes with a string that identifies your organization. In fact, acceptance of your schema by others will likely be hindered if the names include something specific to your organization!

Obtaining and Assigning Object Identifiers

Recall that each LDAP object class or attribute type must be assigned a unique name and OID. One of the biggest stumbling blocks faced by people new to LDAP directories is how to obtain OIDs for the new attribute types and object classes they want to define. Simply put, OIDs can be obtained from anyone who has one. In fact, one OID is sufficient to meet all your schema needs; you can simply add another level of hierarchy to create new branches, or *OID arcs*, for your attributes and object classes. An OID arc is an OID that has been reserved for use as a container for defining additional OIDs. The process of assigning an arc of the OID space to another party is called *delegation*.

As already mentioned, an OID can be obtained from anyone who has one; that person just needs to delegate it to you as an arc and record this fact so that he does not use the OID for any other purpose. The Internet Assigned Numbers Authority (IANA) gives out OIDs to any organization that asks. IANA calls the OIDs *enterprise numbers* because it gives them out primarily for use with Simple Network Management Protocol (SNMP). The IANA OIDs work fine for LDAP as well because any one OID is as good as another. A form for obtaining an OID from IANA can be accessed on the World Wide Web at <http://www.isi.edu/cgi-bin/iana/enterprise.pl>.

Other organizations known to give out OIDs are the American National Standards Institute (ANSI) for U.S. organizations and the British Standards Institution (BSI) for U.K. organizations. General information on OIDs maintained by a gentleman named Harald Alvestrand can be accessed at <http://www.alvestrand.no/objectid>.

As an example of obtaining and assigning OIDs, consider the University of Michigan (U-M) directory services team, which contacted IANA to obtain an OID arc (1.3.6.1.4.1.250) for its own use. The team then created the OID arcs shown in [Table 8.6](#) for use in defining its own directory schema. The team assigned the OID 1.3.6.1.4.1.250.1.1 to the first attribute type it defined, 1.3.6.1.4.1.250.1.2 to the second, and so on.

Tip

After you obtain an OID, you should maintain a registry similar to [Table 8.6](#) to ensure that no OID is ever used for more than one purpose (the registry can just be a text file, perhaps maintained by a revision control system). You should then publish the list of OIDs with your schemas (more on this topic later). Although it may seem unimportant to establish an OID registry when you're just getting started, you will want to have one because the number of attributes and object classes that you or others in your organization create may be quite large in the end.

Table 8.6. University of Michigan OID Schema Arcs

OID Arcs	Description	Owner/Contact
1.3.6.1.4.1.250.1	U-M defined attribute types	U-M directory service team
1.3.6.1.4.1.250.2	U-M defined attribute syntaxes	U-M directory service team
1.3.6.1.4.1.250.3	U-M defined object classes	U-M directory service team

Modifying Existing Schema Elements

It may be tempting just to alter some predefined schema elements to meet your needs, perhaps by adding a new attribute to a predefined object class. At first glance, this seems like a reasonable thing to do, but it isn't. Do not modify existing schema elements! Changing existing schemas will break some directory servers and clients, and it will probably lead to a lot of confusion. If the `person` object class that everyone knows about is different within each directory service deployment, chaos will rule.

Also be careful when completely deleting object classes or attributes from any of your directory server software's preinstalled schemas. It's OK to do this as long as you're sure that the schema is not used internally by any of the directory service software. However, you may run into trouble when you upgrade your software because the upgrade process may require that all the vendor's schemas be present. In general, there is no reason to remove schema elements even if you do not plan to use them; there is little or no penalty associated with leaving the schema elements installed.

Subclassing an Existing Object Class

It is fairly common to extend, or *subclass*, an existing, predefined object class to add new attribute types to it. To do this, define a new object class that is a subclass of the existing one by indicating in the definition of the new class that the existing class is its superior. You also need to define the new attribute types and include them in the new object class as required or as optional attributes.

When you subclass an existing object class in this way, the new class should generally be used to represent the same type of object as the class from which it was derived. For example, you might create a new subclass of a `printer` object class called `hpPrinter` that allows additional attributes specific to Hewlett-Packard printers. The `hpPrinter` class is still used to represent printers; it just holds more information.

Also the new object class should be the same kind as the class from which it is derived. For example, if you are subclassing a structural object class, the new class should also be a structural class. Note that in some directory service implementations, such as Netscape's, little or no distinction is made between different kinds of object classes during schema checking, so this may not be something you have to worry too much about initially.

As another example of subclassing an existing object class, suppose that the Example Corporation is developing a directory-enabled application called the Birthday Notification Service (to which managers will subscribe so that they remember to take each employee out to lunch on his or her birthday). The designers of the Example directory might define a new attribute to hold the day and month a person was born. The attribute type definition might look like this:

```
( bday-OID NAME 'exampleBirthday' DESC 'birthday day-month' EQUALITY caseIgnoreMatch SUBSTR  
➡ caseIgnoreSubstringsMatch SYNTAX 1.3.6.1.4.1.1466.115.121.1.15 )
```

Note that `bday-OID` would need to be replaced by a real OID assigned by the Example directory administrators. An extension of the `inetOrgPerson` class that allows the new `exampleBirthday` attribute to be included in entries could be defined as shown in [Listing 8.17](#).

Listing 8.17 The `examplePerson` Object Class

```
( examplePersonOID NAME 'examplePerson' DESC 'Example Corporation extended person' SUP  
inetOrgPerson  
➡ STRUCTURAL MAY exampleBirthday )
```

Note that `examplePersonOID` should be replaced by a real OID assigned by the directory administrators. The `examplePerson` class would be used whenever a user entry was created inside Example Corporation. The following is a sample entry:

```
dn: uid=jcarter,ou=People,dc=example,dc=com  
  
objectclass: top  
  
objectclass: person  
  
objectclass: organizationalPerson  
  
objectclass: inetOrgPerson  
  
objectclass: examplePerson  
  
cn: John Carter  
  
sn: Carter  
  
uid: jcarter  
  
userPassword: secret  
  
exampleBirthday: 29-February
```

Now we all know why John claims to be so much younger than he looks; he must be counting birthdays instead of elapsed time!

Note

In the LDIF just shown, values for all the superior object classes (`top`, `person`, `organizationalPerson`, and `inetOrgPerson`) are explicitly shown in the entry's `objectClass` attribute. This is how an entry will look when you retrieve it from an LDAP server. When you add or modify an entry, however, you should need to provide only the most specific superior object class value (for example, `inetOrgPerson`). On the other hand, some LDAP server implementations do require all of the superior object class values to be listed, so it is safest always to do so.

Adding Auxiliary Information to a Directory Object

Sometimes it is preferable to create an auxiliary object class that allows attributes to be added to any type of LDAP entry, regardless of the kind of real-world object it represents. A class like this is sometimes called a *mix-in class* because it allows additional attributes to be "mixed into" an existing class. A mix-in class may be added to a wide range of entry types—a much simpler approach than creating a subclass for each object class in which you want to allow the new attributes to appear.

To create an auxiliary object class, simply define a new class that is not subclassed from any existing object class (it should have the special class `top` as its superior). Typically, all the attributes in the auxiliary class should be optional rather than mandatory. That way, the auxiliary object class itself can be associated with an entry regardless of whether any values for its attributes are present. As a result, the burden on directory clients is reduced because they do not have to worry about removing the object class value itself when the auxiliary attributes are removed.

[Listing 8.18](#) shows an example of a useful auxiliary object class. This class can be added to any LDAP entry to allow a `dc` (domain component) attribute value to be included in the entry.

Listing 8.18 The `dcObject` Auxiliary Object Class

```
( 1.3.6.1.4.1.1466.344 NAME 'dcObject' SUP top AUXILIARY MUST dc )
```

The following is an example of an organization entry that also contains domain components:

```
dn: o=Example Inc.,c=US
objectclass: top
objectclass: organization
objectclass: dcObject
o: Example Inc.
dc: example
```

As another example of an auxiliary object class, suppose that you develop a directory-enabled application that keeps track of your organization's network-related inventory by storing information in the following custom attribute types:

```
( OID NAME 'inventoryID' DESC 'numeric inventory ID' EQUALITY integerMatch SYNTAX 1.3.6.1.4.1.
  ↪ 1466.115.121.1.27 SINGLE-VALUE )

( OID NAME 'inventoryDatePlacedInService' DESC 'date item was placed into service' EQUALITY
  ↪ generalizedTimeMatch ORDERING generalizedTimeOrderingMatch SYNTAX 1.3.6.1.4.1.1466.115.
  ↪ 121.1.24 SINGLE-VALUE )

( OID NAME 'inventoryContactPerson' DESC 'pointer to entry of the person responsible for item'
  ↪ EQUALITY distinguishedNameMatch SYNTAX 1.3.6.1.4.1.1466.115.121.1.12 )

( OID NAME 'inventoryComments' DESC 'comments related to inventory status' EQUALITY
  ↪ caseIgnoreMatch SUBSTR caseIgnoreSubstringsMatch SYNTAX 1.3.6.1.4.1.1466.115.121.1.15 )
```

All the instances of `OID` would, of course, need to be replaced with real object identifiers you assign. Because you want to mix these attributes into several different types of entries (printers, hosts, and so on), you could handle your schema needs by defining an auxiliary object class such as the `inventoryItem` class shown in [Listing 8.19](#).

Listing 8.19 The `inventoryItem` Auxiliary Object Class

```
( OID NAME 'inventoryItem' DESC 'auxiliary object class to hold inventory information'
  ↪ SUP top AUXILIARY MUST inventoryID MAY ( inventoryDatePlacedInService $ 
  ↪ inventoryContactPerson $ inventoryComments ) )
```

Here is a sample printer entry that has inventory information attached to it:

```
dn: cn=2nd floor HP LaserJet 8150dn,ou=printers,dc=example,dc=com
objectclass: top
objectclass: printer
objectclass: inventoryItem
cn: 2nd floor HP LaserJet 8150dn
pagesPerMinute: 32
inventoryID: 129055581
inventoryDatePlacedInService: 20011201000000Z
inventoryContactPerson: uid=bjensen,ou=people,dc=example,dc=com
inventoryComments: on loan to the art department
```

Accommodating New Types of Objects

If you cannot find a predefined object class that is similar to the type of object you need to represent in your directory service, simply define a new structural class to hold whatever attributes are appropriate. This task is similar to creating a new auxiliary class, except that the structural class can stand on its own as the primary object class for an entry.

For example, if our friends at the Example Corporation plan to use their directory to track company-owned and -operated telephone sets, they might reuse some predefined attributes but create a new object class such as the one shown in [Listing 8.20](#).

Listing 8.20 The `exampleTelephone` Object Class

```
( OID NAME 'exampleTelephone' DESC 'Example Corporation telephone' SUP top STRUCTURAL MAY ( cn
  ↪ $ telephoneNumber $ owner $ L ) )
```

Here is a sample entry:

```
dn: cn=Mark Smith's phone,ou=phones,dc=example,dc=com
objectclass: top
objectclass: exampleTelephone
cn: Mark Smith's phone
telephoneNumber: 3477
owner: uid=bkady,ou=people,dc=example,dc=com
L: security office
```

Note that when you create an object class for an entirely new kind of object, you will need to put some thought into

which attribute will most likely be used to form the RDNs of entries that belong to the class. For most object classes, the `cn` (common name) attribute is a good, generic choice.

In the `exampleTelephone` object class we just looked at, the telephone number itself might actually be a better choice for naming phone entries simply because it is more likely to be unique. The sample entry we just used might instead be defined like this:

```
dn: telephoneNumber=3477,ou=phones,dc=example,dc=com
objectclass: top
objectclass: exampleTelephone
cn: Mark Smith's phone
telephoneNumber: 3477
owner: uid=bkady,ou=people,dc=example,dc=com
L: security office
```

The only difference is in the RDN (the first part of the DN), which uses the `telephoneNumber` attribute instead of `cn` to name the entry. See [Chapter 9](#), Namespace Design, for more information on entry naming.

Tips for Defining New Schemas

Defining a good schema is as much art as science, and the more you do it, the easier the process becomes. The following are some tips that will help you produce better results:

- Reuse existing elements as much as possible. Even if you need to define a new object class, you may find that many of the attributes you need already exist. When you reuse existing elements, make sure that the meaning and expected use is the same; otherwise, directory applications and users may be confused. For example, suppose that an attribute called `drink` has been defined that stores the name of a person's favorite beverage. Do not try to reuse the `drink` attribute with intended values of `yes` or `no` to indicate whether a person consumes alcohol. If you need to store a data value for that purpose, define a new attribute such as `drinksAlcohol`.
- Define several smaller auxiliary object classes to mix needed attributes into existing objects. The alternative is to subclass many object classes, and this subclassing typically requires more new classes to be defined. By providing a general solution in an auxiliary class, you will make your schema simpler and easier to understand.
- Minimize the number of mandatory attribute types within your object classes. If you're thinking of making an attribute required, proceed with caution. In our experience, required attributes inevitably get in the way at some point. Even the `cn` and `sn` attributes, which are required by the standard `person` class, can be a burden if, for example, a user needs to appear in the directory to get access to various systems but wants her name to remain private.
- Do not define more than one object class or attribute type to hold the same kind of information. To maintain consistency in your schema, and thus in your directory service, strive to use a single schema element for a given purpose. For example, the following three attributes should likely all be consolidated into one:

```
dateOfBirth
birthDate
birthDay
```

- When in doubt, keep it simple. Remember that the goal of a collection of schemas is to provide a framework for your data elements that is easily understandable and usable by directory applications, administrators, and users. The more complicated the schemas that you define are, the less approachable your directory service will be.

Documenting and Publishing Your Schemas

Documenting all the schemas you use within your directory service is important for several reasons. First, you may need the schema information when you add additional servers to your service. Second, you can share your schema documentation with software vendors and authors of custom applications to aid in design, development, and troubleshooting. Finally, if you create a new, useful schema, you can easily share your design with others and consider promoting it as a standard.

When documenting schemas, you can use any format you want, but we recommend the LDAPv3 format, which is the one used by most software and throughout this book. However, if you have already chosen directory service software, you may want to use its favored format. You should also identify the attribute types and object classes you plan to use and add their names to the data elements list you created during the data design process (see [Chapter 7, Data Design](#)).

We recommend that you publish your directory schema definitions at least within your own organization. Schema information is useful to users of the directory and developers of directory-enabled applications. If you want other organizations and independent software vendors to adopt your schemas, you should publish them on an external Web site or in another appropriate place where everyone will have access to them.

Finally, if you plan to promote some schema elements that you designed for consideration as a standard, you should publish them using the process defined by the standards body. For the IETF, which handles most Internet standards (including LDAP), this means writing an Internet Draft and submitting it to the IETF secretariat for publication. Connect to the IETF World Wide Web site at <http://www.ietf.org> for more information on how to contribute. Other groups, such as the DMTF (Distributed Management Task Force) and OASIS (the Organization for the Advancement of Structured Information Standards), help organizations develop and publish industry standards.

Schema Maintenance and Evolution

Your schema needs will change over time as you bring up new applications and find new and interesting ways to use your directory service. So far in this chapter, we have generally assumed that one person or a small group of people will look after the schema for the directory service as a whole. This is a good model to follow initially, but when your directory service becomes popular and new applications are rapidly being proposed, it may be difficult to keep up with the demand for new schemas. A more decentralized approach to schema design would then be needed.

After you gain some deployment experience, you may also find yourself wishing you could change some of the schema rules that you defined when you initially deployed your directory service. Changing the existing schema is tricky, but it is possible in certain situations described later in this section. There are also some schema-related issues to be aware of when you're upgrading directory service software, which we discuss as well.

Establishing a Schema Review Board

One option is to allow people to define and submit schemas to a centralized review committee for approval before they are installed in the directory service. The main job of the review board (which can be just one or two people) is to check for inconsistencies in the schema, ensure that redundancy is not being introduced, and make sure that the schema is well defined and well documented. This same group can also perform clerical tasks such as assigning OIDs, and it can serve as a central point for schema advice and consent.

Granting Permission to Change the Schema Configuration

If your directory server software supports it, you may want to allow people installing new applications to perform online schema updates over LDAP. Be careful to limit the number of people who have the access rights necessary to do this; you do not want frivolous, inappropriate, or inconsistent schemas to be installed. Check with your directory server software vendor to see if online schema updates are allowed and to find out how to control access.

Changing Existing Schemas

As with all other aspects of design, it is difficult to produce a perfect, complete schema design the first time. Because the use of your directory service will change over time, so will your schema needs. It may be tempting to change your defined schemas to accommodate your changing service, but proceed with caution. It is probably OK to add optional attribute types to an object class you previously defined, but it is risky to try to remove any attribute types or add required attribute types. In practice, there is usually no reason to remove attributes or add mandatory ones.

If you defined an attribute type that has the wrong syntax or name, you need to define a new type but keep the old one around and transition away from it. The most important consideration when you're contemplating changes to an existing schema is to make sure that you have thought carefully about how those changes affect users, directory-enabled applications, and the directory itself.

Upgrading Directory Service Software

When the time comes to upgrade your directory service software, make sure that all your schema additions are preserved during the upgrade process. Well-designed software takes care of this for you, but otherwise you need to reconfigure the new version of the software to make it aware of your schema rules. Also, the potential trouble with software upgrades is the most compelling reason not to remove any of the schemas that come preconfigured with your directory service software.

Team LiB

◀ PREVIOUS

NEXT ▶

Schema Design Checklist

Follow these steps to produce a comprehensive schema design for your directory service:

- Locate schemas provided with applications.
- Locate standard and directory vendor–provided schemas.
- Choose predefined schema elements to meet as many needs as possible.
- Define new schemas to meet remaining needs.
- Document the schema design.
- Plan for schema maintenance and evolution.

Further Reading

Abstract Syntax Notation One (ASN.1)—Specification of Basic Notation (ITU-T Recommendation X.680). 1994. Available for purchase on the World Wide Web at <http://www.itu.ch>.

The Directory: Models (ITU-T Recommendation X.501), 1996. Available for purchase on the World Wide Web at <http://www.itu.ch>.

The Directory: Selected Attribute Types (ITU-T Recommendation X.520), 1996. Available for purchase on the World Wide Web at <http://www.itu.ch>.

The Directory: Selected Object Classes (ITU-T Recommendation X.521), 1996. Available for purchase on the World Wide Web at <http://www.itu.ch>.

DMTF (Distributed Management Task Force) Web site, <http://www.dmtf.org>.

"A Layman's Guide to a Subset of ASN.1, BER, and DER: An RSA Laboratories Technical Note." B. Kaliski, 1993. Available on RSA's FTP site at <ftp://ftp.rsa.com/pub/pkcs/ascii/layman.asc>.

Lightweight Directory Access Protocol (v3): Attribute Syntax Definitions (RFC 2252). M. Wahl, A. Coulbeck, T. Howes, and S. Kille, 1998. Available on the World Wide Web at <http://www.ietf.org/rfc/rfc2252.txt>.

Microsoft Active Directory Schema. Available on the World Wide Web at http://msdn.microsoft.com/library/en-us/netdir/ad/active_directory_schema.asp.

Netscape Directory Server Deployment Guide: Chapter 3, How to Design the Schema. Available on the World Wide Web at <http://enterprise.netscape.com/docs/directory>.

Netscape Directory Server Schema Reference. Available on the World Wide Web at <http://enterprise.netscape.com/docs/directory>.

Novell Directory Services Schema Reference. Available on the World Wide Web at http://developer.novell.com/ndk/doc/nasl/lib/schm_enu/data/h4q1mn1i.html.

OASIS (Organization for the Advancement of Structured Information Standards) Web site, <http://www.oasis-open.org>.

A Summary of the X.500(96) User Schema for Use with LDAPv3 (RFC 2256). M. Wahl, 1997. Available on the World Wide Web at <http://www.ietf.org/rfc/rfc2256.txt>.

Understanding X.500: The Directory. D. Chadwick, International Thomson Computer Press, 1996. Now out of print; selected portions available on the World Wide Web at <http://www.isi.salford.ac.uk/staff/dwc/X500.htm>.

Looking Ahead

You should now have a plan that lists all the data elements you will place in your directory service, along with the complete collection of schemas you will use to represent the data elements as LDAP entries. Schema design is important, but it can be somewhat tedious. [Chapter 9](#) will cover an area in which you will have a chance to be much more creative: directory namespace design.

Chapter 9. Namespace Design

- The Structure of a Namespace
- The Purposes of a Namespace
- Analyzing Your Namespace Needs
- Examples of Namespaces
- Namespace Design Checklist
- Further Reading
- Looking Ahead

Designing a directory namespace is one of the most important tasks you will undertake when designing your directory service. The directory namespace provides the basic means by which you reference information in your directory, but it has many other implications as well. A properly designed namespace can lead to

- Easier data maintenance
- More flexibility in setting access control and replication policies
- The ability to satisfy a wider variety of directory-enabled applications
- More natural navigation through the directory
- Happy directory users and administrators

On the other hand, a poorly designed namespace can lead to administrative hassles when directory entry names change, replication or access control requirements change, or users try to find information. In the worst case, the namespace must be redesigned to support a vital new directory application. Poor design generally results in unhappy directory users and frustrated, overworked administrators.

The namespace you design for your directory has far-reaching implications that are often not at all obvious when you set out. The design of your namespace can affect replication, whether and how you are able to partition your data among servers or distribute administration of the directory, and other aspects of the service. Furthermore, changing your namespace after you've designed and deployed your directory service is a difficult task, unpleasant for administrators and often inconvenient for users.

These implications make namespace design one of the most critical tasks you will face during your directory design process. Don't be surprised if your initial namespace design proves inadequate when you move on to designing your replication or access control framework, or even when you begin to pilot your directory service. Don't be afraid to redesign during these early stages; redesigning later is much more costly.

This chapter is closely related to [Chapters 10](#), Topology Design, and [11](#), Replication Design. In fact, we suggest that you read all three chapters as a unit because decisions concerning namespace design have direct consequences for your directory topology, which in turn has a direct bearing on your replication strategy.

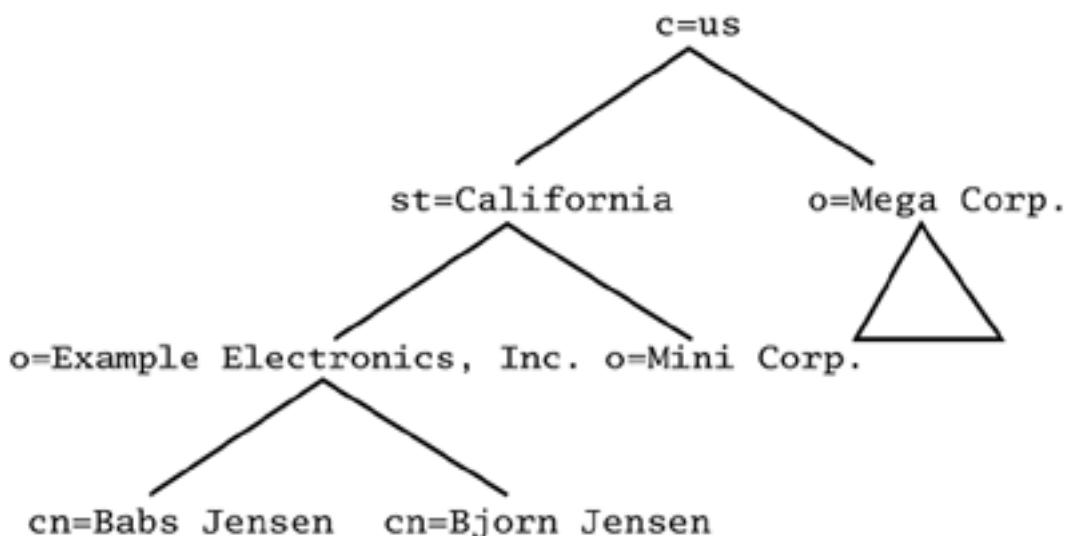
This chapter introduces the fundamentals of namespace design, starting with a brief review of the syntactic structure of an LDAP namespace, followed by a discussion of the purposes of a namespace. Then we describe how to analyze your namespace needs and design the best namespace for you. Several different namespace designs addressing a variety of needs and environments are presented near the end of the chapter, followed by a checklist of things to consider when you're designing your namespace.

The Structure of a Namespace

The LDAP model defines a flexible namespace framework, which means that you can design a namespace to satisfy your requirements no matter what they may be. However, it also means that you have more choices to make than you might like.

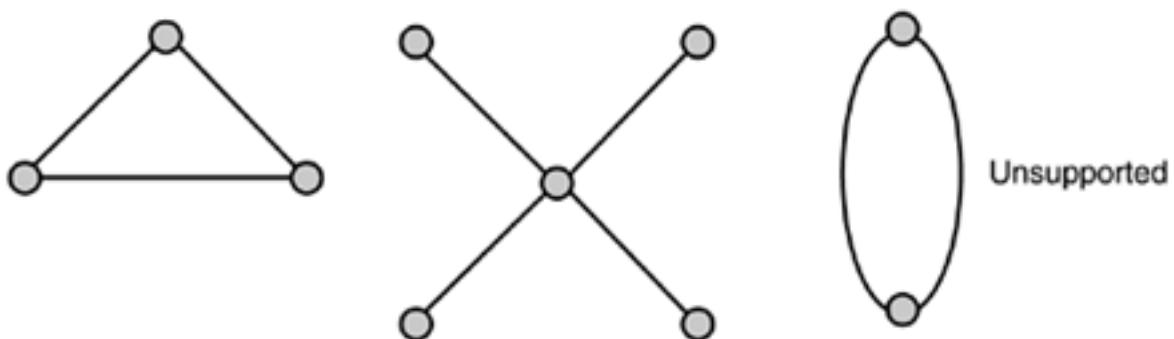
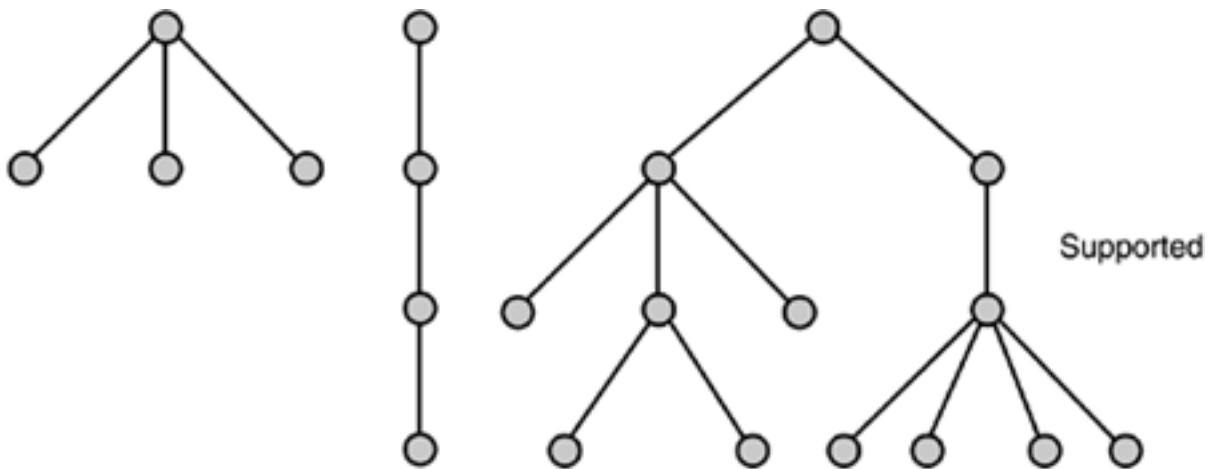
The LDAP namespace model is inherited from the X.500 directory standard, which is intended to be used in a rigid, worldwide, hierarchical directory service. A typical X.500 namespace starts with countries at the top of the namespace, perhaps followed by states in the United States, and then organizations below that, with further hierarchy in each organization (see [Figure 9.1](#)). Although such a hierarchical namespace may suit your needs and has a certain aesthetic appeal, it has some serious drawbacks, which are discussed later in this chapter. Fortunately, the LDAP model is flexible enough to allow other, more practical designs such as those discussed in the section [Analyzing Your Namespace Needs](#) later in this chapter.

Figure 9.1. A Typical X.500 Namespace



As mentioned in [Chapter 2](#), Introduction to LDAP, the basic LDAP model is hierarchical, or tree-structured. Each entry has exactly one parent, except for the root entry, and every entry may have any number of child entries beneath it. [Figure 9.2](#) shows some examples of supported and unsupported namespace structures.

Figure 9.2. Sample Supported and Unsupported Namespace Structures



Of course, the simplest hierarchical case of a one-level, flat namespace is also allowed. LDAP does not directly support an arbitrarily connected namespace, or graph structure, in which an entry might be both child and parent of the same entry. LDAP does, however, support the concept of aliases, which can be used to construct such structures (see [Chapter 2](#), Introduction to LDAP, for more information on aliases).

Also keep in mind that through the use of `seeAlso` and other distinguished name (DN)-valued attributes, you can construct arbitrary relationships among entries. We prefer the use of this latter approach over the use of aliases, which tend to cause problems with performance and consistency maintenance.

After determining the hierarchical relationship among entries (a subject we tackle later in this chapter), the next question is how each entry in the hierarchy is named. First, recall from [Chapter 2](#) that an LDAP directory entry is a collection of attribute values. To create the name of an entry, we usually choose one of these attribute values to form a *relative distinguished name (RDN)*.

For example, for the entry depicted in [Figure 9.3](#), you could choose any of the RDNs shown, provided that none of the entry's sibling entries has the same name. An entry's RDN must be unique among all entries sharing the same parent entry.

It's possible to use more than one attribute value from the entry when forming the RDN. An RDN formed in this fashion is called a *multivalued RDN*. The multiple values that form the RDN are separated by a plus sign. For example, `uid=babs+sn=jensen` is a valid RDN for the entry shown in [Figure 9.3](#). We recommend not using multivalued RDNs for the reasons discussed in [Chapter 2](#). To review, multivalued RDNs introduce needless complexity, can adversely affect performance, and often don't solve the problems they are intended to solve.

When the RDN of an entry is chosen and its position relative to other entries in the hierarchy

is determined, forming the entry's full name is simple: Just combine the RDNs of the entry and all its ancestor entries. The resulting name is called a distinguished name, or DN.

Figure 9.3 Examples of RDNs

cn: Barbara Jensen	cn=Barbara Jensen
cn: Babs Jensen	uid=babs
sn: Jensen	mail=babs@example.com
title: Director	
uid: babs	
mail:babs@example.com	

Entry

Examples of RDNs

Figure 9.4 Examples of DNs

cn=Barbara Jensen, o=Netscape, c=us
cn=Barbara Jensen, o=Netscape.com
c=us
o=Netscape, c=us
dc=example, dc=com
o="Mega Corp., Inc.", c=us

In LDAP, DNs are written in little-endian order, like an e-mail address, rather than big-endian order, like a file system name. *Little-endian* means that the least significant component is written first, with increasingly more significant components written subsequently. Each component of the name is separated by a comma, and if the entry is named with a multivalued RDN, the multiple components of the RDN are separated by a plus sign. If these or other troublesome characters occur in one of the values used to form the name, they are escaped by means of a backslash quoting mechanism. [Figure 9.4](#) shows some examples of DNs.

More information on the LDAP namespace model, including a discussion of special characters that must be escaped when used in DNs, can be found in [Chapter 2](#), Introduction to LDAP.

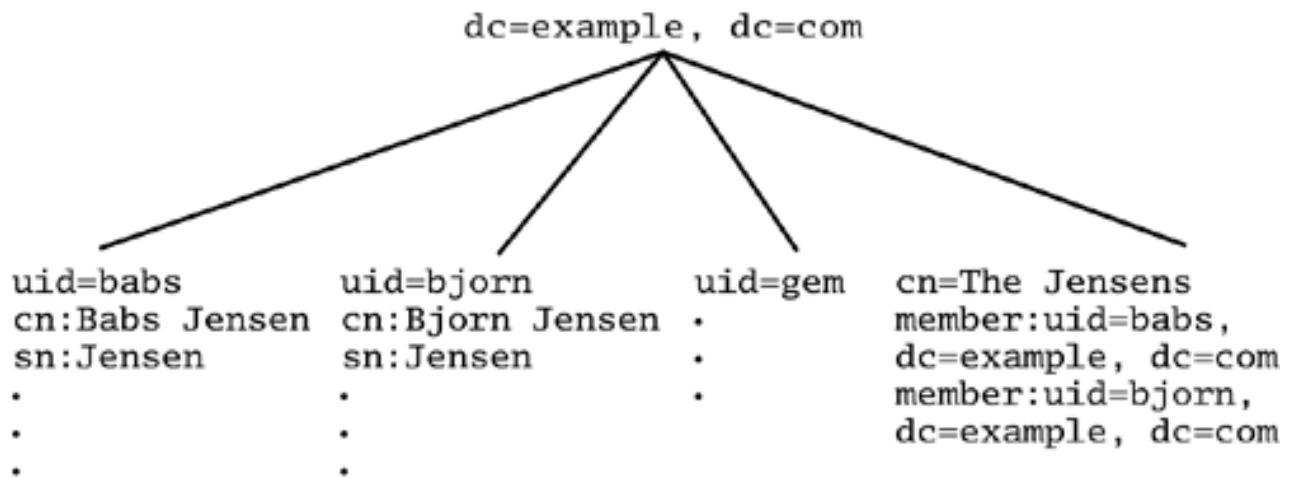
The Purposes of a Namespace

A namespace provides the means by which directory data is named and referenced. In this respect, directory entries need names for the same reason that you or I need a name—so that we can be referred to by a more meaningful and precise term than "Hey, you!"

In some environments, few, if any, additional requirements are placed on the namespace. In others, requirements for access control, replication, data partitioning, application access, and perhaps other aspects of the service impose additional requirements. This section summarizes the more common purposes of a namespace:

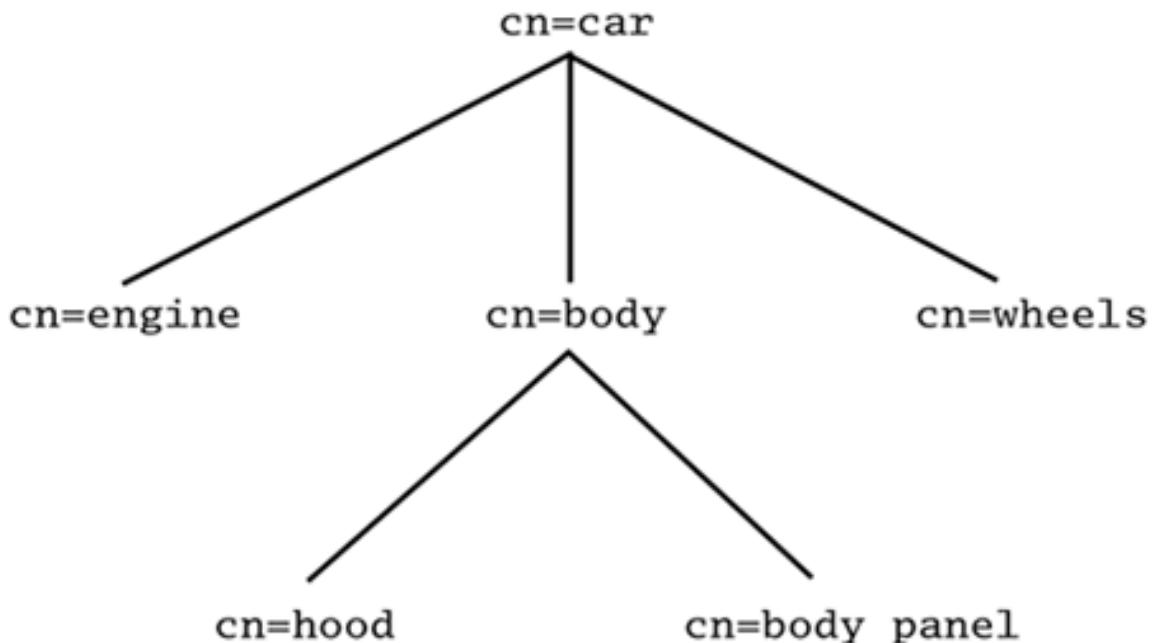
- **Data reference.** A namespace provides the means by which directory data is referenced, which is important for two reasons. First, there must be a way for directory clients to refer unambiguously to a directory entry. Second, the directory name provides a compact and efficient way to support groups of directory entries and directory entries that refer to one another (for example, through the use of a `member`, `seeAlso`, `owner`, or `manager` attribute). [Figure 9.5](#) shows an example of this application.

Figure 9.5. A Data Reference Example



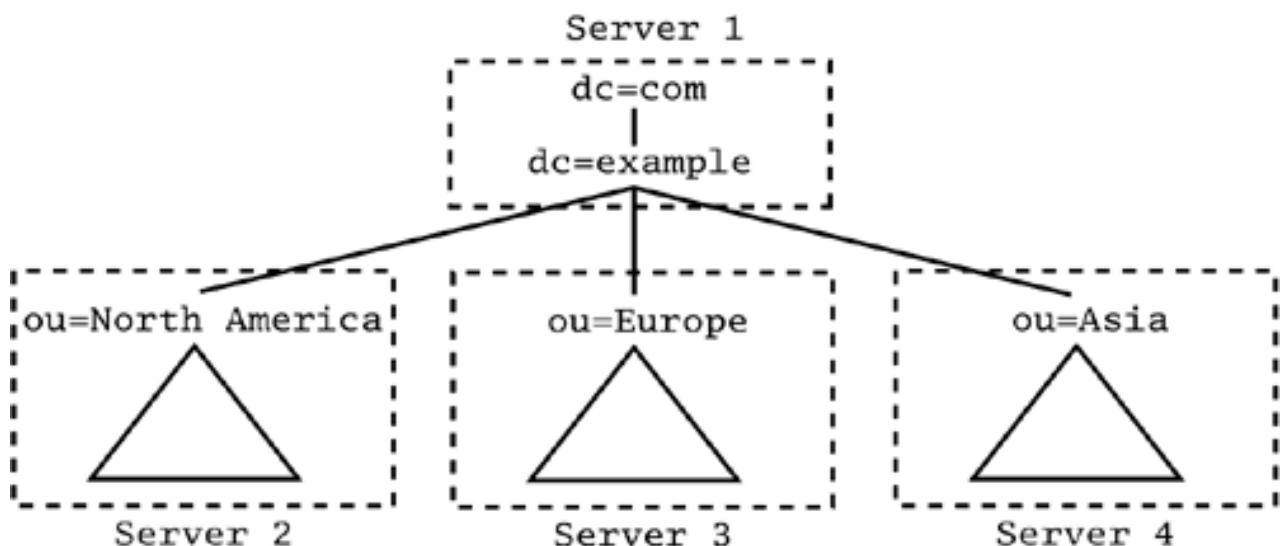
- **Data organization.** A namespace provides a way to organize data. For example, you might place all entries corresponding to people in one portion of the namespace, entries corresponding to devices (such as printers) in another, and entries corresponding to groups in yet another part of the namespace. Further organizational subdivisions are also possible, perhaps based on geographical or organizational information. Such organization can help facilitate browsing of the directory. For example, a printing application that enables a user to choose a nearby printer on the basis of characteristics such as speed, color, and two-sided printing capability can more easily present only printers in a user's vicinity if that portion of the directory is organized by locality. [Figure 9.6](#) shows a namespace designed for this purpose.

Figure 9.6. A Namespace Example for Data Organization



- **Data partitioning.** A namespace enables directory data to be partitioned, or divided, among multiple servers. Typically, data can be partitioned only at a branch point in the directory. If your namespace does not contain branch points (for example, if you have designed a one-level, flat namespace), you cannot partition your data. In contrast, [Figure 9.7](#) shows a partitioning scheme in which the namespace enables a company's three main offices each to own and manage its own part of the data. Partitioning is discussed in more detail in [Chapter 10](#), Topology Design.

Figure 9.7. A Namespace Example for Partitioning, Replication, and Delegation through Access Control



- **Data replication.** Although not directly related to replication, your choice of namespace can constrain the replication scenarios your directory can support. This is a logical consequence of the partitioning restrictions already described. Most replication solutions allow replication of only an entire partition. Therefore, if you choose a completely flat namespace, it is impossible to replicate only a portion of your data. The example shown in [Figure 9.7](#) shows the directory divided into four partitions, each of which may be replicated (each of the four partitions is represented by a box in [Figure 9.7](#)). [Chapter 11](#), Replication Design, says more on this topic.
- **Access control.** Like replication, access control can be affected by your choice of namespace. Some products allow the setting of access control only at namespace branch

points. This is especially true when you're delegating authority to some of your data. Delegation may be possible only if the data is divided on a subtree basis. Some products have a more flexible access control scheme, allowing control to be delegated on a basis other than the namespace. For example, Netscape Directory Server 6 allows you to specify access control rules that apply to any entry in your directory that satisfies a given directory search filter. This ability to delegate authority over portions of data is one of the most important features of a directory service. Access control is discussed extensively in [Chapter 12](#), Privacy and Security Design.

- **Application support.** The purposes of a namespace we've discussed so far have all been related to operation of the directory itself. The final purpose we will mention is that your namespace should support the applications for which you're designing your directory in the first place. This may seem almost too obvious to mention, but in our experience it is easy to get caught up in the excitement of directory design and lose track of the big picture. Check whether any of the applications you plan to deploy require a particular namespace design, and make sure that the namespace you design satisfies the requirements of your applications.

You may be tempted to ascribe another purpose to your namespace, perhaps requiring that it should hold some aesthetic value of its own. After all, if people have to look at the names, you might as well make them meaningful, even pleasing to look at. Perhaps your users might even be able to guess the names of entries, given a standard, intuitively constructed namespace. This may be a nice idea in theory, but such dreams seldom work out in practice. Do not be fooled into considering such ideas.

A namespace should have primarily functional value, and the structure of your namespace should be hidden from users as much as possible. The primary purpose of a name is to provide a unique way of referencing entries, and the driver behind its design is ease of administration. The other purposes described, including the effect your namespace may have on replication and access control, may also affect your choice of namespace. But these goals are administrative, and they should not be inflicted on your users.

A well-designed directory client should hide the directory namespace from users of the client (or at least it should provide this option), and most modern clients do this successfully. Failure to hide directory names often results in users being confused or upset by the name given to their entry, and users are unlikely to be sympathetic to the administrative concerns that caused you to name entries as you did. Furthermore, if you ever need to redesign your namespace, you will be glad you hid the original namespace from your users.

Tip

Choose names for administrative convenience, not aesthetic value. Try to hide names from users of the directory as much as possible. When developing applications, avoid making any assumptions about namespace design. Be sure to make your application flexible enough to adapt to different namespaces.

In summary, a namespace is required to reference entries, to support features such as groups, and to support the applications using the directory. Your choice of namespace interacts with other important design decisions, often constraining the choices available when you're distributing, replicating, or controlling access to your data. These considerations are primary; other considerations, such as the aesthetics of your namespace, should be given less weight.

Analyzing Your Namespace Needs

Now that you have some idea what you're going to do with your namespace after you define it, it's time to turn our attention to the design process itself. The first step in designing a namespace is to understand your needs. Do you need a flat namespace or a hierarchical one? What attributes should you use to name entries? Do you have replication or partitioning needs that may affect the design of your namespace? What about access control? What applications will the directory be supporting? Are your needs constant, or might they change over time? These questions and others need to be answered before you can have confidence in your namespace design.

This section takes you through the major decisions you'll need to make when designing your namespace. Keep in mind that, like many other design problems, namespace design involves a series of trade-offs, such as administrative convenience for future flexibility. As we examine each of these trade-offs, we'll try to point out what you gain and what you sacrifice at each step. At the end of this chapter, we provide a checklist summary of the issues you should consider during the design process.

Choosing a Suffix

Your directory may have only a local scope, or it may be part of a larger, or even a global, directory system. In either case, one of the first choices you have to make when designing your namespace is determining the suffix below which your namespace will live. Picture your namespace as a tree: A *suffix* is the name of the entry at the top of the subtree you're designing.

Although LDAP places no restrictions on the suffix you use, three methods are commonly used. All the methods base the suffix on the name of your organization so that it's likely to be unique. This practice allows your directory to coexist with other LDAP servers, should the need arise (imagine what would happen if your company merged with another company).

The first method, and the one we recommend, is the technique described in RFC 2247 that maps a DNS domain name to a DN. In summary, the technique is to separate the components of the domain name, prepend "dc=" to each, and join them with commas. So, for example, the DNS domain name `example.com` would map to the DN `dc=example, dc=com`. This method has the advantage that your domain name is already guaranteed to be unique because it was assigned by an Internet domain name registrar. Therefore, the suffix you derive from that domain name should also be unique, assuming that other organizations are following the same convention. Beneath the suffix entry, you are free to divide the namespace however you see fit. Netscape Directory Server 6 and Microsoft Active Directory both use this method as their default for naming suffixes, although both allow you to override the default and invent your own suffix.

The second common method is to use your organization's DNS name. If your organization's domain name is `example.com`, then the suffix for your directory will be `o=example.com` (the `o` attribute signals that the entry probably has the organization object class). This method also has the benefit of leveraging your already unique domain name. However, it deviates from the standard set by RFC 2247, and we do not recommend it.

The third method is to use the X.500 model for choosing your suffix. In this model you choose a suffix that can plug into the global X.500 directory. In the United States, the X.500 hierarchy has the `c=US` entry at the top, with entries for each of the U.S. states and territories directly beneath `c=US`. Organizational entries reside beneath each state entry. The

RDN of the suffix entry is named with the `o` (organization) attribute, and the name of the company, as registered with the state or federal government, is used.

If the company is named Example Electronics, Inc., and is incorporated in the state of Delaware, the suffix will be `o=Example Electronics\, Inc., st=Delaware, c=US`. This DN is cumbersome for two reasons. First, it is long (47 characters versus 18 for the RFC 2247–derived suffix). Second, the RDN contains a comma, which must be escaped. Although good client software hides DNs from users, directory administrators frequently need to type them. For these reasons, we recommend against using X.500-style suffixes unless you know that you need to participate in the global X.500 directory.

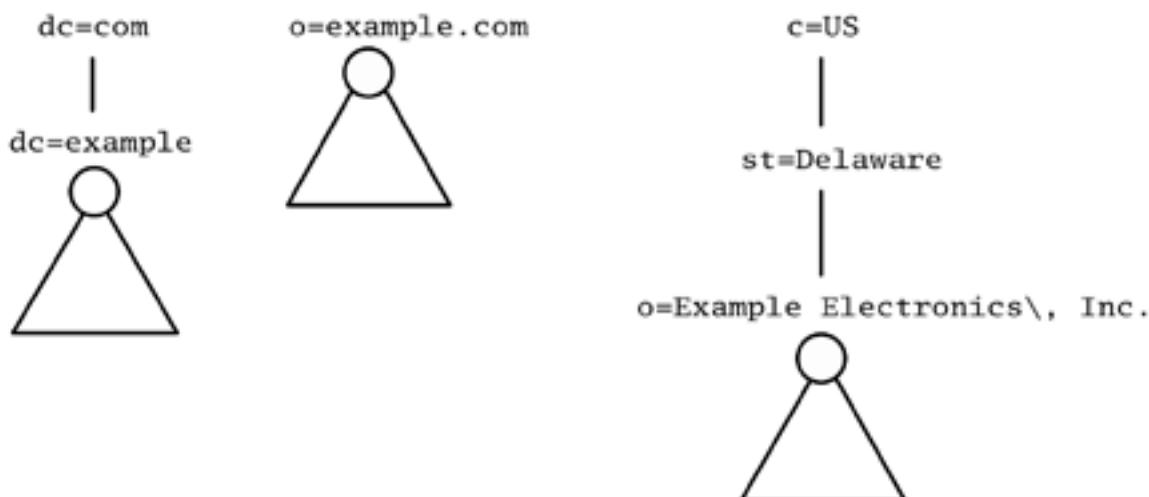
How Suffixes Work

It may help you to understand how a suffix is used by a directory server when the server is servicing a typical directory operation. For example, suppose that a client wants to modify the entry `uid=bjensen,dc=example,dc=com`. When a Netscape Directory Server 6 server receives a modification request, it compares the DN in the request to the directory suffixes it holds to determine whether the entry to be modified is beneath one of the suffixes held by the server. If the server holds the suffix `dc=example,dc=com` or the suffix `dc=com`, the modification proceeds.

If the server does not hold either of these suffixes, it can do one of three things. It might refer the client to a different directory server that does hold the requested data. It might forward the operation to the directory server that holds the requested data. Or the directory server might simply return an error, assuming that the requested entry does not exist. The specific behavior depends on the directory's configuration.

If you are designing a namespace for your department, which is only one part of a larger tree designed for the company, your suffix will name the entry at the top of your department's tree. For example, the Engineering department at Example Electronics might choose the suffix `ou=Engineering,dc=example,dc=com`. [Figure 9.8](#) shows examples of suffixes.

Figure 9.8. Examples of Directory Suffixes



Suffix: dc=example, dc=com o=example.com o=Example Electronics\, Inc., st=Delaware, c=US

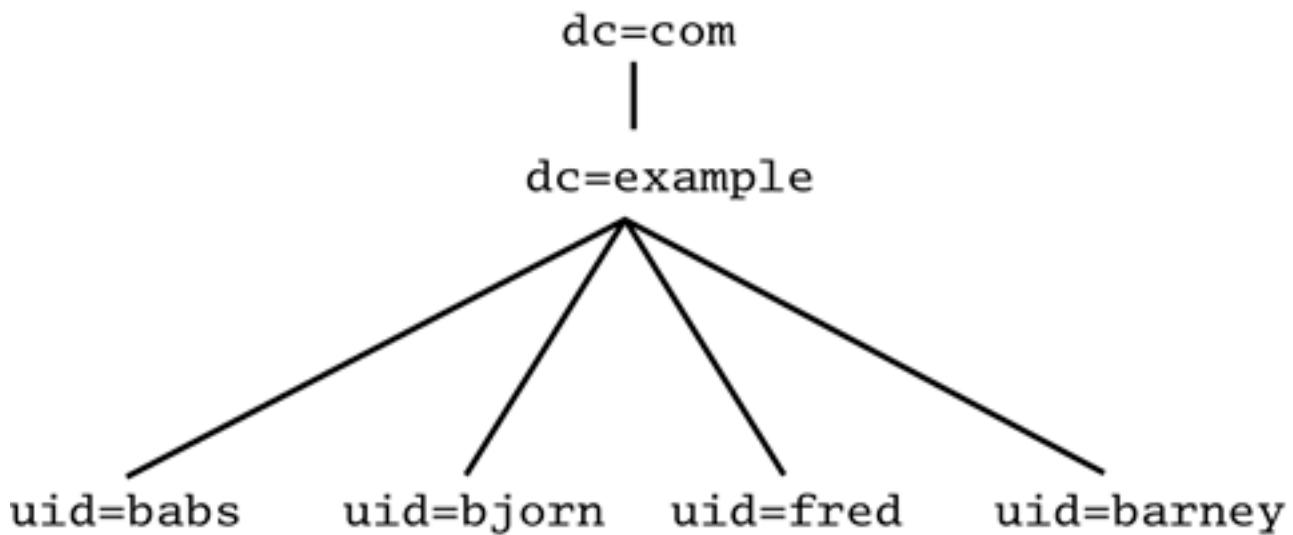
Often a directory server may hold more than one suffix. You may want to use this capability to design a service with multiple suffixes if you have two or more directory trees of information that do not have a natural common root.

Flat and Hierarchical Schemes

The next choice you need to make when you're designing a namespace is whether to use a flat or hierarchical scheme, and if you choose a hierarchical scheme, what type of hierarchy to construct. Of course, this is not a binary decision; your real decision is how much hierarchy and what type to introduce. As a guiding design principle, you should strive to make your namespace as flat as possible.

Name changes are typically one of the more burdensome administrative tasks of running a directory, inconvenient for both administrators and users. The flatter a namespace is, the less likely names are to change. All other things being equal, one would expect the likelihood of a name change to be proportional to the number of components in the name with the potential for change. The more hierarchical a namespace is, the more components it has and the longer the names are. The longer a name is, the more likely it is to change. Thus, shorter, flatter names will change less frequently. [Figure 9.9](#) shows an example of a flat namespace that requires only short names.

Figure 9.9. A Flat Namespace That Minimizes Name Changes



Tip

Make your namespace as flat as possible, while still meeting your other needs concerning topology, replication, and access control. Flat names change less and are easier to administer. Long names introduce needless complexity and administrative burden.

Of course, there are equally valid reasons to introduce a certain amount of hierarchy into a namespace. As described in the previous section, hierarchy may be required to enable data partitioning among multiple servers, replication, and certain kinds of access control. In addition, hierarchy can be useful to applications that want to browse the directory, although these applications are often better served by construction of virtual directory browsing views using attributes such as `seeAlso`, which can refer to other directory entries.

If you anticipate a centralized directory small enough to exist on a single machine, there is no need to introduce hierarchy to enable data distribution. Such a directory is not as constraining as it may sound. An average-sized Pentium-class machine can handle a directory on the order of millions of entries (depending on your directory implementation, of course), and it could be replicated to several other machines to handle additional client search load.

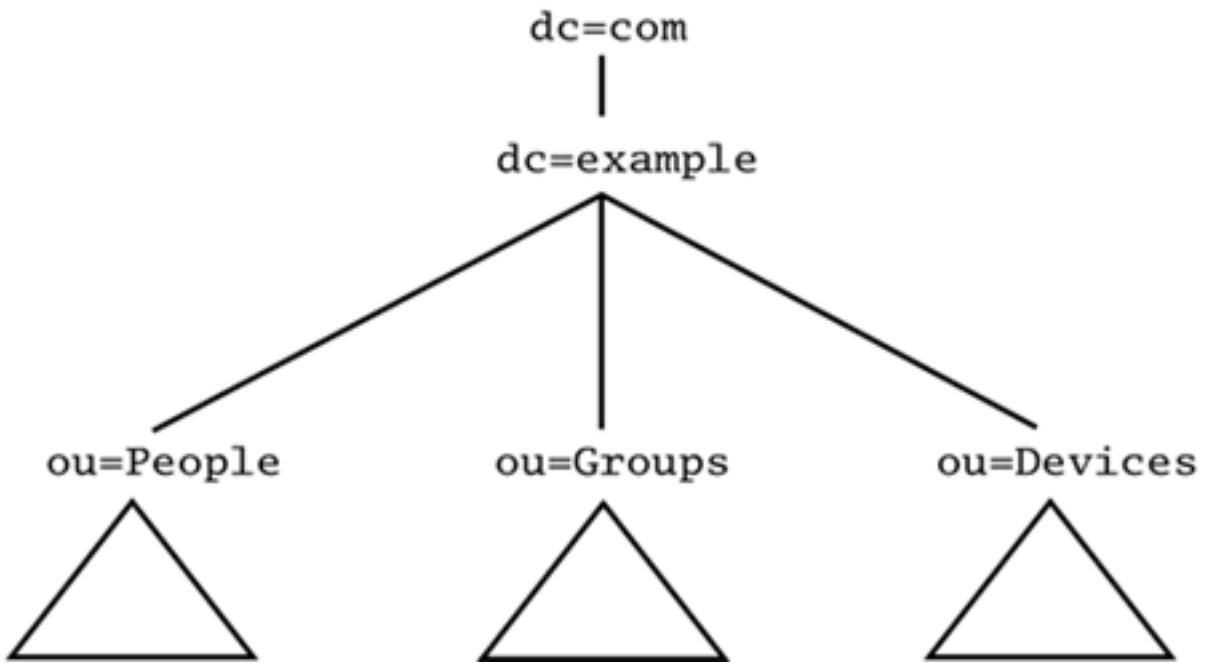
Another reason to introduce hierarchy is to enable the distribution of administrative authority via access control. For example, suppose that you want to allow an administrator from the Marketing department to have control over marketing entries, the Engineering administrator to have control over engineering entries, and so on. Many directory products allow this kind of administrative distribution of control only at branch points in your namespace. With such a system, different access control rules cannot easily be applied to the same subtree.

Some modern systems allow the setting of access control on the basis of directory content rather than the directory namespace. With Netscape Directory Server 6, for example, you can define a single access control rule stating that the Engineering administrator has access to all entries that have an attribute value indicating that they belong to the Engineering department (for example, `ou=engineering`). Carefully examine your chosen directory implementation's access control capabilities to make sure you understand how they will restrict your namespace design, or look for software that supports your preferred design.

If you do need to introduce hierarchy in your namespace, try to do so sparingly and in a way that avoids problematic name changes as much as possible. The reason for needing hierarchy in the first place may nullify much of your flexibility. For example, if you need hierarchy to distribute authority to different departments, there is not much hope in avoiding a name change when a user changes departments.

However, name changes can be avoided if you are able to design your hierarchy on the basis of information that is not connected to directory information that is likely to change. For example, you could base your hierarchy on the types of objects in each tree, with one area of the tree for people, another for groups, and so on. It is unlikely, to say the least, that an entry would need to move from one area of the hierarchy to another with a scheme like this. This kind of partitioning can make replication easier in some cases as well. [Figure 9.10](#) shows an example of this kind of hierarchical namespace.

Figure 9.10. A Hierarchical Namespace in Which Data Is Not Likely to Change



Naming Attributes

After you've decided on the basic structure of your namespace and the level of hierarchy, you need to decide on attributes to use when you're naming entries. The attribute you should use depends on the type of entry you are naming and other requirements at your site. In this section we present some general principles that you can apply to naming all kinds of entries.

The only requirements imposed on naming attributes by the LDAP model are these:

- The RDN of the entry must be chosen from one or more of the entry's attributes.
- The RDN must be unique among all its sibling entries (other entries that have the same parent).

Although these are the only restrictions imposed by the LDAP model, we suggest that you adopt a policy ensuring that all RDNs for people are unique across your entire directory. This policy has the benefit that, even if you need to move an entry to a new location in your namespace, its name will not clash with another entry's name.

To meet this additional restriction, there are two approaches you can take: (1) You can name entries using an existing name that is already guaranteed to be unique, or (2) you can generate your own unique names for entries. We discuss these two approaches next.

Naming Entries by Using Existing Unique Names

Your organization may already have a method for assigning unique names to users. Many companies assign a unique user ID to an employee when she is hired, and the employee uses this ID to log in to various computing services throughout the company. If the user IDs assigned by this organizational process are known to be unique, they can serve as the naming attributes for users in your directory.

If you are fortunate enough to have such a method already deployed, we suggest that you use the unique name assigned by this process as the `uid` attribute of each user's entry, and that you name the entry with the `uid` attribute.

For example, suppose that during the process of being hired at Example Electronics, Inc., Barbara Jensen chose the login name "babs." Using our suggestion, the `uid` attribute of Barbara's entry would be `babs`, and the entry would be named by that attribute. Assuming that Barbara works in the Engineering department, that a separate branch exists for that department, and that RFC 2247-style naming is in use, her entry's DN would be `uid=babs, ou=Engineering,dc=example,dc=com`.

Because the login ID "babs" has been guaranteed to be unique by an external organizational process, we can be certain that no other "babs" will end up in another branch of our directory. Thus we will not need to worry about name clashes if Barbara moves to a different department within the company.

Be careful when choosing existing unique identifiers for users. Some naturally occurring naming attributes may be sensitive, and you may not want to use them for fear of unintentionally revealing information that should not be revealed. A U.S. Social Security number is a good example of such an attribute. If you use it to name your people entries, you guarantee uniqueness—but at the expense of publishing everybody's Social Security number, a practice guaranteed to make you highly unpopular.

Naming Entries by Constructing New Unique Identifiers

If your organization does not currently have a process for assigning login names to people, you have more work to do.

You might consider creating such a process yourself. Creating a unique login ID for each new employee is not all that difficult, and it is beneficial. You can guide users in choosing a unique ID by allowing them to propose an ID and checking whether the name is in use by searching the directory. If the name is not in use, the user's entry can be created. Otherwise, the name clash is reported to the user, and a new name can be chosen.

Another approach to guaranteeing uniqueness is to artificially make the attribute you have chosen unique, perhaps by appending a number. For example, suppose that you choose the `cn` attribute to name entries and have two entries with the same parent that would otherwise both be named `cn=Barbara Jensen`. You could append a number to one or both of the names, making `cn=Barbara Jensen 1` and `cn=Barbara Jensen 2` the names of the two entries.

Although this approach may have some aesthetic value, it is also more difficult to maintain. In our experience, users generally dislike having their names changed in any way, even for such a clear administrative reason. It may well be better to use something more arbitrary with no value to the user. This scheme may be more difficult to maintain because it requires an external mechanism to manage the process of making names unique. Be sure to pilot any user-naming decisions with your user community; it's often difficult to predict what they will like and dislike. This is another good reason to hide DNs from your users.

The LDAP model also allows the use of multiple attributes from an entry to form a multivalued RDN. The idea behind this capability is to use the additional attributes to distinguish entries that otherwise would have the same name. For example, suppose that you have two users named Barbara Jensen—one in the California office and the other in the Michigan office. Using multivalued RDNs, you could distinguish between these two entries by naming one `cn=Barbara Jensen + L=California` and the other `cn=Barbara Jensen + L=Michigan`.

This practice tends to lead to long, complicated names that change frequently (what if either

Barbara moves?). Also some directory implementations, such as Netscape Directory Server 6, do not fully support multivalued RDNs. For these reasons, we strongly discourage their use and encourage you instead to use one of the other naming conflict resolution strategies discussed.

A final approach you might consider is to make up a meaningless identifier that is unique. For example, you might generate a random number and use that as the `cn` attribute for a new entry. Although this approach has no aesthetic appeal, it meets the requirements for uniqueness, and if your LDAP clients hide entry names from users, there's no reason not to adopt this approach.

Application Considerations

Most people do not design and run a directory service for its own sake. Typically, the directory is required to support one or more directory-enabled applications. The requirements these applications place on the namespace and other aspects of your design are important design considerations. After all, if your directory does not satisfy the requirements of the applications driving its deployment, your chances of postdeployment employment are small.

Multivalued RDNs and Client Complexity

Multivalued RDNs pose a difficult problem for LDAP client software writers. Often LDAP client software needs to compare two DNs for equivalence. Multivalued RDNs make this difficult because each RDN is a set, according to X.500 standards (on which the LDAP standards are based). In mathematical terms, a set is an unordered list of items. This means that the individual attributes that make up a multivalued RDN may appear in any order in the RDN. For example, the name `cn=Barbara Jensen + L=California,dc=example,dc=com` refers to the same entry as the name `L=California + cn=Barbara Jensen,dc=example,dc=com`. Clients that need to compare DNs need to be able to understand this. In our experience, few clients properly handle this situation. This is another reason we recommend that you avoid using multivalued RDNs.

The requirements that an application can place on your directory are as varied as the applications themselves. Lest you become dismayed and think that anticipating the needs of an endless parade of different applications is a lost cause, consider the following.

First, focusing on the needs of directory applications existing or being deployed in your organization today will probably provide you with a fairly representative cross section of requirements. Make sure that you understand these needs as well as possible before you consider yourself finished with your directory design (see [Chapter 6](#), Defining Your Directory Needs). Piloting your directory on a smaller, test-scale deployment is also a good idea.

Second, some general principles you can follow will help prepare you for that future parade of directory-enabled applications. These principles are important to keep in mind both when you're designing your directory and when you're writing a directory-enabled application. For more information, see [Chapter 21](#), Developing New Applications.

A well-written directory-enabled application makes a concerted effort to assume as little as possible about the directory service it will access. An application should be configurable and capable of adapting to new namespaces, new types of acceptable queries, schema differences, nonstandard port numbers, new host names, and more. Of course, not all

applications can provide this kind of flexibility. How can you design your namespace to anticipate as many of these problems as possible?

If your existing needs allow it, one good approach is to use a standard namespace design, such as the RFC 2247 style of naming we described earlier in this chapter. Because this naming method is documented in a standards document, it's likely that application vendors will support it.

Another good approach is to be conservative when picking the attribute used to name entries. Try to use a standard attribute such as `cn` or `uid`. If you're considering creating a new attribute to hold the naming value—for example, `employeeID`—consider placing the value of the `employeeID` attribute in the `cn` or another standard attribute, and then use that attribute for naming instead of `employeeID`. Although this approach might seem aesthetically unpleasant, applications that assume standard attributes for the namespace will not become confused.

Namespace and other directory design choices like this are common. When starting from scratch, you can often afford to make things aesthetically pleasing as well as functional. But it is rare, unfortunately, that you will be able to start completely from scratch without worrying about any existing applications.

Administrative Considerations of Naming Attributes and RDNs

When designing your namespace, consider the effect the namespace will have on common administrative tasks. For example, when you're adding an entry, can the naming attributes be generated automatically, or is it a manual process? When entries are deleted, can their naming attributes be reused, or should they forever be reserved for the deleted entry? What effect will a name change have? How often are names likely to change? What other things depend on the namespace? The answers to these questions are seldom independent of the design decisions discussed in the previous sections. In this section we discuss the administrative implications of those decisions.

If your organization already has a unique identifier assigned to each user (for example, an employee number, login name, or user ID), it may make sense to use it as the value of the naming attribute. This saves you the administrative burden of devising and maintaining another unique identifier, and it is a good solution for naming user entries.

Other entries, perhaps for printers, groups, or other entities, are another matter. In either case, using an existing attribute type can eliminate another small administrative task: defining a new attribute type to use when naming entries. It also reduces the likelihood that a less-than-intelligent client could be confused by an unknown attribute.

Maintenance of the naming attribute is also a consideration. Whether or not the directory is the ultimate source of authority, the problem of reusability must be addressed. Depending on your policies, namespace identifiers like user login names might be (1) assigned only once and never reassigned, (2) reassigned after a suitable interval, or (3) reassigned immediately. Whatever policy you choose, it must be enforceable.

Most directory software does not support an out-of-the-box namespace reuse policy. Instead you have to enforce such a policy either through an external administrative agent or through an extension to the directory software itself. For example, if a user leaves your organization, you may want to avoid assigning the same login name to a new user for a few months. One way to do this is to mark the terminated employee's entry in a special way (possibly by marking it with a special `objectclass` attribute value and removing the password). Because the entry is still in the directory, it prevents the name from being reused, but it is impossible

to authenticate as that user. It's also useful to store the employee's termination date in the entry so that an automated task can clean up these deletion records.

Almost inevitably, names will change for various reasons. If you choose a naming attribute that has any significance other than its uniqueness, it can change. If you choose a naming attribute that is related to a real-world attribute of the entity being named, or if you choose a hierarchical namespace whose upper components could change, names will also change. The consequences of a name change should be considered carefully. How much trouble will the change cause, and what is the likelihood of its occurrence?

Tip

Beware of thinking that a name change will be the exceptional case, so rare that you would not mind handling such occurrences through even a tedious manual process. Our experience shows that such trouble has a habit of occurring more frequently than you might imagine. You also need to consider what will happen as your directory grows (a prediction likely to come true). What may seem uncommon in a small directory can become a downright nuisance in a large one.

Privacy Considerations

Directory names are usually public information available to anyone who can access the directory. Trying to control access to names via your directory's access control mechanism can often lead to difficulties. In practical terms, it's not possible to hide directory names from clients, no matter how advanced the access control capabilities in your server are. For this reason, you must carefully consider the privacy implications of your namespace design. Your goal should be not to divulge any information through the namespace that you do not intend to divulge.

For example, if you design a namespace for your people entries based on organizational hierarchy, you reveal the part of your organization in which an entry (and presumably the corresponding person) resides. The same problem holds true for many other hierarchical namespace designs.

As described earlier, the attribute after which you choose to name your entries may be considered sensitive. We saw an example earlier involving the use of Social Security numbers for naming attributes. Clearly, this would be a bad idea, so we suggested not using the social security number as a naming attribute.

There are other, more subtle privacy concerns as well. For example, using the `cn` attribute containing a person's name to name entries has a host of implications. A person's gender can often be inferred from his or her name, as can other information such as nationality or ethnicity. Not to mention the fact that a name is often enough to gain other information—such as an address, phone number, and so on—from other publicly available sources.

Care should be taken to protect privacy and to ensure that unwanted disclosure of information is minimized. Keep in mind that things obviously acceptable to you may be completely unacceptable to some of your users. For example, you might not mind disclosing your name or even your address to everyone in your company or the world. However, one of your users who might be concerned about potential harassment or stalking—or even worse—might feel differently.

Try to design a namespace that is free of such considerations, and be prepared to make exceptions for people who have legitimate concerns with any design you come up with. It may be a good idea to involve your Legal department to help interpret legal issues associated with directory information privacy. Directory privacy is covered in more detail in [Chapter 12](#), Privacy and Security Design.

Anticipating the Future

Finally, as difficult as it may be, you must try to anticipate the future when designing your namespace. The reason is simple: Redesigning a namespace is a costly and inconvenient process that you want to avoid. Because none of us have a crystal ball, the best we can do is try to avoid common situations in which namespace changes are required.

The question naturally arises, therefore, about the kinds of situations that precipitate a namespace redesign. Here are some of the more common situations:

- Choosing the wrong naming attribute can easily lead to a namespace redesign. For example, if you name entries with the `cn` attribute using a value of first name followed by last name, what do you do when two people have the same name? Either a namespace redesign is required or you must be prepared to artificially make one of the names unique, as described previously.
- If your directory starts out under central administrative control, but later you decide to delegate control of a portion of the data, a namespace redesign may be required. As mentioned earlier, some access control implementations do not allow delegation except at subtree boundaries. The same is true for replication and partitioning of the data.
- If you choose a hierarchical namespace with a hierarchy based on a geographical, organizational, or other scheme that is likely to change, constant namespace redesigns, both big and small, may haunt you. It is best to avoid this situation altogether from the start. If you choose to reflect your organizational hierarchy, for example, a namespace redesign will be required each time your company reorganizes. For some reorganization-happy companies, this can be a problem!

Although no one can accurately predict the future, you can use some defensive namespace design tactics to minimize your risk. Choosing a flat namespace is one such tactic. Subdividing your namespace on the basis of unchanging information—perhaps into areas for people, groups, and devices—permits redesigns in one space that do not affect the others.

Examples of Namespaces

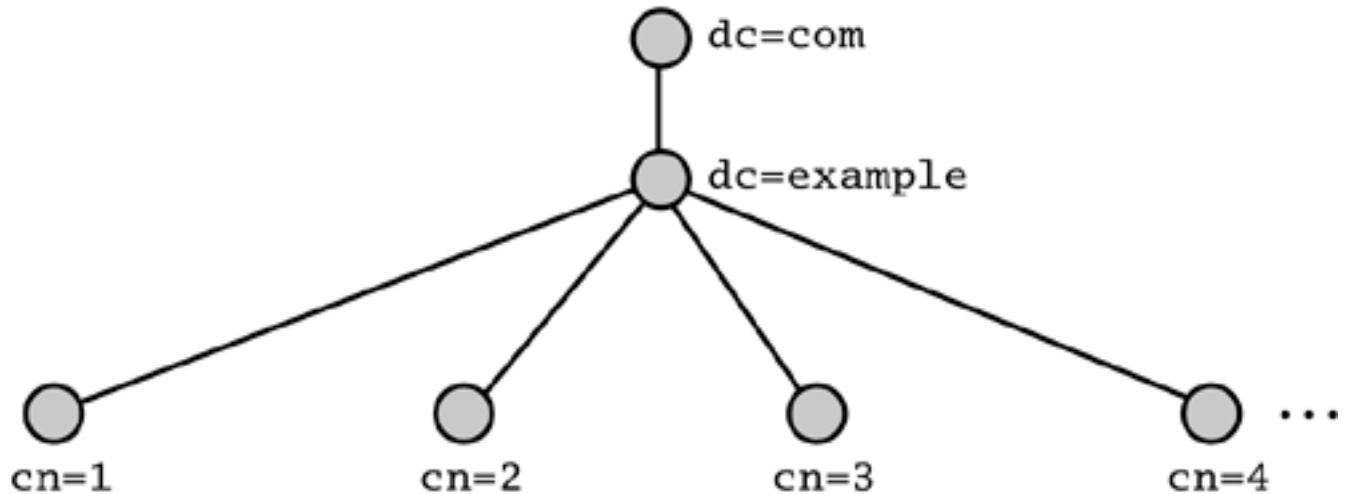
Now that we've described the purposes of a namespace and the major decisions that must be made when designing one, we turn our attention to some examples that should help illustrate the points we've been discussing. Here we consider both flat and hierarchical namespaces. (More design examples can be found in [Part VI](#), Case Studies.)

Flat Namespace Examples

We advised you to design your namespace to be as flat as possible within the constraints created by your replication, access control, and other needs. This section describes two examples of flat namespaces for the fictitious company named Example Electronics.

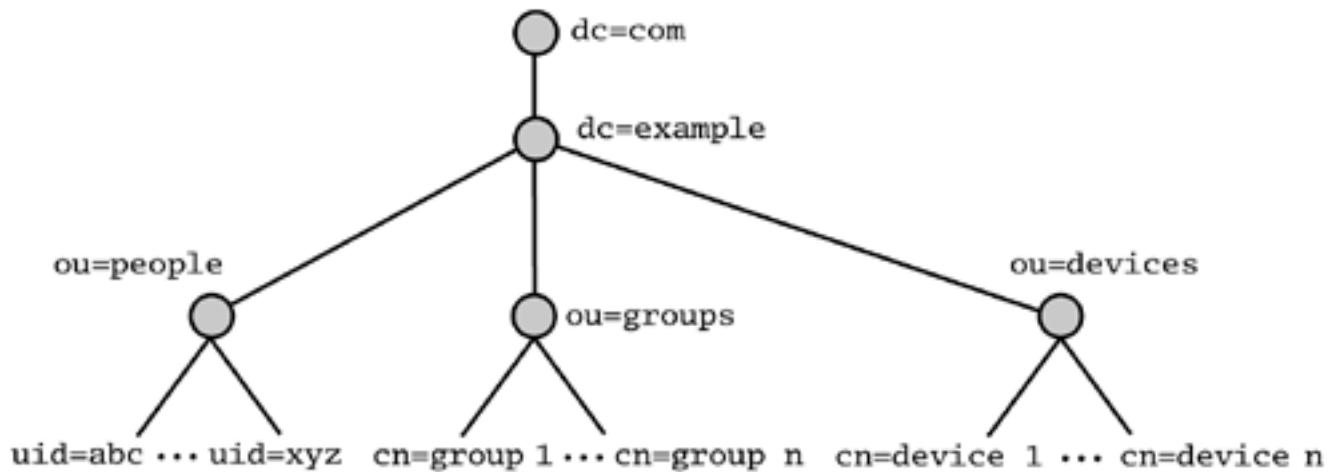
In the first example, a completely flat space is created. For simplicity, the naming attribute is `cn`. Each entry's name is a sequentially generated number that has no other significance. [Figure 9.11](#) shows the resulting namespace.

Figure 9.11. A Completely Flat Namespace



Our second flat namespace example introduces a bit of hierarchy, with a flat namespace beneath each hierarchical component. The small amount of hierarchy in this namespace is used only to group different types of objects under a common portion of the tree. This arrangement has the effect of insulating namespace changes in one space from the others. It also makes it possible to partition and therefore replicate information on these same boundaries. Administrative control can also be distributed on these boundaries. [Figure 9.12](#) shows the resulting namespace.

Figure 9.12. A Flat Namespace with Some Hierarchy



In our experience, a namespace like the one shown in [Figure 9.12](#) is a good choice for smaller directory deployments. It has enough hierarchy not to hinder you when your needs change (for example, in replication or access control); however, the hierarchy it does have is not based on information likely to change (for example, geographical or organizational information).

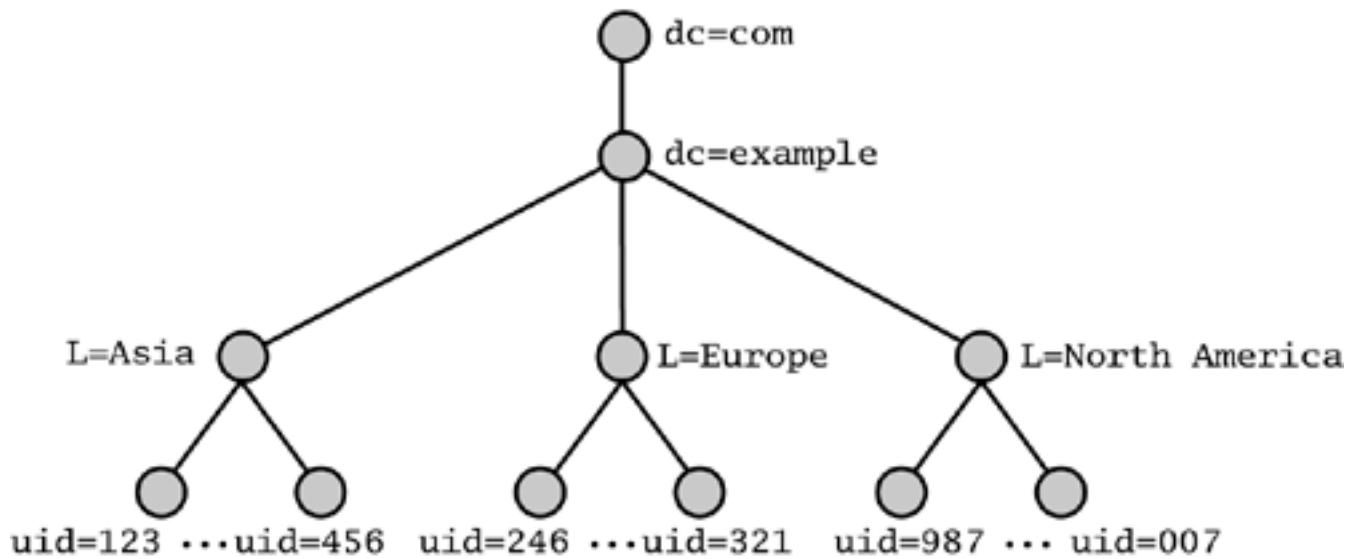
Hierarchical Namespace Examples

You may need a hierarchical namespace because of limitations in the directory software you've chosen, constraints imposed on your directory by applications, or other reasons. This section illustrates two examples of hierarchical namespaces based on different hierarchical schemes. Again, we use the fictitious company Example Electronics to illustrate our design.

For the first hierarchical example, assume that Example Electronics is a global company distributed across three continents. It has an office on each continent that must have control over its own data, yet the sum total of the data must appear to the outside world as a single, coordinated information tree.

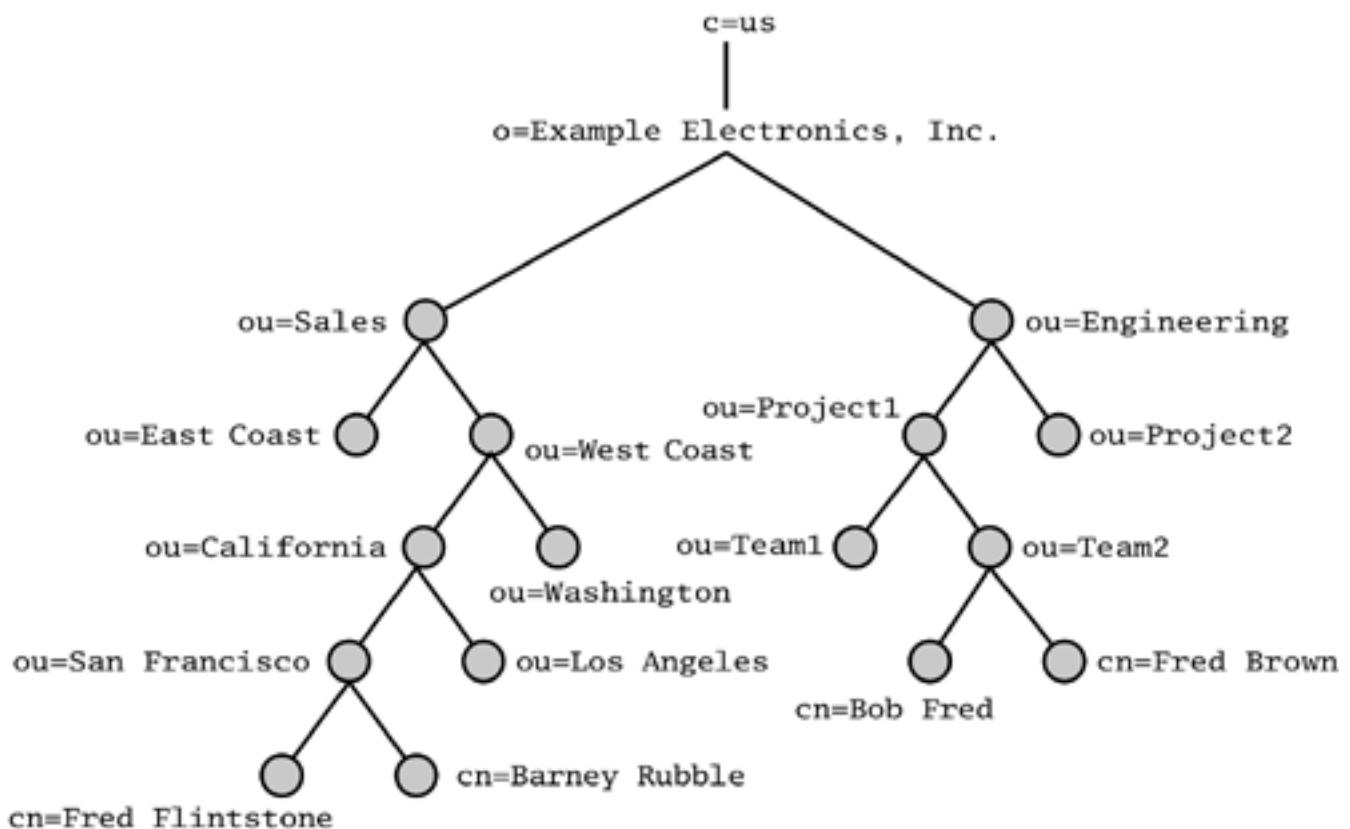
Taking a lesson from the flat namespace example, the Example Electronics directory designer decides to use a similar namespace in each of its divisions. Because divisions need autonomy in their ability to manage, access, and change their own data, the top-level namespace hierarchy is divided according to a geographical scheme. [Figure 9.13](#) shows the resulting namespace.

Figure 9.13. A Hierarchical Namespace Based on a Geographical Scheme



Our second example of a hierarchical namespace is based on Example Electronics' internal organizational structure and uses the X.500 naming model. Although you might be able to rationalize building such a namespace, we don't recommend it. But in this example the misguided directory designer has decided to forge ahead. To reflect the organizational hierarchy, he's chosen to use the `ou` attribute with values corresponding to department names. To name leaf-level entries, the `cn` attribute has been chosen with a value of first name and last name. In cases of name collisions, a number is appended to entries that collide. [Figure 9.14](#) shows the resulting namespace.

Figure 9.14. A Hierarchical Namespace Based on Organizational Structure



The examples presented in this chapter, combined with the guiding principles we've discussed, should help you arrive at the best possible design for your directory's namespace.

Namespace Design Checklist

The following checklist summarizes the issues you should consider when designing your namespace:

- Consider the number of entries your directory will need to support, and try to predict the rate at which that number will increase.
- Consider the types of entries your directory will store. Decide whether the entries are all the same or of different types and whether new types will likely be added in the future.
- Determine whether your directory data needs to be centralized or distributed. When choosing a directory product, make sure that it supports your choice.
- Consider the requirements for delegating authority to different parts of your directory and how they affect the design of your namespace.
- Consider the effects of replication requirements on your namespace design.
- Keep firmly in mind the applications that will access your directory and any special requirements they may have.
- Think carefully about the attribute you will use to name entries and how best to maintain it. Using the `uid` or `cn` attribute to name entries for people is standard practice.
- If possible, leverage existing organizational processes to generate unique identifiers for your users instead of inventing your own.
- Design a system for ensuring and maintaining uniqueness of names in your directory.
- Formulate a name reuse policy that determines whether, when, and how names may be reused.

Further Reading

Lightweight Directory Access Protocol (v3) (RFC 2251). M. Wahl, S. Kille, and T. Howes, 1997. Available on the World Wide Web at <http://www.ietf.org/rfc/rfc2251.txt>.

Lightweight Directory Access Protocol (v3): UTF-8 String Representation of Distinguished Names (RFC 2253). M. Wahl, S. Kille, and T. Howes, 1997. Available on the World Wide Web at <http://www.ietf.org/rfc/rfc2253.txt>.

Naming and Structuring Guidelines for X.500 Directory Pilots (RFC 1617). P. Barker, S. Kille, and T. Lenggenhager, 1994. Available on the World Wide Web at <http://www.ietf.org/rfc/rfc1617.txt>.

Naming Guidelines for the AARNet X.500 Directory Service (RFC 1562). G. Michaelson and M. Prior, 1993. Available on the World Wide Web at <http://www.ietf.org/rfc/rfc1562.txt>.

Using Domains in LDAP/X.500 Distinguished Names (RFC 2247). S. Kille, M. Wahl, A. Grimstad, R. Huber, and S. Sataluri, 1998. Available on the World Wide Web at <http://www.ietf.org/rfc/rfc2247.txt>.

Looking Ahead

Now that we've covered the topic of namespace design, it's time to turn our attention to an even more exciting aspect of directory service design: server topology. This topic will be covered in [Chapter 10](#).

Chapter 10. Topology Design

- Directory Topology Overview
- Gluing the Directory Together: Knowledge References
- Authentication in a Distributed Directory
- Advantages and Disadvantages of Partitioning
- Designing Your Directory Server Topology
- Topology Design Checklist
- Further Reading
- Looking Ahead

LDAP directory services are designed to support a distributed directory, in which the complete directory tree is spread across multiple physical directory servers. Your directory service's topology describes the way you divide your directory tree among physical servers and how you allocate those servers among your organization's physical locations. Making good choices about your directory topology will help you

- Achieve the best possible performance for your directory-enabled applications
- Increase directory availability
- Better manage your directory

In this chapter we first examine how a distributed directory works. We discuss several important background concepts, including the definition of a directory partition, name resolution in a distributed directory, and how separate partitions are joined into a single directory using referrals or chaining. Finally, we discuss why you might or might not want to partition your directory, and we work through two sample scenarios.

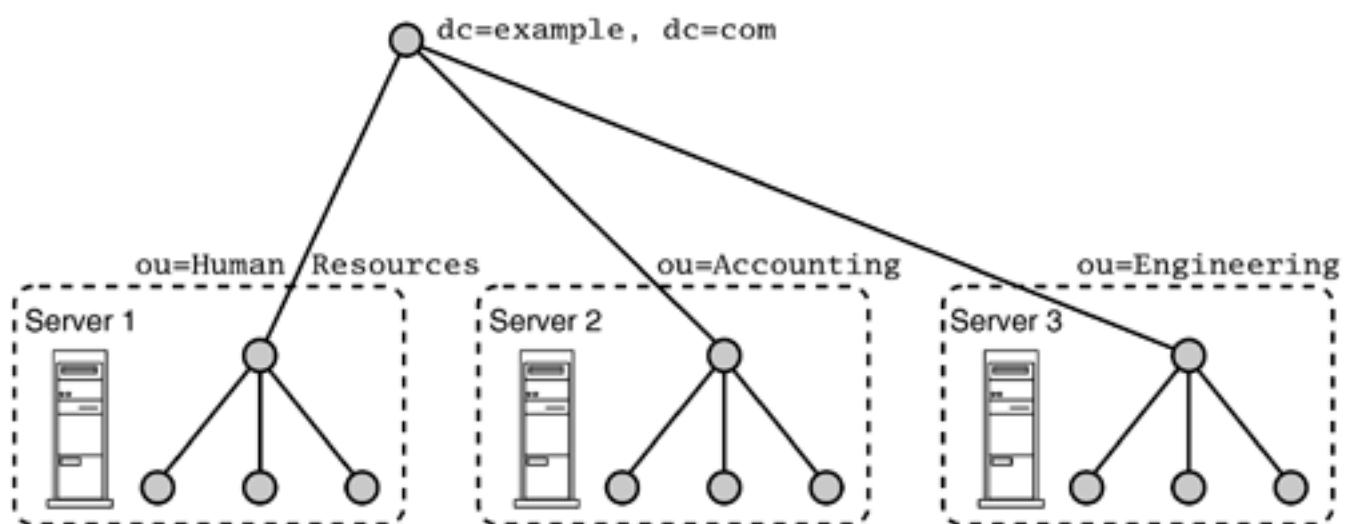
This chapter is closely related to [Chapters 9](#), Namespace Design, and [11](#), Replication Design. In fact, we suggest that you read all three chapters as a unit because decisions concerning namespace design have direct consequences for your directory topology, which in turn has a direct bearing on your replication strategy. After you read this chapter, you'll know how to partition your directory service for maximum performance and manageability and how to allocate servers across your LAN or WAN (local or wide area network). This knowledge, along with the information presented in [Chapters 9](#) and [11](#), will help you design a highly robust, high-performance directory service.

Directory Topology Overview

A directory service can be asked to store a potentially large number of entries. We have encountered directories with up to 100 million entries. Depending on your directory software, that number may be more than one server can reasonably be expected to hold. To enable a directory to hold such large numbers of entries, it may be necessary to divide the directory database across multiple servers.

A directory that resides on more than one server is a *distributed directory*. When you carve a single directory into manageable chunks and assign them to separate servers, you are *partitioning* the directory. For example, a large corporation might choose to partition its directory as shown in [Figure 10.1](#).

Figure 10.1. A Distributed Directory



The dotted lines surrounding each server computer in [Figure 10.1](#) indicate that the partition resides on that particular server. We will use this convention throughout this chapter.

When the directory tree is divided among multiple servers, each server is responsible for only a portion of the tree, which reduces the amount of work it needs to do. Using this principle of dividing a directory namespace into partitions and assigning those partitions to separate servers, a directory can scale to a much larger number of entries than would be possible with a single server. The Domain Name System (DNS) operates similarly, with each portion of the DNS namespace (for example, `example.com`) assigned to a particular DNS server that may be replicated to improve availability.

Note

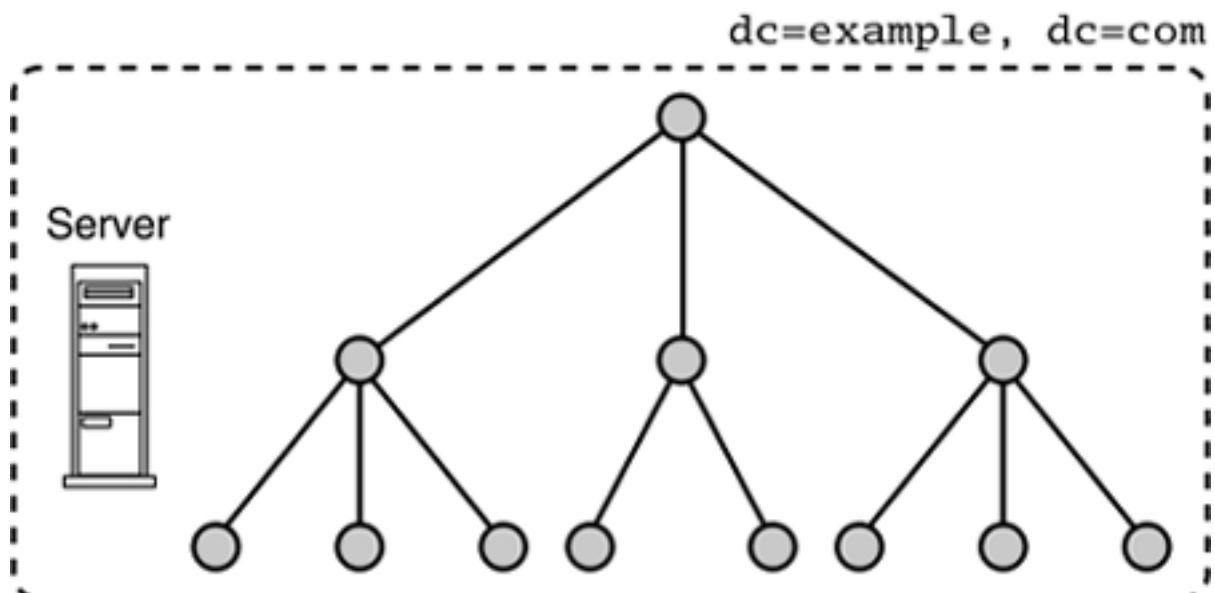
The unit of division is known by several different names, depending on the directory server software you are using or the standards documentation you may be reading. Whereas Novell eDirectory uses the term *directory partition*, the X.500 standards documents use the term *naming context*. Netscape Directory Server 6 calls these units *databases*, and Microsoft Active Directory uses the term *domain*. All these terms mean essentially the same thing, but we'll use the term *directory partition* throughout this chapter.

An important point to remember is that the directory itself is responsible for hiding all these partitioning details from the user. As far as users and applications are concerned, a single directory answers their directory queries. The mechanics of how these details are hidden from users are discussed in detail later in this chapter. For now, simply remember that the various partitions are glued together into a single, logical directory tree from the client's or application's point of view.

Definition of a Partition

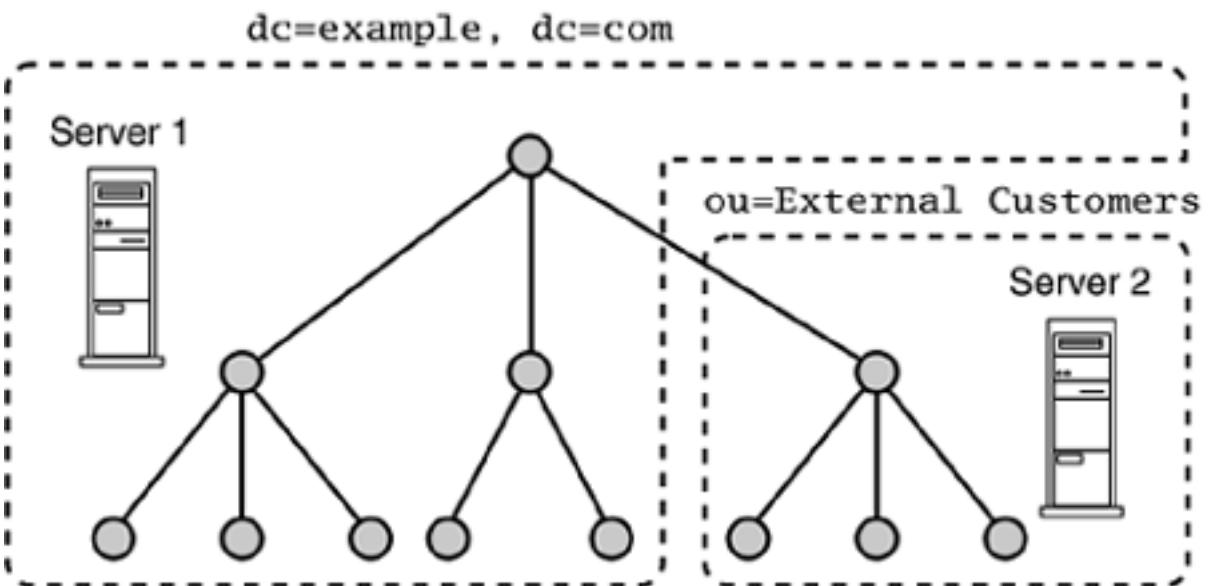
A *directory partition* is a complete subtree of the directory information tree (DIT), minus any subtrees held within other partitions. A given directory entry resides in only one directory partition, and all entries within a partition must share a common ancestor known as the *partition root*. [Figure 10.2](#) shows a basic directory partition with a partition root of `dc=example,dc=com`. The partition, denoted by the dotted line, extends downward from the partition root (`dc=example,dc=com`) and does not exclude any entries. In other words, it is a complete subtree.

Figure 10.2. A DIT Contained in a Single Partition



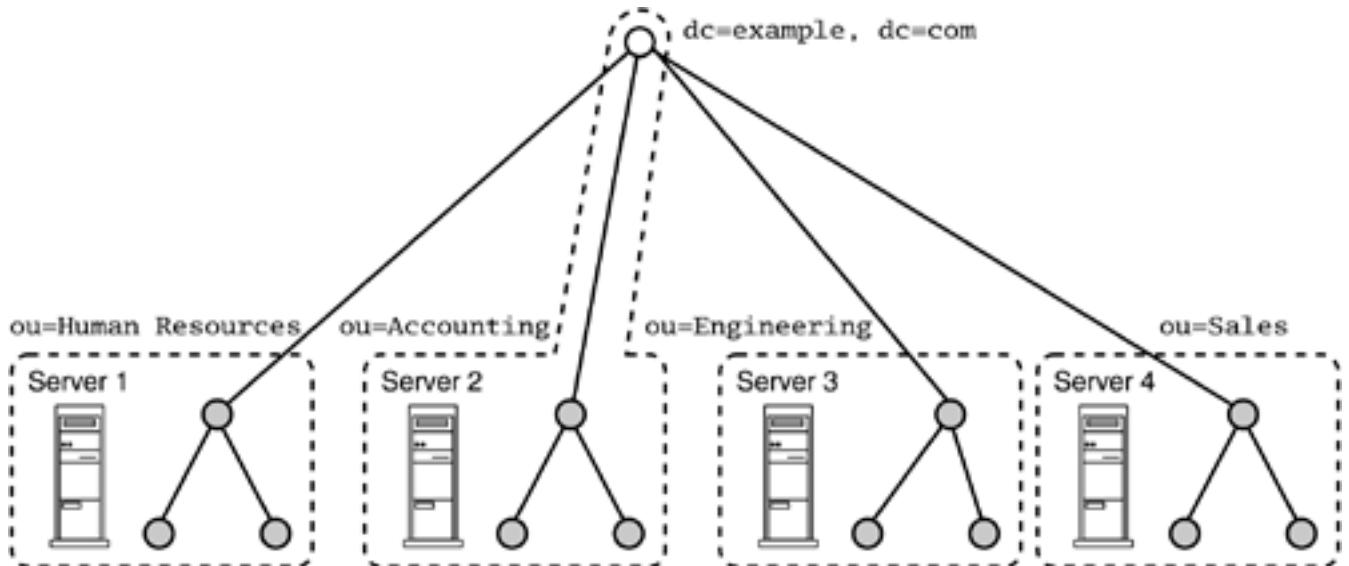
It is also possible to selectively exclude subtrees from a partition. In [Figure 10.3](#) there are two partitions. One is rooted at `dc=example,dc=com` and includes all entries beneath `dc=example,dc=com` except for those in the other partition. This second partition contains the entry `ou=External Customers,dc=example,dc=com` and all entries beneath it. This partitioning arrangement would allow the subtree `ou=External Customers, dc=example, dc=com` to reside on a different server from the rest of the directory tree.

Figure 10.3. A DIT Split into Two Partitions



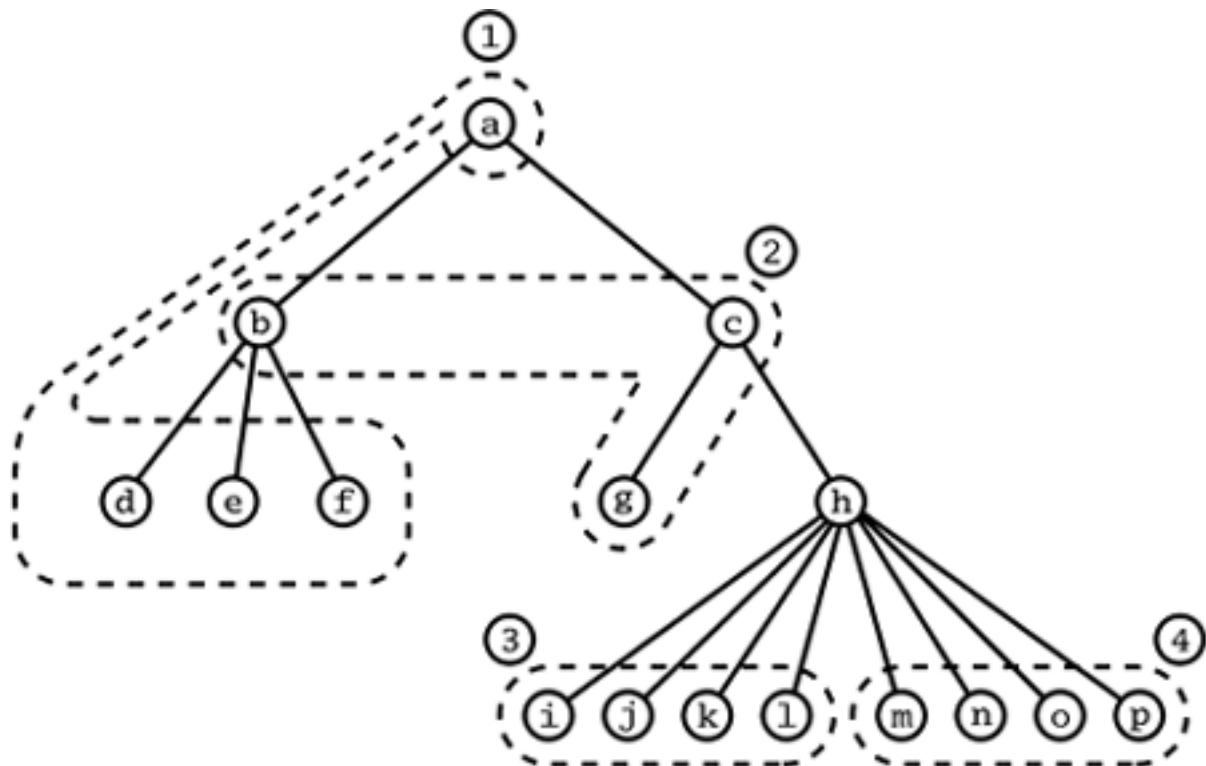
Using this principle, you can divide a single large directory tree into multiple smaller partitions. Each partition can be assigned to a separate server, if required, either to handle the client load or to accommodate limits on the number of entries that can be held by a server. For example, `example.com`'s directory could be divided into partitions and assigned to four servers, as shown in [Figure 10.4](#).

Figure 10.4. A DIT Partitioned across Four Servers



To further clarify the concept of a directory partition, let's also look at some illegal directory partitions. In [Figure 10.5](#), Partition 1 is invalid because it contains a "hole": Entry **b** is missing from the partition. Partition 2 is invalid because it is not a proper subtree: Not all the entries in the partition share a common ancestor. Partitions 3 and 4 are invalid for a similar reason as Partition 2: Although all entries do share a common ancestor, the ancestor is not contained within the partition.

Figure 10.5. Examples of Illegal Partitions



Although we've shown only examples in which a given server holds a single partition, this need not be the case. A server can actually hold many directory partitions. For example, a directory server might hold a read-only copy of the top-level partition, along with the master copy of a particular organizational unit's partition. Alternatively, an Internet service provider might choose to offer directory services to corporate clients and deploy several "virtual" directories on a single server—one that has sufficient RAM, CPU, and disk resources, of course.

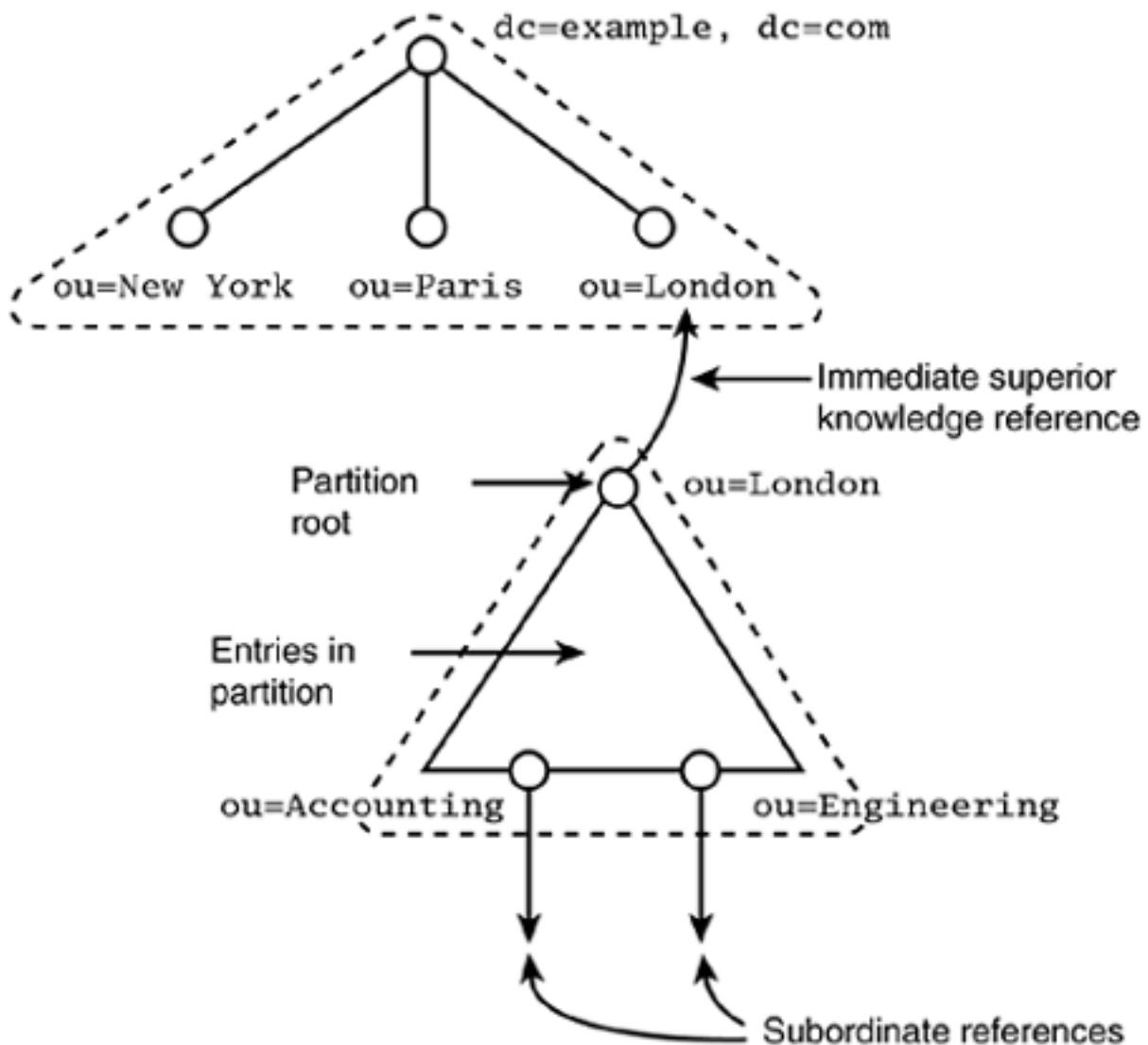
Gluing the Directory Together: Knowledge References

So far, we've described directory partitions in terms of the entries they contain. However, we still need some way to describe the relationships between directory partitions. The LDAP and X.500 standards define these relationships in terms of *knowledge references*—pointers to directory information held in another partition. There are two types of knowledge references:

1. **Immediate superior knowledge references.** This type of knowledge reference points upward in the DIT toward the root and ties the directory partition to its ancestor. In effect, it provides the "hook" at the top of the directory partition that you use to hang a partition from its ancestor.
2. **Subordinate references.** These knowledge references point downward in the DIT to other partitions. They are the hooks onto which you hang other directory partitions.

[Figure 10.6](#) illustrates how directory partitions fit together into a larger DIT. In this example, the partition root `ou=London,dc=example,dc=com` denotes the top of the partition. The partition contains the partition root and all the entries underneath it, except for those entries held in partitions pointed to by the two subordinate references, `ou=Accounting,ou=London,dc=example,dc=com` and `ou=Engineering, ou=London,dc=example,dc=com`. An immediate superior knowledge reference also points upward in the DIT, connecting this partition to its parent partition. Knowledge references always come in pairs; although not shown, the parent partition has a subordinate reference pointing to the partition in the figure.

Figure 10.6. A Directory Partition in a DIT



Note

Directory servers based on the LDAPv3 standard advertise the partitions they hold in a special entry called the *root DSE* (*DSA-specific entry*; *DSA* stands for *Directory System Agent* and is the X.500 term for a directory server). As explained in [Chapter 2](#), Introduction to LDAP, the root DSE is a special entry that contains information about the directory server, including its capabilities and configuration.

To discover the partitions held by a given server, you can search for the entry with a zero-length name, with a scope of `base` and a search filter of `(objectclass=*)`. The command `ldapsearch -h host -s base -b "" "(objectclass=*)"` will display the root DSE on the host named `host`. The attribute `namingContexts` of the returned entry lists the partition root of each partition held by the server.

Your directory server software may use these same terms, or it may have an alternative way of talking about how partitions are named and related. But in virtually all cases, this model is a good basis for understanding how your server software can be used to construct real-world directories.

Dividing a directory in this fashion isn't useful all by itself. If we just left the partitions as a

series of isolated islands, locating an entry or searching across a series of partitions would be tedious. The user of the directory would need to know which servers to contact to perform an operation that spanned multiple partitions. Clearly, directory partitions need to know about each other to work together as a single, cohesive unit. In the next section we discuss how directory partitions work together to appear as a single unit.

Name Resolution in the Distributed Directory

Now that you understand partitions and knowledge references, the basic building blocks of a distributed directory, let's examine an important concept: *name resolution*. This is the process by which a directory maps a distinguished name (DN) provided by a client to an actual object in the directory. We use this process in the following circumstances:

- When locating the base object of an LDAP search or compare operation
- When locating the target entry for a modify, delete, rename, or bind operation
- When locating the parent of an entry to be added to the directory

A DN presented in this fashion is called a *purported name*. The client purports (assumes) that it exists, and it is up to the directory system to check whether this is true.

In the simplest case—a client contacts a directory server and presents the DN of an entry contained in that server—the name resolution process is simple: The server simply checks whether the entry exists and takes the appropriate action. For example, if a directory client attempts to read an entry via an LDAP base object search, the server checks whether the DN of the entry is within one of the directory partitions it holds; if it is, it checks its database for the entry. If the entry exists, any requested attributes are returned. Otherwise, a "no such object" error is returned to the client.

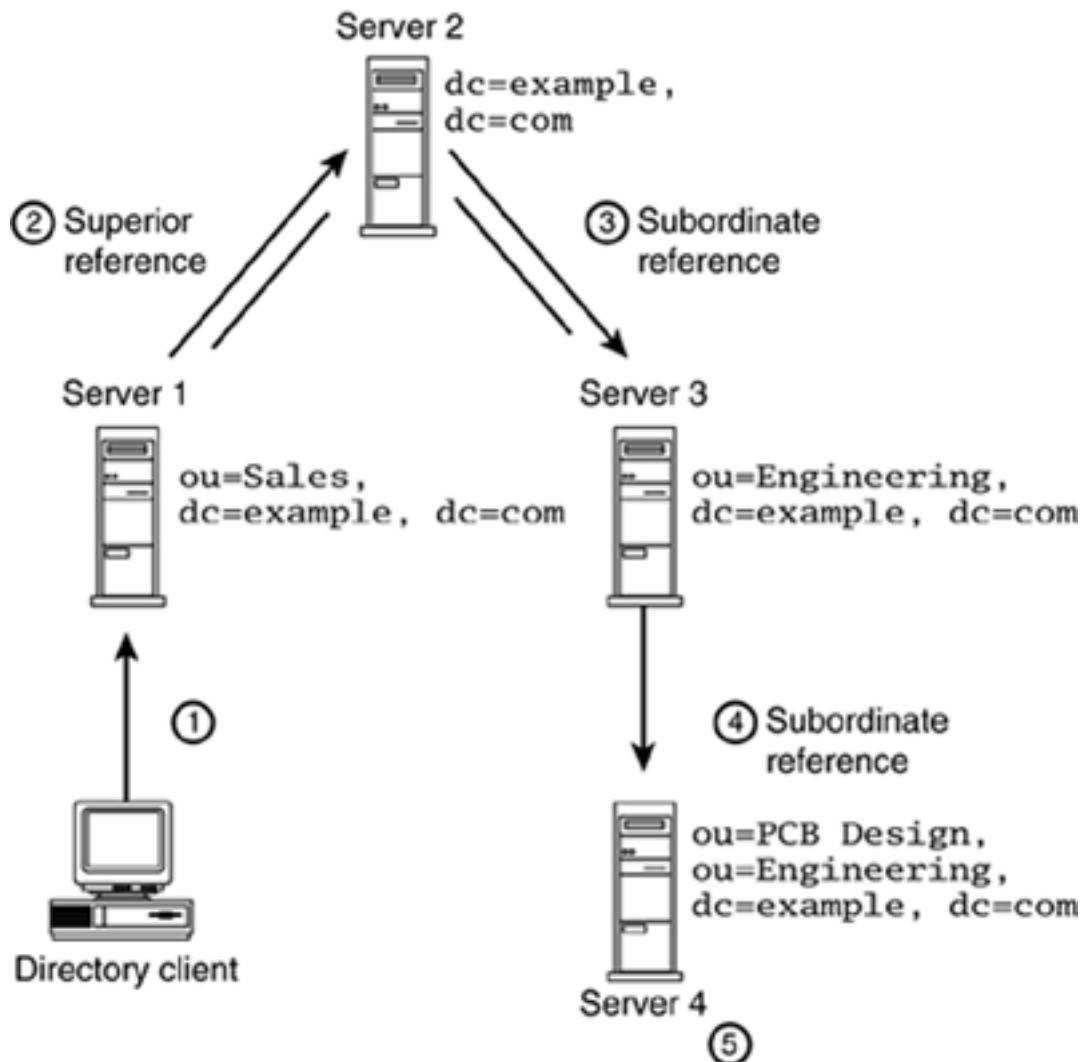
If the client presents a purported name whose DN is not within any of the directory partitions held by a server, the server must use any available knowledge references to resolve the name itself or aid the client in locating the server that can resolve the name. Conceptually, the server resolves the name by "walking up" or "walking down" the directory tree until the appropriate server is located. The knowledge references fully describe how a given directory partition fits into the DIT, so a server always knows whether the next step in the name resolution process involves walking up or walking down the DIT.

For example, suppose that the directory client depicted in [Figure 10.7](#) contacted Server 1 and presented the purported name `cn=Barbara Jensen,ou=PCB Design,ou=Engineering, dc=example,dc=com`. Name resolution would proceed as follows:

1. The directory client would present the purported name.
2. Because Server 1 does not hold a directory partition that could contain the entry, the immediate superior knowledge reference would allow the name resolution process to walk up the tree to Server 2.
3. Server 2 contains a subordinate reference for the directory partition `ou=Engineering, dc=example,dc=com`, so the name resolution process would walk down the tree to Server 3.
4. Server 3 holds a subordinate reference for the directory partition `ou=PCB Design, ou=Engineering,dc=example,dc=com`, so the process would walk down the tree again.

5. Finally, Server 4 would be consulted to check whether the entry actually exists.

Figure 10.7. Name Resolution in a Distributed Directory



Purported name: `cn=Barbara Jensen, ou=PCB Design, ou=Engineering, dc=example, dc=com`

Up to this point, we've been discussing knowledge references and name resolution in an abstract sense, and you may be curious about the software in the directory system that actually performs this tree-walking operation. LDAPv3 supports client-side processing of knowledge references, as well as server-side processing. In client-side processing, the knowledge reference information is provided to the client, which uses that information to contact additional servers to fulfill the request. In server-side processing, the server contacts other servers on the client's behalf and provides the complete set of results. In the next sections we discuss these two approaches in detail, examine the benefits and drawbacks of each approach, and offer advice on how to choose between them.

Handling Distribution in the Client: LDAP Referrals and Search Result Continuation References

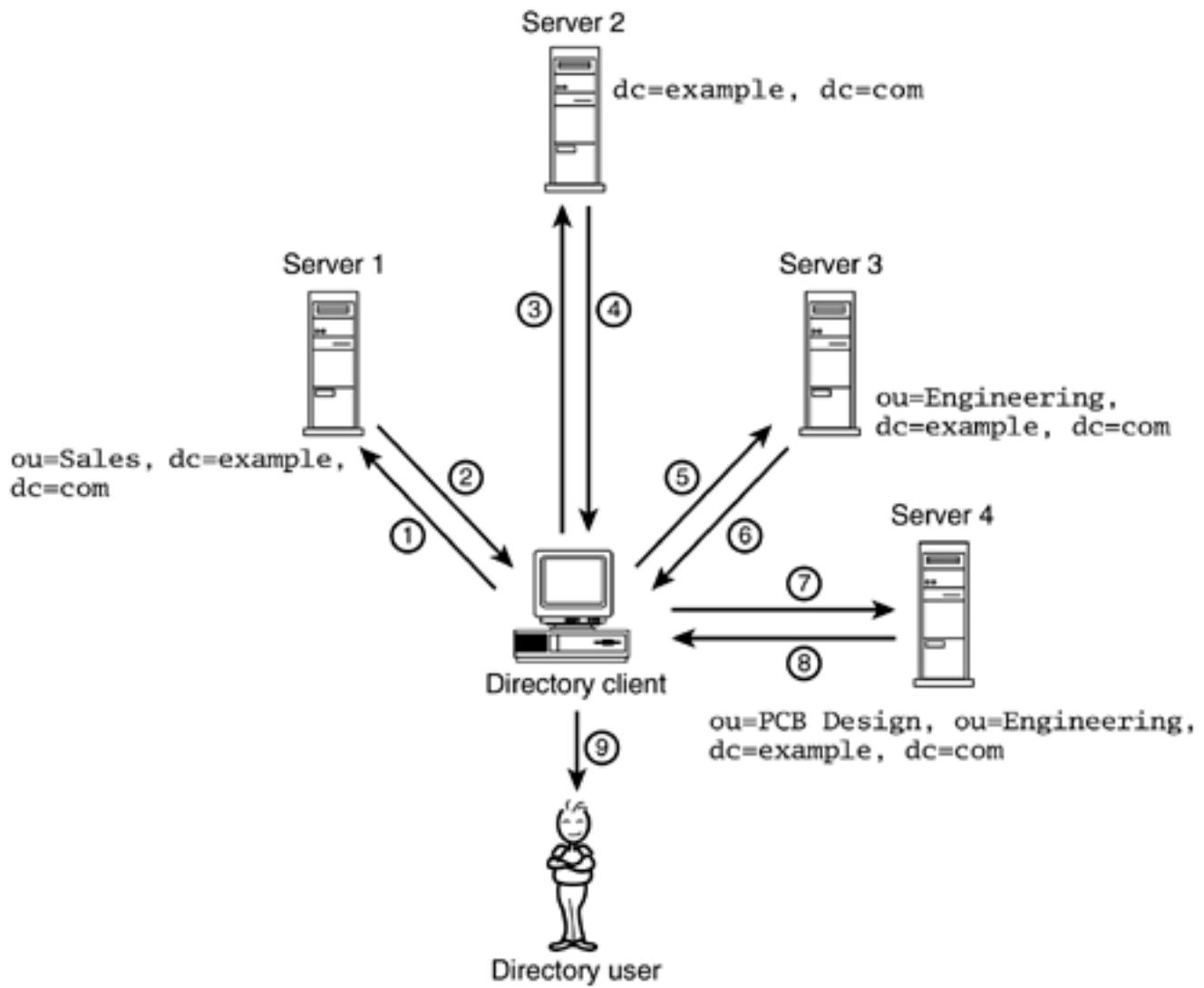
LDAP referrals and search result continuation references are pieces of knowledge reference information sent from an LDAP server to an LDAP client indicating that other servers need to be contacted to fulfill the client's request. Referrals and search result continuation references are defined as follows:

- **LDAP referrals.** When an LDAP client performs an LDAP add, modify, delete, modify DN, compare, or search operation with a base-level scope, and the server does not hold the target entry, the knowledge reference information is returned to the client in an LDAP referral. An LDAP referral is a special type of LDAP result message (LDAP errors are also carried in result messages) that directs the client to another server that may be able to service the operation.
- **LDAP search result continuation references.** When an LDAP client performs a one-level or subtree search and the server has knowledge of other subordinate directory partitions, knowledge references are returned as search result continuation references. A search result continuation reference is a search result (an entry is another type of search result) that carries information about the subordinate directory partitions and how to contact servers that hold those contexts.

LDAP referrals and search result continuation references are very similar. The only significant difference between them is how the information is packaged: Whereas LDAP referrals are contained in LDAP result messages, search result continuation references are contained in search result messages. Normally, the LDAP client libraries automatically handle processing of referrals and search result continuation references, and the subtle differences are hidden.

[Figure 10.8](#) illustrates with an example. Suppose that an LDAP client issues a search operation to Server 1 with a search base of `ou=Engineering,dc=example,dc=com`, a scope of `subtree`, and a filter of `(&(objectclass=person)(sn=smith))`. In other words, the client wants to find all persons within `ou=Engineering,dc=example, dc=com` whose surname is Smith.

Figure 10.8. Distributed Searching with Client-Side Processing of Knowledge References



To service this search request, the distributed directory first uses name resolution to find the base object of the search. It then uses any knowledge references to refer the client to all the servers it needs to contact to obtain the entire set of matching entries.

The following sequence describes this search procedure in detail:

Part 1: Resolving the name of the base object of the search.

Step 1. The client submits the search request to Server 1.

Step 2. Server 1 does not hold an appropriate directory partition, so it examines the superior reference and returns to the client a referral to Server 2.

Step 3. The client submits the search operation to Server 2.

Step 4. Server 2 contains a subordinate reference to Server 3 for an appropriate directory partition, so it returns to the client a search result continuation reference that points to Server 3.

Step 5. The client submits the search operation to Server 3.

Part 2: Servicing the search operation.

Step 6. Server 3 searches its database and returns any matching entries to the client. Because it also contains a subordinate reference

to Server 4, it returns to the client a search result continuation reference that points to Server 4.

Step 7. The client submits the search operation to Server 4.

Step 8. Server 4 returns any matching entries to the client.

Step 9. The client combines the lists of entries returned from Server 3 and Server 4, and presents them to the user.

This example shows how the directory, using knowledge references, can direct a client to the correct servers—even if the client happens to ask the wrong server. In fact, the whole point is that LDAP clients should not have to worry about which server contains which data within the directory service.

The Structure of LDAP Referrals and Search Result Continuation References

The information in LDAP referrals and search result continuation references is in the format of an LDAP Uniform Resource Locator (URL), as defined in RFC 2255. The following information is contained in a referral or reference:

- The host name of the server to contact.
- The port number of the server.
- The base DN (if processing a search operation) or target DN (if processing an add, delete, compare, or modify DN operation). This part is optional; if absent, the client should use the same base DN it used when performing the operation that resulted in the referral.

This information is all that is required for the client to chase the referral. For example, if a client contacts Server 2 and searches the subtree `dc=example,dc=com` for all entries with the surname Smith (as in step 3 of the preceding example), the referral would be returned as the following LDAP URL:

```
ldap://server3.example.com:389/ou=Engineering%2Cdc=example%2Cdc=com
```

This URL indicates that the client should contact the host `server3.example.com` on port 389 and submit a search rooted at `ou=Engineering,dc=example,dc=com` to collect the rest of the search results (each comma within the DN is represented by the hexadecimal escape sequence "%2C" as required by the URL specifications). Notice that the base DN in the referral is different from the one originally supplied by the client. When a server changes the search base before sending the referral or search result continuation reference, it is indicating to the client that, in addition to contacting a different server, it must search another part of the DIT to complete the search processing.

Note

Support for LDAP referrals and search result continuation references is standardized as part of the LDAPv3 specification. Servers and clients based on the older U-M LDAPv2 source code distribution support an experimental implementation that supports only referrals and is incompatible with the LDAPv3 standard. Netscape Directory Server 6 operates with both v2 and v3 clients by sending LDAPv3-format referrals to LDAPv3 clients and LDAPv2-format referrals to LDAPv2 clients.

Tip

If you are developing an application using the Netscape C LDAP SDK or the Netscape Java LDAP SDK, you can either instruct the SDK to follow, or "chase," referrals automatically (the default with the C SDK), or you can tell it to pass referral information back to your program for action.

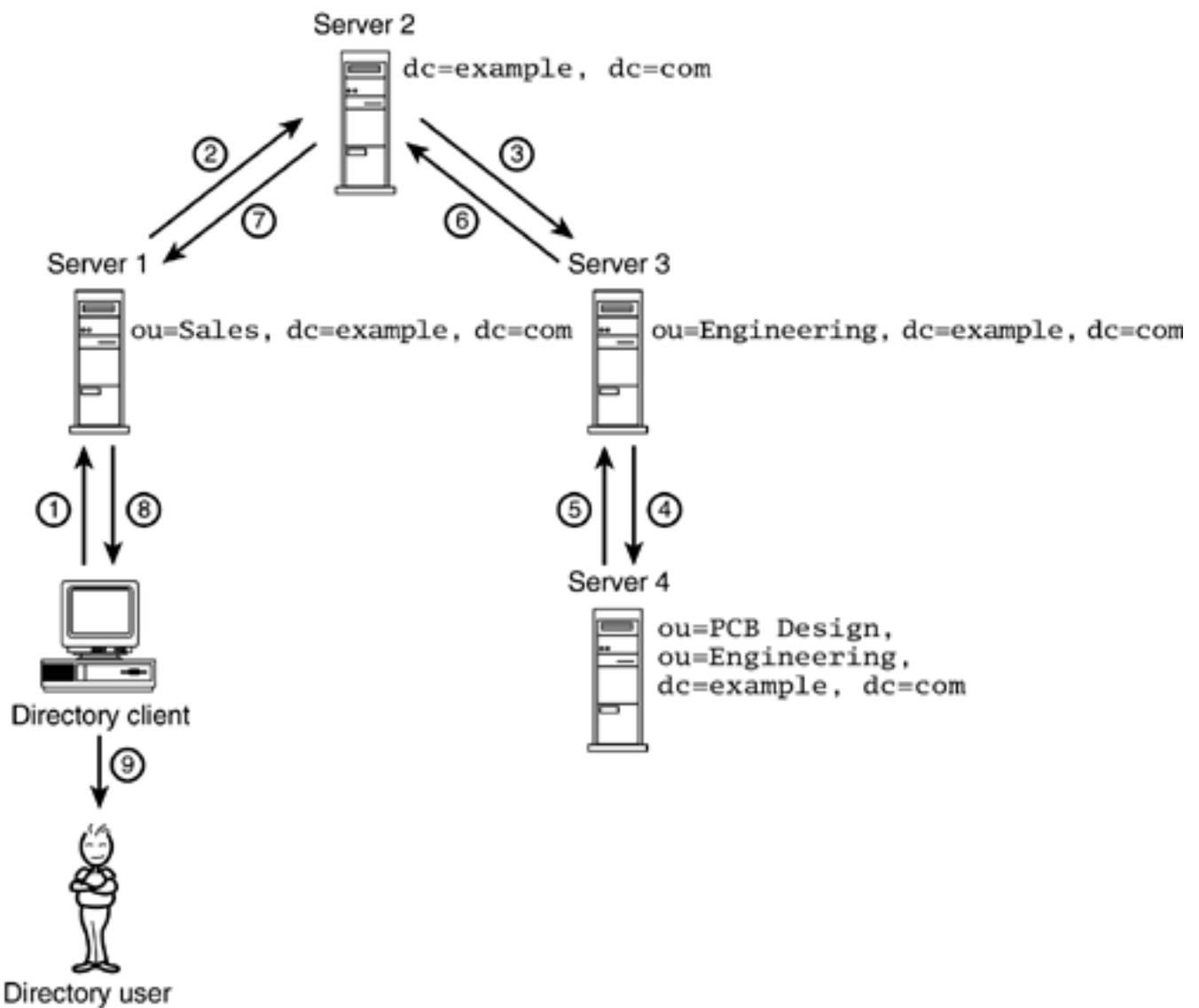
One reason you might not want to chase referrals automatically is if you wanted to provide feedback to the user indicating that a referral was being followed. Or you might want to warn the user if a referral outside your corporate firewall is returned.

As a developer, you have total control over how your application handles referrals. Good directory client software even allows the end user to configure whether referrals should be automatically followed and how to handle authentication when referrals are being followed.

Handling Distribution in the Server: Chaining

Another way to glue together directory partitions is with *chaining*. When this approach is used, the client starts by submitting an operation to a server in the usual way. If the server is unable to process the request completely because it does not hold all the required data, it contacts other servers on behalf of the client (forming a chain of connections between servers). After the other servers have completed the operation, the server that was contacted by the client returns the combined results to the client. [Figure 10.9](#) shows how a directory distributed by means of chaining would respond to the query shown in [Figure 10.8](#).

Figure 10.9. Distributed Searching with Chaining



When using chaining, servers perform the following steps to service a search operation, as illustrated in [Figure 10.9](#):

Part 1: Resolving the name of the base object of the search.

Step 1. The client submits the search request to Server 1.

Step 2. Server 1 does not hold an appropriate directory partition, so it consults its immediate superior reference and chains the operation to Server 2.

Step 3. Server 2 contains a subordinate reference to Server 3 for an appropriate directory partition, so it chains the operation to Server 3.

Step 4a. Server 3 verifies that the base object exists.

Part 2: Servicing the search operation.

Step 4b. Server 3 searches its database for any matching entries. It also discovers that it holds a subordinate reference to Server 4, so it chains the operation to Server 4.

Step 5. Server 4 searches its database and returns any matching entries to Server 3.

Step 6. Server 3 combines the entries retrieved from its own database with those returned by Server 4 and sends the resulting set to Server 2.

Step 7. Server 2 returns the set of entries to Server 1.

Step 8. Server 1 returns the set of entries to the client.

Step 9. The client presents the set of entries to the user.

As you can see, the same number of steps is involved in client-side and server-side processing of knowledge reference information. The major difference is which directory service component performs the processing necessary to retrieve the distributed information. In a chained system, the processing is done by the servers; in a system that uses referrals, the processing is done by the clients.

Deciding between Client-Side and Server-Side Processing of Knowledge Reference Information

Each method of linking distributed directory servers has advantages and disadvantages. Server-side processing generally reduces client complexity, but at the cost of increased server complexity. Servers that support chaining must be able to collect search results from remote servers and send the resulting set to directory clients. Client-side processing, on the other hand, requires more client complexity to handle the chasing of referrals and collation of search results. However, it offers more flexibility for client application writers and allows a developer to provide more feedback to an application's users about the progress of a distributed directory operation.

You may not have a choice about which method you use if your directory server supports only one method. Some directory server software, such as Microsoft Active Directory, supports only referrals, and the Microsoft LDAP client software automatically follows referrals generated by Active Directory. Other directory software, such as Novell eDirectory and Netscape Directory Server 6, allows you to glue your directory together using either referrals or chaining, or a combination of the two.

Servers based on X.500 standards usually support chaining and may also support referrals. The X.500 specification does not actually mandate that servers support chaining, so the type of distribution method (chaining or referrals) again depends on the software in use. Consult your server software documentation to determine which methods are supported and recommended.

Configuring a Distributed Directory

The procedure by which you configure a distributed directory depends on the type of directory server software you're using. The following are the four most common types of directory server software:

1. **Microsoft Active Directory.** Active Directory manages the relationships between domains using proprietary methods. Tools provided with Active Directory allow you to add and remove domains as needed.
2. **Novell eDirectory.** Novell eDirectory is modeled on the X.500 standards, but it uses proprietary methods to establish new partitions and set up the hierarchical relationships between them. Novell's Partition Manager utility allows a directory administrator to create new partitions and assign them to Novell eDirectory servers.

3. **X.500.** X.500 servers based on the 1993 standard define a protocol for setting up and tearing down relationships between servers. The actual protocol is known as the Hierarchical Operational Binding Protocol. As the name implies, it is used to establish a hierarchical relationship between two X.500 naming contexts and sets up the superior and subordinate references. Your software may not use this rather frightening terminology (in fact, one would hope not!); instead, it may provide a more user-friendly interface for specifying partitions and how they relate to one another. Consult your server's documentation for specific procedures.
4. **Netscape Directory Server 6.** Netscape Directory Server 6 requires that you manually configure the relationship of each server to other directory servers in your organization. You can configure distribution via client-side or server-side processing. With client-side processing, knowledge reference information is returned to the client in referrals and search result continuation references. With server-side processing, the server chains operations to other servers as needed to service the client's request, and then sends the combined result to the client. We'll discuss Netscape Directory Server 6 in more detail next.

Configuring Distribution with Netscape Directory Server 6

Before you configure your distributed directory, you must decide how to apportion your directory data across multiple servers. That topic is complex, and it is discussed later in this chapter, in the section titled Designing Your Directory Server Topology. For the purposes of this example, let's assume we have decided that our topology will be as shown in [Figure 10.3](#). Also assume that the host name of Server 1 is `server1.example.com` and the host name of Server 2 is `server2.example.com`. Server 1 holds the directory partition `dc=example,dc=com`. This partition contains all of the corporate data for `example.com`. Server 2 holds the directory partition `ou=External Customers, dc=example,dc=com`. This partition holds information about `example.com`'s customers. Note that this partition is subordinate to the previous partition.

We would like to be able to search across the entire directory without worrying about the fact that it's split across two servers. Therefore, we need to connect these two servers into a single directory topology. As mentioned previously, we can accomplish this with client-side or server-side processing of knowledge reference information.

Configuring Distribution with Client-Side Processing

Client-side processing of distribution information is configured in three steps. To configure client-side processing for the topology that was depicted in [Figure 10.3](#), for example, first you create a suffix for `ou=External Customers,dc=example,dc=com` on Server 1 (a suffix is another name for a directory partition). Second, you configure that suffix to generate referrals to the subordinate partition by providing the LDAP URL `ldap://server2.example.com:389`. Third, on the server holding the subordinate partition (Server 2), you set a default referral that directs clients to the superior partition's server by providing the LDAP URL `ldap://server1.example.com:389`.

Once the servers have been configured in this manner, clients will be referred to the appropriate server, no matter which server they submit their search operation to. Here are some examples of this behavior:

- If a client contacts Server 1 and performs a subtree search with a search base of `dc=example,dc=com` and a filter of `(sn=smith)`, Server 1 will return any matching entries in its database, along with a search result continuation reference to Server 2.

The client will then contact Server 2 and submit the same search operation, but with a search base of `ou=External Customers,dc=example,dc=com`, and Server 2 will return any matching entries.

- If a client contacts Server 2 and submits this LDAP search, Server 2 will realize that it does not hold the partition `dc=example,dc=com`. Therefore, it will return its default referral, which directs the client to contact Server 1. The client will submit the operation to Server 1, and the process will proceed as in the previous example.

Complete instructions for configuring client-side processing can be found in the section titled *Creating Suffix Referrals* in Chapter 3 of the *Netscape Directory Server 6 Administrator's Guide*.

Configuring Distribution with Server-Side Processing

To configure server-side processing of distribution information, you establish a relationship between two directory servers. The Netscape term for this relationship is a *database link*. A database link is configured on the server that holds the superior directory partition—in our example, Server 1. A database link contains the following configuration information:

- The host name and TCP port number of the server containing the subordinate partition
- Whether to use SSL when contacting that server
- The DN of the entry to bind as, and the password to use when binding

You can configure database links using Netscape Console.

After the database link is configured on the superior server, you must perform two more configuration steps on the server holding the subordinate partition (Server 2 in [Figure 10.3](#)):

Step 1. Create the entry that corresponds to the bind DN that the superior server will use when connecting, and set its password. Creating this entry allows Server 1 to authenticate to Server 2 when chaining an operation.

Step 2. Set an access control instruction (ACI) in the entry at the top of the subordinate partition. This ACI must grant proxy authorization rights to the entry you just created. The proxy authorization rights allow the server holding the superior partition to impersonate other users. In our example, Server 1 will authenticate to Server 2 using the configured bind DN. However, when Server 1 chains an LDAP operation to Server 2, it includes an LDAPv3 Proxied Authorization control that instructs Server 2 to perform the operation as if it had been performed by the user who originally submitted the operation to Server 1. For more information on the LDAPv3 Proxied Authorization control, see [Chapter 3](#), LDAPv3 Extensions.

At this point the two servers are configured so that Server 1 will chain search operations to Server 2. However, there is still another configuration step that must be performed. Server 2 must be configured so that it refers search operations outside of `ou=External Customers,dc=example,dc=com` to Server 1. Once this step has been performed, client operations will be processed as follows:

Step 1. If a client contacts Server 1 and performs a subtree search with a search base of `dc=example,dc=com` and a filter of (`sn=smith`), Server 1 will chain the operation to Server 2, collect the entries returned by Server 2, and combine those entries with any matching entries in its database. The complete result set will be returned to the client.

Step 2. If a client contacts Server 2 and submits this LDAP search, Server 2 will realize that it does not hold the partition `dc=example,dc=com`. Therefore, it will return its default referral, which directs the client to contact Server 1. The client will submit the operation to Server 1, and the process will proceed as in the previous example.

Complete instructions for configuring server-side topology processing can be found in the section titled Creating and Maintaining Database Links in Chapter 3 of the *Netscape Directory Server 6 Administrator's Guide*.

Authentication in a Distributed Directory

Handling authentication of directory clients in a distributed directory presents some challenges. The server or servers that ultimately handle a client request must verify the identity of the client so that they can enforce access control restrictions. This is true even if the server handling the request is not the server to which the client originally authenticated. For example, consider what happens when a directory client connects and authenticates to a server and then submits a search operation chained to another server.

The following steps describe how authentication is performed in chained environments such as that depicted in [Figure 10.10](#).

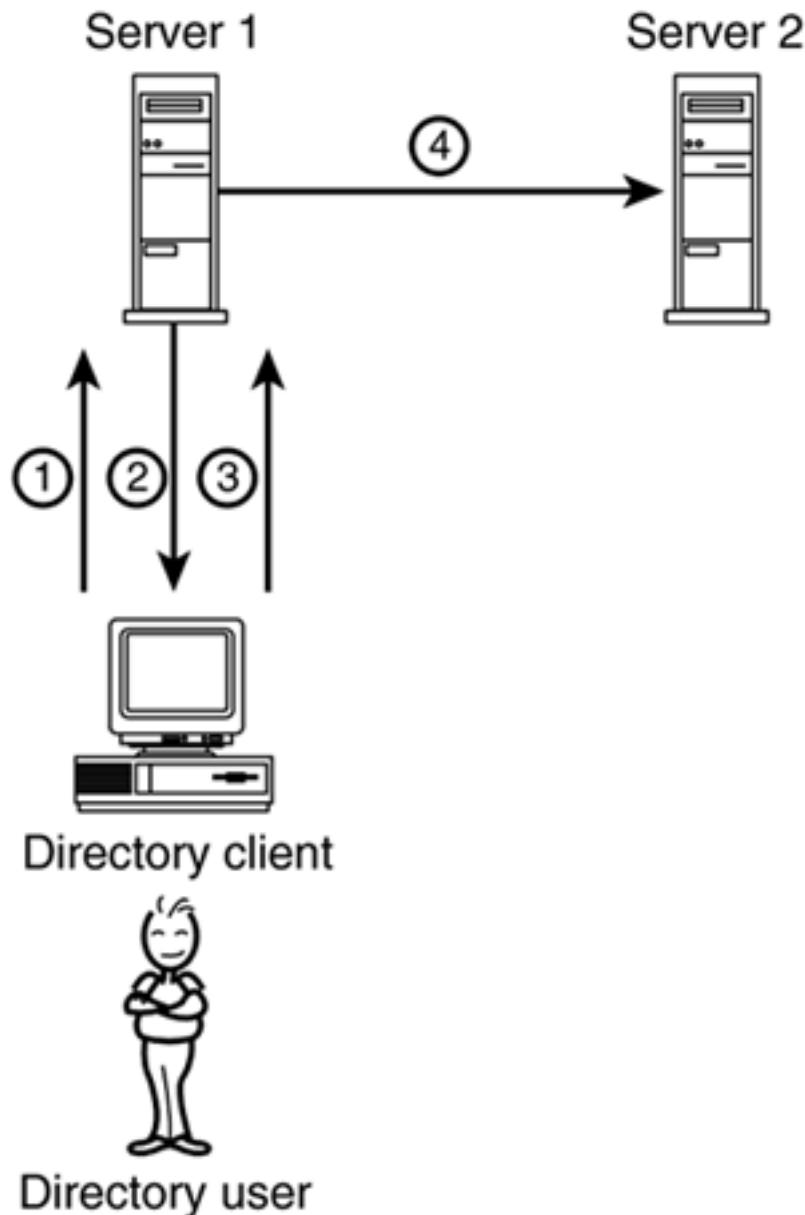
Step 1. The directory client connects to Server 1 and authenticates.

Step 2. If the client's authentication credentials are successfully verified, Server 1 returns a success code to the client.

Step 3. The client submits a search operation to Server 1.

Step 4. Server 1 determines that it does not hold the appropriate partition, so it chains the operation to Server 2.

Figure 10.10. Authentication in a Chained Topology



For Server 2 to enforce access control while performing the search operation, it needs to know the identity of the client. There are two ways it might learn this information:

1. Server 1 might tell Server 2 who the client is, and Server 2 might simply choose to believe Server 1. This approach requires that Server 2 trust Server 1 to verify authentication credentials correctly. This is the approach used by Netscape Directory Server 6 when it makes use of the LDAPv3 Proxied Authorization control.
2. Server 1 might pass to Server 2 the client's identity and authentication credentials. Server 2 could then independently verify the credentials. This approach requires that Server 1 trust Server 2 not to misuse the authentication credentials (for example, if the credentials consist of a plaintext password, Server 2 must not reveal it to a third party).

In a chained environment, the chaining server (Server 1) can pass along the identity of the client when it chains the operation. If the server it chains to (Server 2) trusts the other server (presumably because the chaining server authenticated to Server 2), the operation can proceed. This is, in fact, how X.500 servers handle authentication in a distributed environment, and it is how Netscape Directory Server 6 handles authentication when configured to chain operations.

On the other hand, if the client has authenticated to Server 1 using certificate-based client

authentication, Server 1 can just pass along the digitally signed request and allow Server 2 to verify the client's identity directly because the digital signature contains the identity of the client. This is how X.500 servers operate when certificate-based client authentication is used. This is also how Novell eDirectory servers operate (all eDirectory authentication is handled with digital signatures).

Microsoft Active Directory uses a variant of this technique. User credentials based on Kerberos v5 accompany directory operations submitted by clients. These credentials can be verified by any server in the domain, and they can be verified by servers outside the domain if explicit trust relationships are established between domains.

Note

For certificate-based client authentication to work in a distributed directory, all servers must trust a common set of certification authorities. If this is not the case, it's possible for one server to successfully authenticate a client while another server rejects the client's credentials during chaining or following referrals. Such discrepancies can lead to end-user confusion.

In a directory distributed via referrals, the client resubmits its request to each server it is referred to during the course of processing an operation. Therefore, each server it refers to authenticates the client directly. This means that no prearranged trust relationships need to be established between servers. If simple authentication is used, the server referred to will likely need to verify the client's password by contacting another server because it's unlikely to contain the actual entry corresponding to the client identity.

For example, if a client submits a bind operation to Netscape Directory Server 6, and the bind DN is not held in any of the server's directory partitions, the server attempts to locate and contact the server that does hold the entry. If it can locate that server, it verifies the credentials by binding to that server using the credentials supplied by the client.

Security Implications

Both methods of combining directory partitions—client-side and server-side knowledge reference processing—require some careful thought about security. When you construct a distributed directory using client-side processing, take precautions to ensure that you have absolute control over the referrals contained in your directory. In addition, allow referrals and search result continuation references only to directories you trust.

Such caution is important because some directory clients may choose to automatically resubmit any referred operations and authenticate to the server to which they were referred. This means that if a rogue referral were placed in a directory, the directory clients might be tricked into resubmitting their authentication credentials to the rogue server. If the authentication method in use does not preclude a replay attack (for example, if the authentication credentials are in the clear), the rogue server can record the credentials and use them to masquerade as the client.

When you construct a distributed directory using chaining, you must use caution when establishing any trust relationships between servers. You should trust only remote servers under your direct control or run by organizations you trust. Administrators of Active Directory also need to be concerned with the trustworthiness of a domain's administrators before establishing a trust relationship.

In both types of environments, if you are using certificate-based client authentication, you should trust only digital signatures signed by a trusted certification authority. This setup requires that you trust that certification authority to issue certificates only to persons who have adequately proven their identity. Usually this means that you trust only those certificates granted by your in-house certification authority. In an extranet environment, however, you might choose to relax this restriction and accept certificates from your trading partners' certification authorities, assuming that you trust them, or from an independent certification authority such as VeriSign.

As with all other complex distributed systems, designing a secure directory requires careful thought and planning. It's always useful to think like a "bad guy" and imagine ways you might compromise the security of the system you're designing. More information on directory security can be found in [Chapter 12](#), Privacy and Security Design.

Advantages and Disadvantages of Partitioning

For many smaller directory deployments, it is perfectly reasonable to keep all your directory data in one partition. This arrangement generally results in better search performance because a single server can provide the data requested by the client without requiring that other servers be contacted.

If you have only a few thousand entries in your directory, there is probably no benefit to partitioning, regardless of the type of server software you're using. Even larger partitions (sometimes as big as millions of entries) are supported by certain server software. Consult your server documentation to determine the largest number of entries that is reasonable to place in a single partition. If the number of entries stored in your directory significantly exceeds the number that can be practically stored in a single partition, split your directory into multiple partitions. Otherwise, try to limit your directory to one or just a few partitions.

Another reason to maintain a single large partition has to do with the types of directory-enabled applications you're using. Some applications tend to search the entire directory namespace repeatedly; these types of applications perform best if they have to search only one server to obtain the data they need.

For example, Sun ONE Messaging Server, when delivering e-mail, must look up the recipient's e-mail address in the directory to determine routing information. If you provide a centralized e-mail forwarding service that routes mail addressed to `employee@domain` to the correct server, it's likely that the entire directory namespace of your enterprise will have to be searched to perform this function. If your directory is split into many partitions, the overhead of searching each of them can have a significant negative impact on the performance of such a directory-enabled application. If your directory must support these types of applications, try to limit the number of directory partitions or point these applications to a server that aggregates all the partitions into a single location. We describe this configuration in more detail in the design example later in this chapter.

Microsoft Active Directory takes a different approach to solving the problem of searching across the entire organizational namespace. A special type of replica called the *global catalog (GC)* contains partial copies of frequently searched attributes from all directory entries in all domains. When an application such as Microsoft Exchange needs to search across the entire organizational namespace, it searches the GC rather than all the individual domains.

Another factor to consider is the amount of update traffic that your servers must handle. If your directory is contained in a single partition, that partition must handle all of the update traffic. Depending on your software, this requirement may cause performance to degrade unacceptably. By partitioning your directory, you reduce the number of entries contained within each directory partition and therefore reduce the amount of update traffic that any given server must handle.

You may also want to partition your directory if your physical network topology includes many slower WAN links, and partitioning the directory enables you to place a partition close to the applications and users that use the data in that partition. For example, if you're using a network operating system (NOS) directory like Active Directory or Novell's eDirectory, it often makes sense to construct a namespace that reflects the physical WAN structure of the company and assign partitions to servers in each location. When users log in to the NOS, their credentials are checked against the directory, and login preferences and scripts are retrieved. If the directory partition holding a user's entry is assigned to a server located close by, traffic does not need to span a WAN link, and therefore performance improves.

On the other hand, if your organization's network is well connected—perhaps if all links are T1 or faster—there isn't much to be gained from this type of approach. These are only general guidelines; examine your network layout and application needs carefully to choose the design that will work best for you. Your directory server software probably comes with a planning and deployment guide that will give you specific advice.

Finally, if you know that your directory will expand significantly in the future and exceed one of the limitations, it may be best to partition the directory sufficiently to accommodate the growth without having to redesign your partitioning scheme. Although it is certainly possible to split and combine partitions, this is generally a time-consuming task. Avoid it if you can.

Following is a summary of the factors to consider when you're deciding whether to partition your directory:

Factors favoring a directory with a larger number of partitions:

- The number of entries is too great for a single partition or server.
- Your directory-enabled applications tend to read and modify entries from only the local workgroup.
- The amount of update traffic to a single partition is too large.
- Partitioning allows data and update traffic to remain local and avoid spanning WAN links.
- Directory use will expand significantly in the future, beyond the point where a single partition is feasible.

Factors favoring a directory with a smaller number of partitions:

- All the directory entries fit easily within a single partition and server.
- Your directory-enabled applications tend to read and modify entries across the whole enterprise.
- Few updates are generated by clients.
- The entire organization is centrally located on a single high-speed network.
- Directory size will remain fairly static, or you know that it will not exceed the capabilities of your software.

We will discuss these partitioning issues in more detail when we look at a sample design later in this chapter.

Designing Your Directory Server Topology

The optimal directory server topology for your organization depends on these five factors:

1. The directory-enabled applications you use or plan to use, and the expectations of the users of those applications
2. Your directory server software and its capabilities
3. Your physical network topology
4. Your directory namespace
5. Political considerations in your organization

As you begin work on your directory topology, you may find that you want to revisit your namespace design. Recall that each partition needs a partition root, which is the DN of the entry at the top of the directory partition. This means that you may create a partition only at a branching point in your directory. In addition, if you've decided on a completely flat namespace design, but later you decide that you need to partition your directory, you'll need to go back and think about your namespace again.

If you are designing an Active Directory system, Microsoft provides many tools and guides that are helpful in leading you to an optimal design. These guides take into account the preceding factors, as well many factors specific to Active Directory, and help you determine the appropriate number of domains and the trust relationships that should exist between them. The sections that follow outline the steps in designing a directory server topology. For more information, see the Further Reading section at the end of this chapter.

Step 1: Inventory Your Directory-Enabled Applications

To decide how your directory should be partitioned, you need to understand where the directory network traffic will be coming from. To get this information, start by taking an inventory of your directory-enabled applications. You probably already have such an inventory if you followed the steps outlined in [Chapter 6](#), Defining Your Directory Needs. We'll augment that inventory somewhat, focusing on the specific aspects of the applications that will affect your partitioning design.

While you're taking this inventory of your directory-enabled applications, you need to understand the demands those applications make on the directory. For each application, you should be interested in understanding the scope of directory operations performed by that application. In other words, what part of your organization's DIT does the application need to work with? Does it need to manipulate only a particular entry? Does it need to work within only a particular organizational unit? Or does it need to search the entire directory?

Knowing something about the performance requirements and the scope of requests an application makes allows you to decide whether that application can tolerate the overhead associated with a partitioned directory. If an application needs to search consistently across the entire namespace, having to contact many servers may be detrimental to performance. In this case the application will perform better if it needs to contact only a single server. For some applications, performance requirements are stringent enough that you need to configure your directory so that only one server must be contacted to meet that application's needs. On the other hand, for applications with relaxed performance requirements, it may

be perfectly acceptable for the client application to experience delays introduced by a highly partitioned directory.

You can inventory your applications in any manner that makes you feel comfortable. [Table 10.1](#) provides a form as a starting point; feel free to modify it as you see fit.

If you're just in the planning stage, you may not be able to produce such a detailed inventory because you have not identified all your directory-enabled applications. Use the following list as a starting point to think about the types of applications you might obtain and install. Although this list is not exhaustive, it does describe some of the more common types of directory-enabled applications:

Table 10.1. Taking Inventory of Your Applications

Application	Scope	Performance Requirements
Mail transfer agent: Sun ONE Messaging Server	Entire directory	Intermittent delays are acceptable; however, overall throughput must be at least 10 messages per second.
Netscape Communicator address book	Entire directory	Searches based on <code>cn</code> or <code>uid</code> should require no more than one second.
Network login (e.g., Active Directory login)	User's organizational unit	Delay of a few seconds is acceptable.

- **Interactive authentication and login applications.** These types of applications typically verify the user's login ID and password via a two-step process. First the directory is searched to find the entry that corresponds to the login ID presented by the user. Then the application attempts to bind to the directory using the entry's DN and the password supplied by the user. If the bind operation succeeds, the user is granted access to the application. Such applications typically make light use of the directory, and a response time of a second or so is acceptable.
- **Authorization applications.** These applications perform some sort of authorization check for the purpose of granting or denying access to a particular resource. For example, the Netegrity SiteMinder access control and authentication system can use an LDAP directory as its store of user information when making access control decisions for a complex Web site. This type of application makes heavy use of the directory because a directory lookup may be required for each HTTP request made to the Web server. Although caching of the resulting user and group information can reduce the load on the directory server, these types of applications generally place a heavy load on the server, and users of these applications expect high performance. A response time significantly faster than one second is required.

Also important, however, is the scope of these operations, which depends on the way the application is configured. Novell eDirectory, for example, requires that the user

or administrator configure a default context, which specifies the subtree under which searches are performed and therefore limits the scope of the search operation that maps the user login ID to a directory entry. If users log in primarily from a single physical location, this works well because the server that can provide the requested information can be located on the same network as the client. On the other hand, if you have a large population of "nomadic" users, little benefit is gained from a highly partitioned directory.

- **Address book applications.** These applications, typically embedded inside e-mail applications, are usually used for looking up information about specific people or groups. The searches generated by these applications usually span the organization's entire namespace. Response times of a second or two are usually tolerable. The address books in Netscape Communicator, Microsoft Outlook, and QUALCOMM Eudora are all examples of this type of application.
- **Messaging applications.** These applications use the directory to route e-mail messages to their destinations. They can generate a heavy load on a directory server, especially when they're delivering mail to groups of users. The searches generated often span the entire organizational directory namespace. The performance of an individual search request is generally not important, but the overall throughput of the directory can be a limiting factor in the number of messages delivered. Examples of this type of application include Sun ONE Messaging Server and most other LDAP-aware e-mail servers.

After you inventory the directory applications that will use the directory and you understand their scope and the expected response times, you can begin to understand the performance implications of a highly partitioned directory versus an unpartitioned directory.

Tip

In our experience, directories increasingly are becoming the nerve center of IT organizations. In the past it was possible to lay out a directory in such a way that clients could get almost all the information they needed from a local server, which held information only about the local workgroup. But now that LDAP has provided a standard way for client applications to retrieve directory information, a much richer, more functional set of directory-enabled software is available. These applications leverage the enterprisewide knowledge contained in your directory and tend to search across your entire organizational directory space.

For example, an e-mail client might support sending signed, encrypted mail by using the directory to look up the public keys of mail recipients. This type of application has a wide scope; it typically needs to look up an e-mail address in your directory and retrieve the public key associated with the entry. You should not be surprised to learn that a majority of your directory-enabled applications require fast access to your entire directory tree. If you are in this situation, a highly partitioned directory will perform poorly. You should strive to reduce the number of partitions if possible.

Step 2: Understand Your Directory Server Software and Its Capabilities

Your directory server will no doubt have some practical limitations on the following:

- The number of entries that can be stored in a partition
- The number of partitions that can be held on a server

- The number of entries that can be held on a server
- The number of subordinate references that can be maintained at a given level of the DIT
- The number of replicas of a partition that can be maintained
- The number of indexed searches that can be serviced per second
- The number of concurrent client connections that can be supported

As you design your topology, you may find that some previous decisions you made are at odds with the limitations of your directory server software. For example, if you've decided on a flat namespace design for your 500,000-entry directory but need to implement it with server software that has a practical limitation of 100,000 entries per server, you need to revisit your namespace design to accommodate the server software.

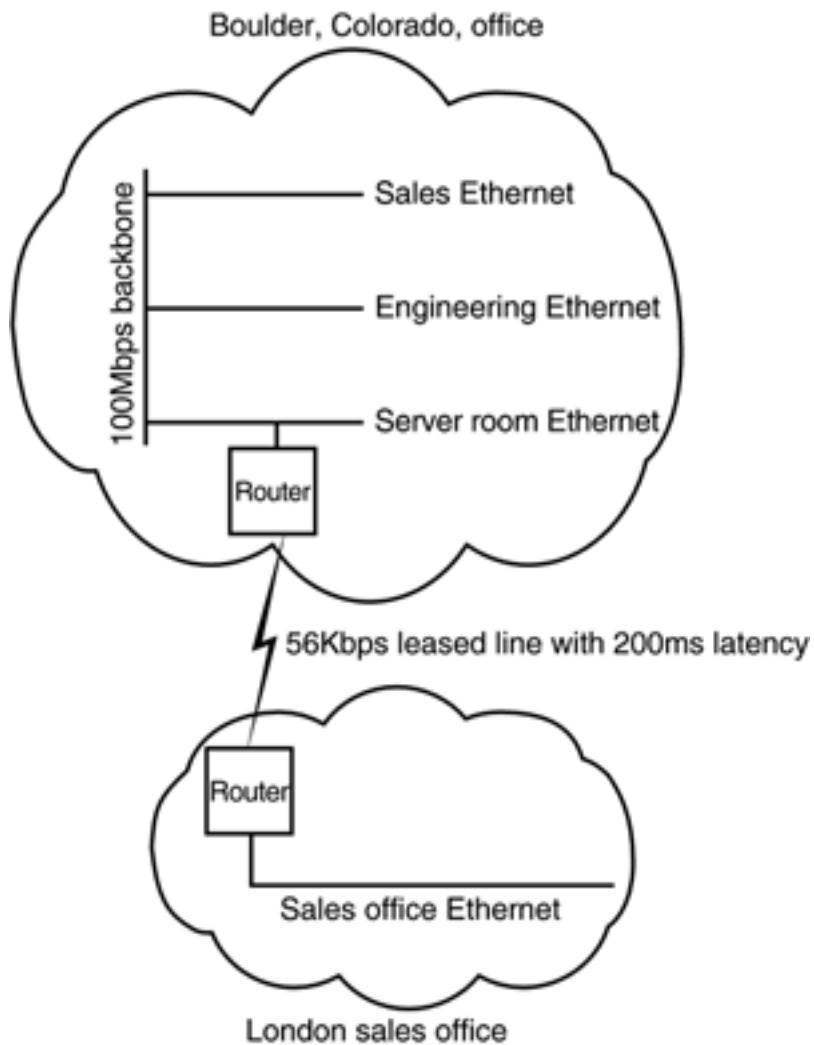
If you have not yet decided on a particular vendor's directory server software, you can use the preceding list as a checklist as you go through the decision-making process described in [Chapter 13, Evaluating Directory Products](#).

Step 3: Create a Map of Your Physical Network

Start with the map of your organization's physical network that we discussed in [Chapter 6, Defining Your Directory Needs](#). The components you should now focus on are your LANs and the connections between them. Label each LAN segment with its physical location, and then label the connections between LANs with the bandwidth, latency, and reliability of the connection. The bandwidth of a network connection is the maximum speed at which data can be transferred, and the latency of a connection is the time it takes for data to travel across the connection. A low-latency connection is ideal for LDAP traffic. High-latency links such as satellite links result in slow round-trip times for interactive traffic such as LDAP authentication, which can result in poor performance for applications. Such links, however, are often adequate for protocols such as File Transfer Protocol (FTP), which transfers data in bulk.

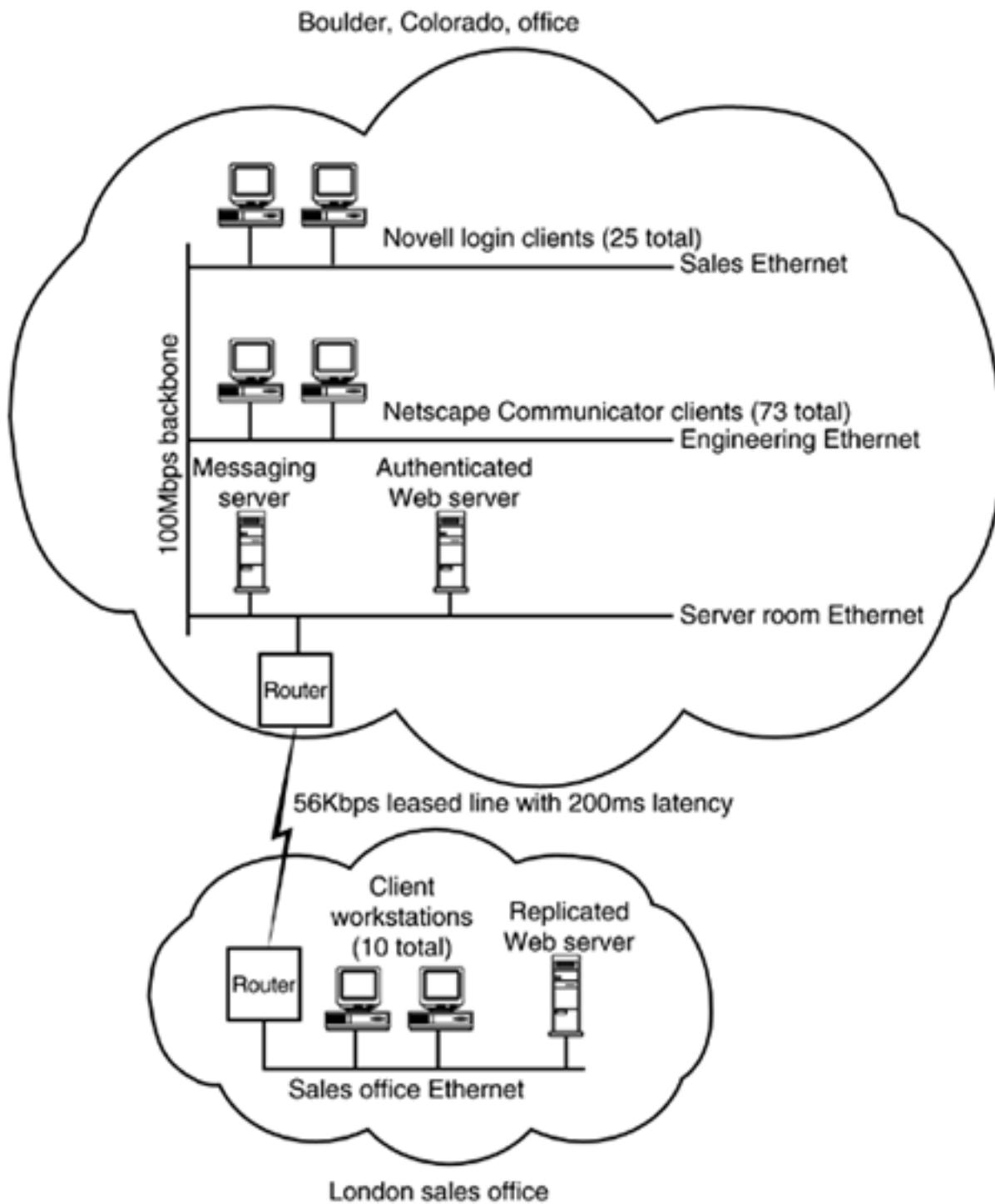
If you know that the connection is not 100 percent reliable or is active only during certain times of the day, note that on the map as well. Consider the hypothetical organization shown in [Figure 10.11](#). The main office, located in Boulder, Colorado, consists of several Ethernet segments connected via a high-speed backbone network. A branch office in London is connected via a 56Kbps leased line that is reliable but exhibits a relatively high latency of 200 milliseconds (one-fifth of a second).

Figure 10.11. A Basic Network Map



Next label on your map the locations of any directory-enabled applications you currently have. For example, if you have deployed (or plan to deploy) LDAP-enabled e-mail client software to users on each LAN, note this on your map. If you know the scope and performance requirements of the directory operations these clients execute, note those as well. As an example, note in [Figure 10.12](#) that there are two types of directory-enabled applications: Novell login clients and Netscape Communicator clients.

Figure 10.12. An Expanded Network Map Showing Locations of Directory-Enabled Applications



If possible, you should also include server-based directory-enabled applications such as e-mail servers, LDAP-enabled Web applications servers, and so on. You may not have installed these servers yet; that's fine, because you can experiment with their placement to optimize availability. Note on the map any other details, such as applications that perform many updates. You may need to enlist the help of other system and network administrators to accomplish this task.

After you put your directory clients and servers on your map, you can start to see traffic patterns emerge. Look for the locations of your high-volume directory clients, especially mail and authorization servers. Strive to limit traffic across slower WAN links; make sure that clients on remote networks have the information they need on the local LAN so that client requests need not traverse the WAN links.

Step 4: Review Your Directory Namespace Design

Now that you have a good idea of how your client and server applications are laid out, you can revisit your namespace design. Does it fit well with your WAN infrastructure and partition design? A properly designed namespace allows you to place partitions close to the users and applications that use the data they contain. If you find that your namespace makes this difficult, try other ways of arranging the namespace.

For example, if you are designing a directory for a multinational company and you've chosen a namespace that reflects the company's organizational structure, you may have problems defining partitions that minimize traffic across WAN links. In this case, consider an alternative namespace design. Rather than having the top-level branch points correspond to organizational structure, try making them correspond to geographical regions.

Step 5: Consider Political Constraints

Almost all organizations have one or two departments that operate autonomously from the main company. These departments have their own servers and their own IT staff, and they administer their own user population. Try to design your topology in such a way that allows these groups to operate autonomously while still participating in your directory system. For example, giving each of these departments its own directory partition and the ability to administer it is one approach that balances participation in the directory topology with the department's need to be autonomous.

Directory Partition Design Examples

This section describes two examples of directory partition design: one with a single partition, the other with multiple partitions. Your directory partition design will almost certainly not be exactly like one of the two scenarios described here. However, you may find some elements in common between your situation and these examples, and we hope you can draw useful conclusions from those common themes. You should also become familiar with the case studies in [Part VI](#) of this book. The case studies represent real-world directory deployments and do even more to illustrate the issues you will face as you set out to design the best directory topology for your environment.

A Single-Partition Directory Design Example

Background

Example Electronics is a small manufacturer of electronics modules for the airline industry. It employs 1,000 people, most of them at its large Boulder, Colorado, office. It has two branch offices: one in London and one in Paris. The branch offices are connected to the main campus by 56Kbps leased lines. The Boulder campus consists of 30 Ethernet segments connected via a 100Mbps backbone network.

Inventory of Directory-Enabled Applications

Example Electronics is deploying its directory to support two major applications initially. The first is a directory-enabled messaging application; the second is an intranet Web application in which the directory will be used to store access control information. In addition, each user's workstation will run an address book application that addresses and digitally signs internal e-mail. [Table 10.2](#) shows the inventory of planned directory-enabled applications.

Capabilities of Directory Software

Example Electronics has not yet chosen a particular directory server package. Part of the planning phase of the directory deployment will be the development of a set of requirements for the directory server.

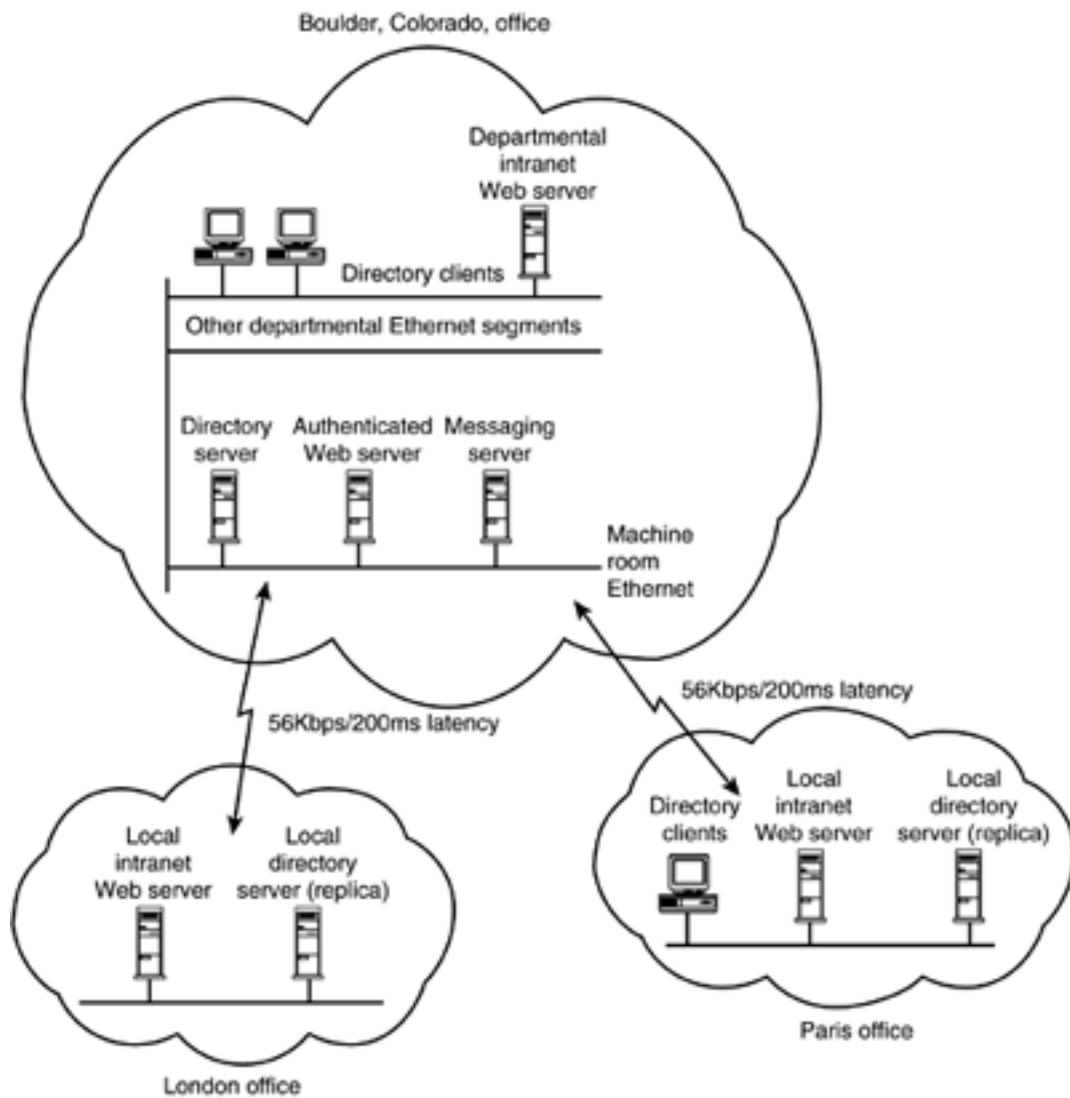
Table 10.2. Performance Requirements of Example Electronics' Planned Applications

Application	Scope	Performance Requirements
Messaging server	Entire directory	Intermittent delays are acceptable. However, overall throughput must be at least 10 messages per second. Each message delivery requires one search per recipient. Estimated peak directory load is 50 searches per second.
Intranet Web server	Entire directory	Approximately 100 authenticated page views per second are expected. Assuming that each page contains four inline images, this corresponds to approximately 500 hits per second. Web server caching of ACL information will reduce the load on the directory server to about 10 directory searches per second. Maximum acceptable delay time to send the entire Web page is one second.
Address book application	Entire directory	Total directory load will be approximately 5 searches per second. Searches should require no more than three seconds for indexed attributes.

Physical Network Topology

Whereas all users on the Boulder campus enjoy high-speed connectivity to other hosts on the Boulder campus, the two European offices are connected via slower leased lines (see [Figure 10.13](#)). Frequently these connections see heavy use from other applications. Actual throughput on the leased lines is likely much less than 56Kbps. Directory clients are present on each Ethernet segment and at both remote office locations. These remote clients require access to the entire directory tree to address and sign e-mail. In addition, multiple local intranet Web servers are planned so that each department or remote sales office can publish information to the intranet with access control. To guarantee fast response at the remote offices, it was decided that a replica of the entire directory needs to be available at each one.

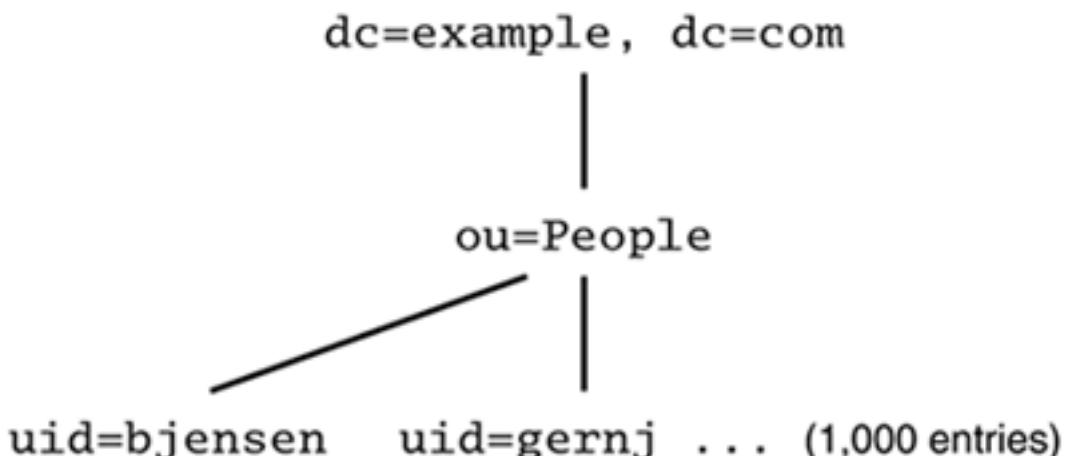
Figure 10.13. A Network Topology Map Showing the Location of Example Electronics' Directory-Enabled Applications



Namespace Design

Example Electronics is organized by projects, and individual employees move among projects fairly frequently. For this reason, the company has decided on a flat namespace (see [Figure 10.14](#)) with all employee directory entries contained directly below `ou=People`, `dc=example, dc=com`. Individual attributes within a person's entry denote the projects the person is working on, so it is not necessary to rename an entry if the employee changes projects.

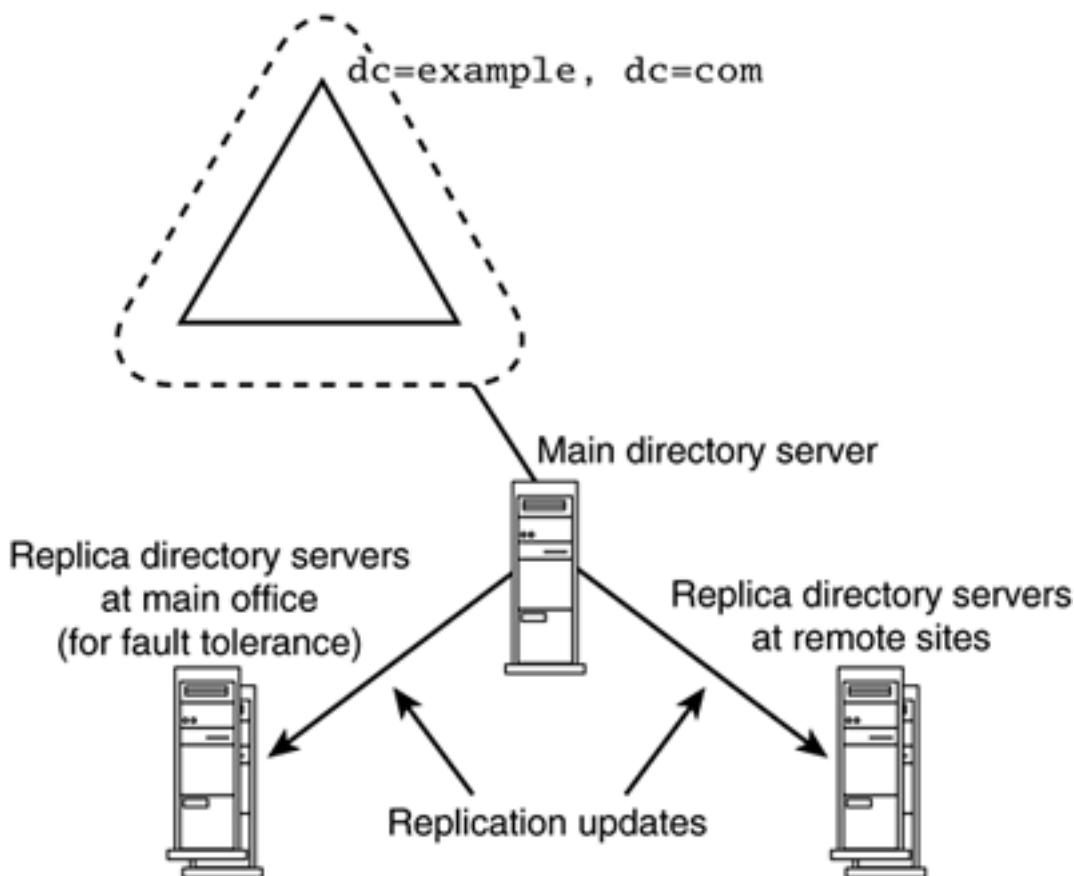
Figure 10.14. Example Electronics' Namespace Design



Conclusions

After reviewing all this information, Example's information systems organization has decided to retain the flat namespace and put all directory entries in a single partition. Two additional replicas of the directory will be placed at the main office for load balancing and fault tolerance. Replicated directory servers will be placed at both European offices, and clients at each of those offices will be configured to use the local replica, falling back on using the server at the Boulder campus only if the local server fails. [Figure 10.15](#) shows a diagram of the final design.

Figure 10.15. Example Electronics' Final Topology Design



A Multiple-Partition Directory Design Example

Background

Our hypothetical company in this example, Example Software, is a company with about 10,000 employees in three separate offices: New York, the corporate headquarters; San Francisco, which houses the software developers and support staff; and a London office, which houses European sales and support staff. [Table 10.3](#) shows the company's organizational chart.

The company has chosen to deploy Novell eDirectory. It also plans to deploy a third-party e-mail hub that leverages information stored in the Novell directory to route inbound Internet mail to internal destinations, including several shared mailboxes used by the support staff to handle support requests that come via e-mail.

Table 10.3. Example Software's Organizational Chart

New York Office	London Office	San Francisco Office
Finance department	European Sales	Software Development
Legal department	European Support	Western Region Support
Eastern Region Support	Human Resources	Human Resources
Human Resources		US Sales

Inventory of Directory-Enabled Applications

As previously mentioned, Example Software is planning to deploy two applications that use Novell eDirectory: NetWare login and e-mail delivery. Whereas the scope of the NetWare login application is generally limited to the local workgroup, the mail hub software needs to search across the entire organizational namespace and has a high throughput requirement (see [Table 10.4](#)).

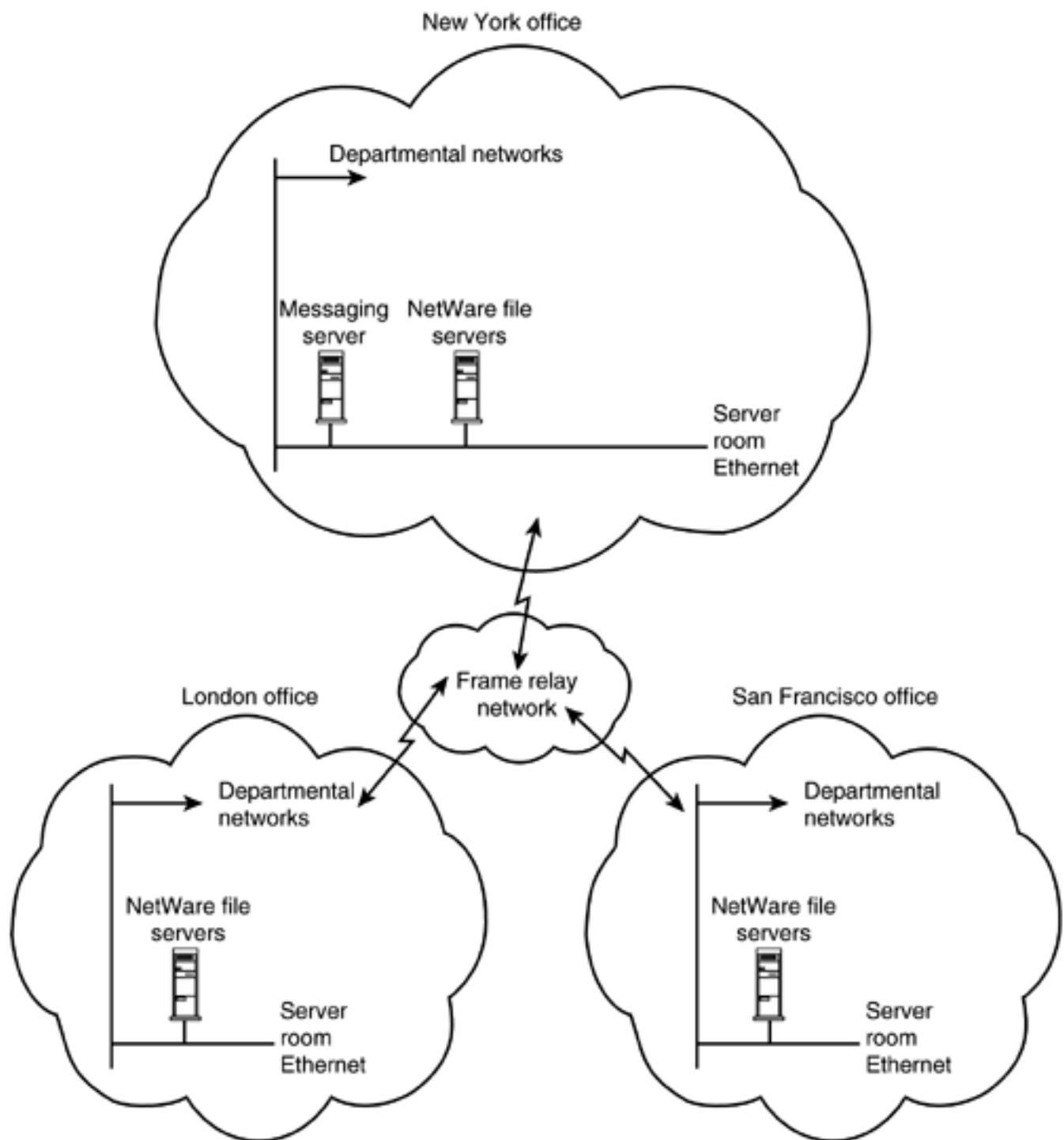
Capabilities of Directory Software

Novell's eDirectory is capable of handling all 10,000 entries in a single partition. In this case the capabilities of the directory software do not constrain our decision-making process.

Physical Network Topology

The three offices of Example Software are connected via a frame relay network. The network is reliable, but the speed (1.544Mbps at peak) is modest compared to the intracampus backbone networks. Furthermore, the intercampus links have only a 64Kbps committed information rate, which means that the measured throughput is often much less than the theoretical maximum. For this reason, it is a good idea to limit traffic across the WAN links as much as possible. [Figure 10.16](#) shows the intercampus links.

Figure 10.16. Example Software's Network Map



We can limit the traffic across the WAN links by placing the data needed by clients at each campus on an on-campus server.

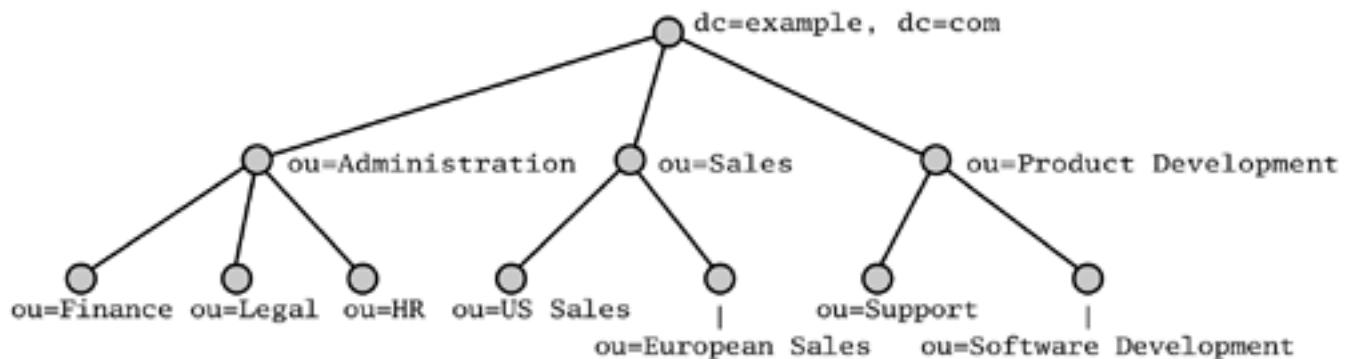
Table 10.4. Performance Requirements of Example Software's Planned Applications

Application	Scope	Performance Requirements
NetWare login	Local to workgroup	Retrieval time of a few seconds for user login scripts is acceptable.

Namespace Design

Example Software originally decided on a namespace that reflected its organizational structure. [Figure 10.17](#) shows this initial namespace layout.

Figure 10.17. Example Software's Original Namespace Design

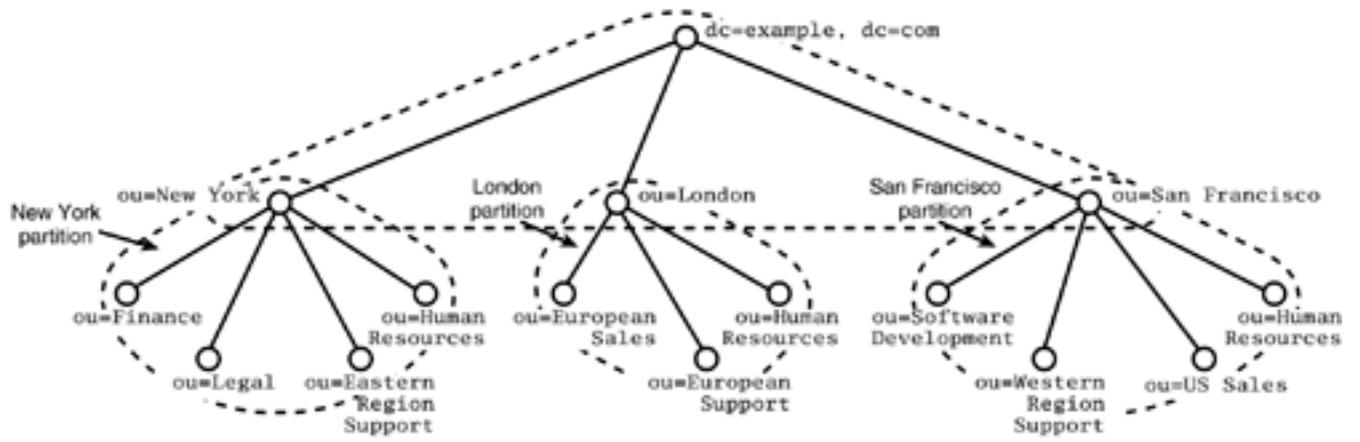


Recall that directory partitions can be defined only at branch points in the directory. This means that any possible partition must have a partition root corresponding to one of the organizational units in the design. Thus the partitioning choices are as follows:

- A single partition rooted at `dc=example,dc=com`. This is a reasonable choice, except it will require all directory traffic to cross the relatively low-speed WAN links.
- Three partitions, rooted at `ou=Administration,dc=example,dc=com`; `ou=Sales,dc=example,dc=com`; and `ou=Product Development,dc=example,dc=com`. Again, this arrangement makes it difficult to avoid sending directory traffic across WAN links. Note in [Table 10.3](#) that the Human Resources staff is divided among the three campuses. No matter where the `ou=Administration` partition is located, Human Resources staffs at the other two locations would need to traverse WAN links to retrieve their directory data.
- Eleven partitions, corresponding to each of the individual workgroups. Although this arrangement would allow partitions to be placed locally to each workgroup, management of all these partitions would be more time-consuming, especially if replication were used to provide additional copies of the partitions.

When you find yourself facing this kind of dilemma, it's often beneficial to step back and reexamine your choice of namespace layout. In this case, doing away with the existing namespace based on organizational structure and replacing it with one based on geographical layout makes the topology fit much better with the WAN infrastructure. [Figure 10.18](#) shows a revised namespace design.

Figure 10.18. Example Software's Revised Namespace Design



Notice that the three organizational units based on geography fit well with the WAN infrastructure. If we make these organizational units the roots of three partitions and locate those partitions at their respective campuses, we succeed in localizing user traffic to the individual campuses. Note that there are actually four partitions, as denoted by the dashed lines in [Figure 10.18](#). There must be a partition rooted at `dc=example, dc=com` to tie the three subordinate partitions together and allow name resolution. This partition, which is very small, can be replicated to each of the three locations via Novell replication.

One problem remains, however. The mail hub application needs to search across the entire organizational namespace frequently—perhaps several times per second. If we don't take special action, the mail hub will need to contact all of the directory servers, resulting in a great deal of traffic on the WAN links.

We can solve this problem by creating a dedicated server to hold replicas of all the partitions (replication will be covered in more detail in [Chapter 11](#), Replication Design). Because the mail hub has high-performance requirements, it's probably a good idea to provide it a dedicated server anyway. We call this an *aggregating server* because it aggregates the directory data into a central location for the mail hub's use. The one drawback to using an aggregating server is that placing all the entries on a single host may tax the system hosting the aggregating server.

Another more expensive but higher-performance alternative would be to place each replica on a separate server and locate all the replicas on the same network as the mail hub application. Performance would improve because each mail hub would have a dedicated directory server. If that approach still did not provide adequate performance, another alternative would be to make a copy of the directory data in a single Netscape Directory Server 6 server that would service the mail hub. The Netscape server, designed specifically with search performance in mind, is a better choice for this particular application than Novell eDirectory, which is designed primarily to support a highly distributed NOS environment.

Conclusions

[Figure 10.19](#) shows the final allocation of partitions to servers. The three campus directory servers each hold a copy of the top-level partition, rooted at `dc=example, dc=com`, and each campus directory server holds a partition comprising the data that local clients need. In addition, the aggregating server, which resides on the same network segment as the mail hub, holds read-only copies of all directory partitions and receives replication updates from the read/write partitions located at each campus site.

Figure 10.19. Allocation of Partitions to Servers at Example Software



dc=example, dc=com (read-only)
ou=New York (read/write)

New York server



dc=example, dc=com (read-only)
ou=London (read/write)

London server



dc=example, dc=com (read-only)
ou=San Francisco (read/write)

San Francisco server



dc=example, dc=com (read-only)
ou=New York (read-only)
ou=London (read-only)
ou=San Francisco (read-only)

Aggregating server
(on same Ethernet as mail hub)

This example shows that it's beneficial to spend some time thinking about namespace design and topology as a unit.

Topology Design Checklist

To review, here are the steps involved in designing your directory topology:

- Take a survey of your directory-enabled applications. Understand the scope of the directory data with which they need to interact. Understand the performance requirements.
- If you have already decided on a particular vendor's directory server software, understand its practical limitations, especially the number of entries that can be stored in a partition and how easily partitions can be managed. If you are still in the decision-making process and selecting server software, consider these factors when making your purchase decision.
- Review your physical network topology. Are there links that might be saturated by excessive directory traffic—for example, a WAN link? If so, strive to limit the amount of traffic that needs to span those links.
- Review your directory namespace requirements and design. Does your namespace fit naturally with a partitioning arrangement that minimizes WAN traffic? Is another namespace arrangement a better fit?

Further Reading

Active Directory Branch Office Guide Series: Planning Guide. Microsoft, Inc., 2001. Available on the World Wide Web at <http://www.microsoft.com/technet>.

Named Subordinate References in Lightweight Directory Access Protocol (LDAP) Directories (RFC 3296). K. Zeilenga, 2002. Available on the World Wide Web at <http://www.ietf.org/rfc/rfc3296.txt>.

NDS eDirectory 8.7 Administration Guide. Novell, Inc., 2000. Available on the World Wide Web at <http://www.novell.com/documentation/lg/edir87/index.html>.

Netscape Directory Server 6 Deployment Guide. Available on the World Wide Web at <http://enterprise.netscape.com/docs/directory>.

Novell's Guide to Netware 5 Networks. J. Hughes and B. Thomas, Novell Press, 1999.

Understanding X.500: The Directory. D. Chadwick, International Thomson Computer Press, 1996. Now out of print; selected portions available on the World Wide Web at <http://www.salford.ac.uk/its024/Version.Web/Contents.htm>.

Looking Ahead

This chapter has examined how the distributed directory works and what issues to consider when you're deciding how to partition your directory among servers. [Chapter 11](#), Replication Design, will examine how to improve the reliability and performance of your directory through the use of replication.

Chapter 11. Replication Design

- Why Replicate?
- Replication Concepts
- Advanced Replication Features
- Designing Your Directory Replication System
- Replication Design Checklist
- Further Reading
- Looking Ahead

Directories play a pivotal role in your organization's computing infrastructure. The services they provide are so crucial that downtime of your directory due to equipment failure probably cannot be tolerated. If your directory data is available from only a single server and that server fails or becomes unreachable, your users will have trouble getting their work done.

Directory replication protects you from this unfortunate situation by making your directory data available on multiple servers. It also improves performance by allowing you to make more copies of your directory data available and place them close to the users and applications that use them.

In this chapter, first we discuss some important replication concepts. Then we examine some related issues, such as replication scheduling and how schema and access control interact with replication. Finally, we explain how you should design a replication solution for your directory environment.

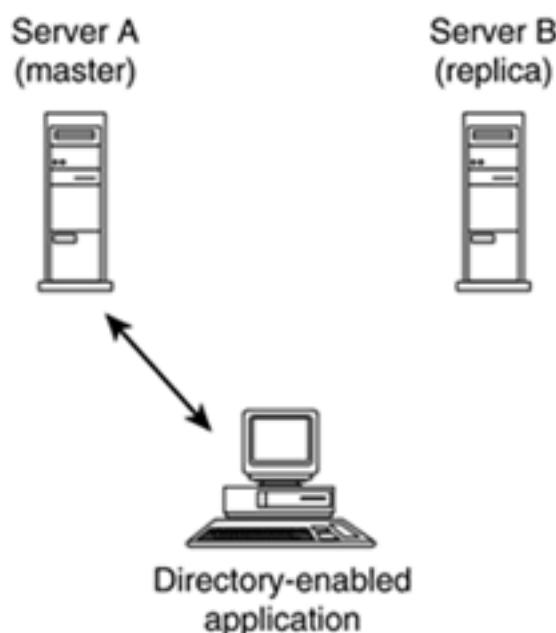
Replication, topology, and namespace design are all closely related. If you have not read [Chapter 9](#), Namespace Design, and [Chapter 10](#), Topology Design, we suggest you do so before attempting to design your replication. Having a good understanding of all these topics will enhance your ability to build a reliable, scalable directory.

Why Replicate?

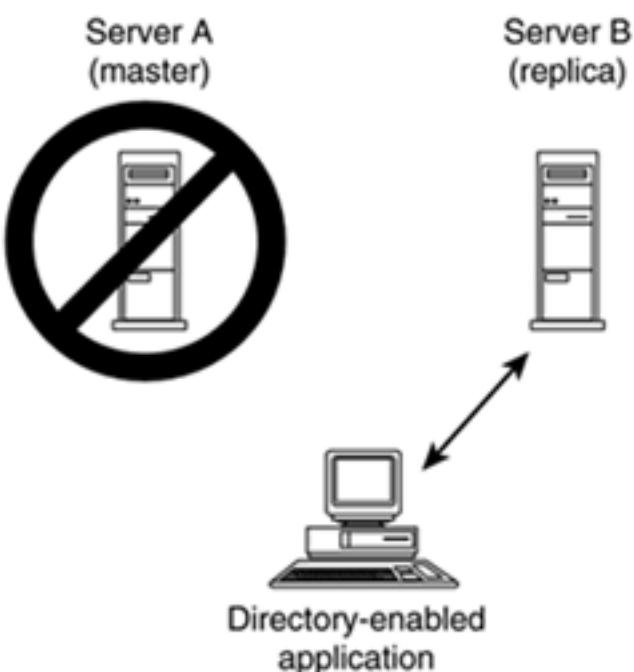
When you replicate directory content, you increase the reliability and performance of your directory service. Let's examine these two benefits and how they are achieved.

By making the directory data available in more than one location, you improve the reliability of your directory service. If a single server fails, your directory clients and directory-enabled application programs can contact a different server for their directory service. [Figure 11.1](#) illustrates how a client can use this redundancy to obtain service from a replica in the event of a server failure.

Figure 11.1. Increasing Reliability through Redundancy



If Server A fails...

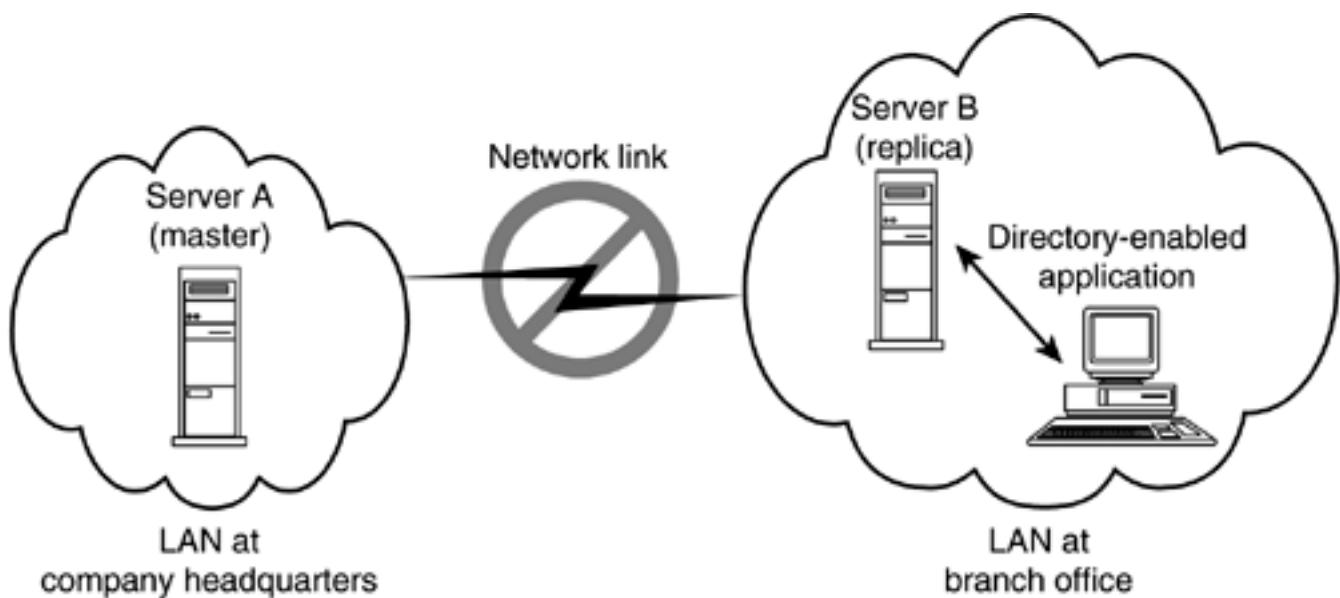


... the directory-enabled application can obtain its directory service from Server B.

... the directory-enabled application can obtain its directory service from Server B.

When you replicate directory content, you also make your directory more resistant to outages due to network failures. In [Figure 11.2](#), the directory-enabled application running on the LAN at the branch office can continue to function even if the network link between the branch office and company headquarters is down. Of course, the directory content on the replicated server may not be entirely up-to-date, but for the vast majority of applications slightly stale data is entirely acceptable. When the network connection between the offices is restored, Server B will once again be synchronized with Server A, eliminating any temporary differences.

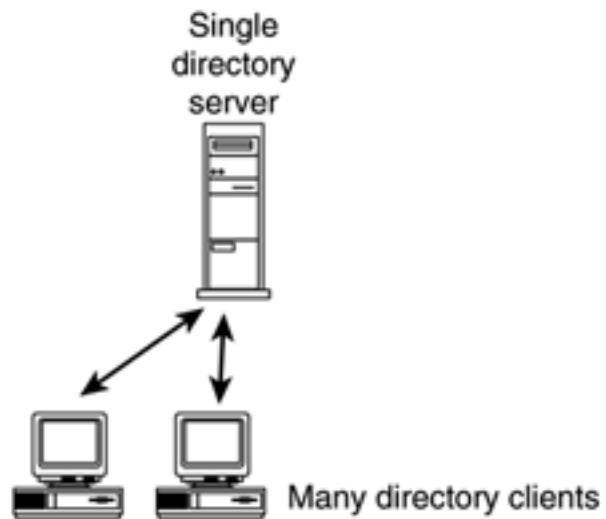
Figure 11.2. Replicating to Minimize the Impact of a Network Failure



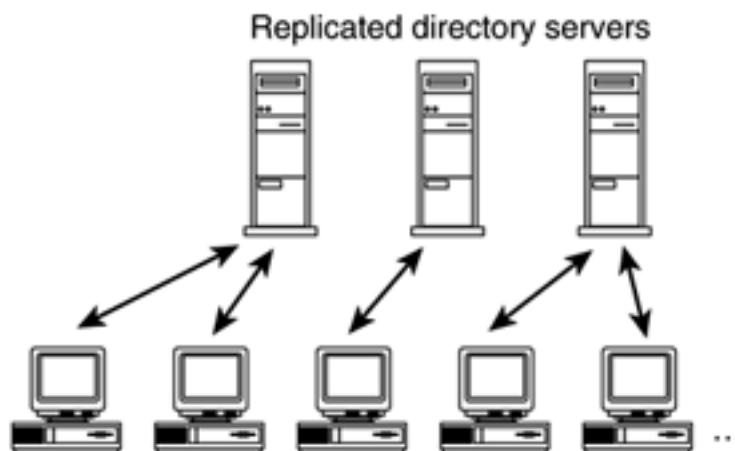
The situation in [Figure 11.2](#) also exemplifies the benefit that replicas can be located close to the users and applications that need them. This arrangement improves performance because there are fewer network nodes between the LDAP clients and the directory data.

Thus, when you replicate directory content, you can also improve the performance of your directory. Your directory can be made to handle more directory client requests by distributing the load across more servers. [Figure 11.3](#) illustrates the distribution of client load across multiple replicas.

Figure 11.3. Distributing Directory Client Load



A single directory server may not be able to handle all the directory client applications in your environment.



With replication, the client load can be distributed across as many servers as are required. Additional servers can be installed as the need arises.

Replication Concepts

Before we dive into designing our replication system, we should spend some time understanding basic directory replication concepts. These concepts are as follows:

- Suppliers, consumers, and replication agreements
- The unit of replication
- Consistency and convergence
- Incremental and total updates
- Initial population of a replica
- Replication strategies
 - Single-master replication
 - Multimaster replication
- Replication protocols

Each issue is discussed in the following sections.

Suppliers, Consumers, and Replication Agreements

In replication systems, we use the terms *supplier* and *consumer* to identify the source and destination of replication updates, respectively. A supplier server sends updates to another server; a consumer server accepts those changes. These roles are not mutually exclusive: A server that is a consumer may also be a supplier.

The configuration information that tells a supplier server about a consumer server (and vice versa) is termed a *replication agreement*. This configuration information typically includes the unit of replication (discussed next), the host name and port of the remote server, and other information about the replication to be performed, such as scheduling information. In other words, the replication agreement describes which consumer should receive the updates, what part of the directory is to be replicated, how the supplier connects to the consumer, and how the supplier authenticates to the consumer. Most directory server software stores replication agreements as entries in the directory, which means that you can often use utility software like the `ldapsearch` command-line utility to examine replication agreements.

The Unit of Replication

When we talk about replication, we need some common language to describe what is to be replicated. With most directory server software, the unit of directory partitioning is also the unit of replication. When you decide on a partitioning scheme, you are also deciding on your units of replication.

For example, with Microsoft Active Directory, the domain is the unit of replication. You cannot replicate part of a domain; all entries contained in the domain are replicated to all Active Directory replicas. Similarly, the Netscape Directory Server 6 unit of replication is the directory database. All entries contained in the database are replicated to all replicas of that database. If you need to replicate a particular subtree, that subtree must be contained in its own domain (Active Directory) or directory database (Netscape Directory Server 6).

Some directory server software allows you to create replicas that are subsets. We'll discuss

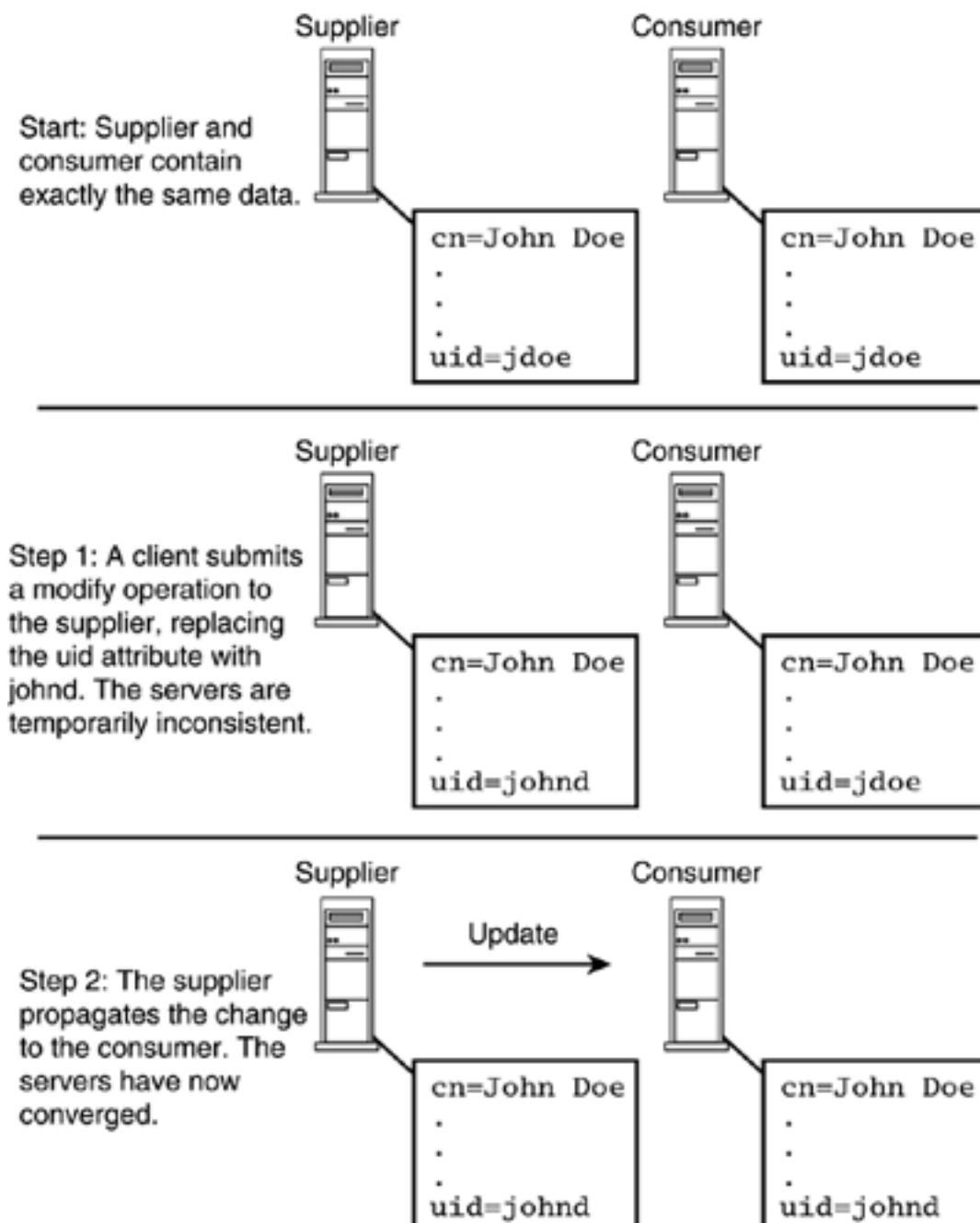
these capabilities later in this chapter, in the section titled Advanced Replication Features.

Consistency and Convergence

Degree of consistency describes how closely the contents of replicated servers match each other at a given point in time. A *strongly consistent* replica is one that provides the same information as its supplier at all times; that is, the effect of an update is not visible to any client until all of the replicated servers have received and acknowledged the update. From a client's point of view, the update occurs simultaneously on all the replicated servers.

On the other hand, a *weakly consistent* replica is permitted to diverge from its supplier for some period. For example, [Figure 11.4](#) shows that there is a period of time after a supplier has been updated but before the update has been propagated to a replica; during that time the replica contains stale data with respect to the current data on the supplier. We say that a supplier and a replica have *converged* when they contain the same data. It is important that replication systems eventually converge over time so that all clients see a consistent view of the directory.

Figure 11.4. Weakly Consistent Replicas



In a directory system that uses weakly consistent replication, directory clients should not expect their updates to be reflected immediately at each replica. For example, a directory application should not expect that it can update an entry and then immediately be able to read it to obtain the updated values. If the client is connected to a read-only replica, its update will have been redirected to an updatable replica and may take some time to be replicated to the server the client is connected to.

It may come as a surprise that all practical directory systems use weakly consistent replicas. Why? The answer has to do with performance. Imagine that a single supplier feeds three replicas, and that each of the replicas handles a large client load of search requests. If the supplier maintains strong consistency with its replicas, it must send a change to each one and receive a positive acknowledgment before returning a result to the client that sent the change. Because each replica is heavily loaded, it may be slow in sending the result to the supplier. The supplier can therefore return a result to the client no faster than the slowest replica acknowledges the update. This limitation can reduce performance unacceptably.

In addition, implementing strong consistency among replicas requires that replicas support a two-phase commit protocol. Such support is necessary so that the supplier server can back out (undo) an update if any of the consumers fails to acknowledge the change. The supplier would then return an error code to the client, and the client would presumably retry the operation later. This means that if any consumer server is offline or unreachable, the supplier server cannot accept any changes.

In addition to its lower performance, strong consistency is incompatible with scheduled replication, an advanced feature we'll discuss later in this chapter. Briefly, scheduled replication permits updates to be deferred to a particular window in time, perhaps to take advantage of reduced bandwidth costs or lower utilization of slow WAN links. Because a strongly consistent system requires that updates be propagated immediately, it is essentially at odds with scheduled replication.

Given all these challenges, weakly consistent replication systems are much easier to implement and provide better performance at the expense of temporary inconsistencies between supplier and replica servers. For virtually all directory applications, this trade-off is perfectly acceptable and represents a well-informed compromise on the part of directory designers.

Incremental and Total Updates

To make two servers consistent, we might choose either to completely replace the contents of the consumer server or to transmit only the minimum information necessary to bring the servers into synchronization. The former approach, termed a *total update* or a *replica refresh*, is useful when you're initially creating a replica (you'll learn more about this creation operation later in this chapter). But always using a total update strategy when updating consumer servers is inefficient because all entries are transmitted even if they have not been modified.

In an *incremental update*, only the changes made to the supplier's directory are sent to the consumer server. For example, if a directory client modifies an entry by replacing its **description** attribute, it is necessary to perform only that same change on all replicas to bring them into synchronization. It is not necessary to send the entire entry, and it is certainly not necessary to transmit the entire contents of the database to all replicas. Incremental updates are much more efficient, and all widely used LDAP directory server software supports them.

Note

If a replica's directory tree is in an unknown state (perhaps it has been damaged or reloaded from an extremely out-of-date backup), it may then be desirable to wipe out any existing contents and perform a total update. This is also what is done when a replica is initially populated with data.

To understand better how the incremental update process works, let's look at the process in general, and then we'll examine how real-world directory services perform incremental updates. Following is an outline of the incremental update process:

- Step 1.** The supplier server connects to the consumer server and authenticates.
- Step 2.** The supplier determines which updates need to be applied.
- Step 3.** The supplier sends the updates to the consumer.
- Step 4.** The consumer applies those updates to its copy of the directory data.
- Step 5.** The supplier and/or the consumer save state information that records the last update applied. This information is used in Step 2 of subsequent incremental updates.

In this way, a supplier transmits only the minimum number of updates necessary to make the consumer server consistent with the supplier. For some more concrete examples, let's examine how a popular directory service—Netscape Directory Server 6—incrementally updates a consumer.

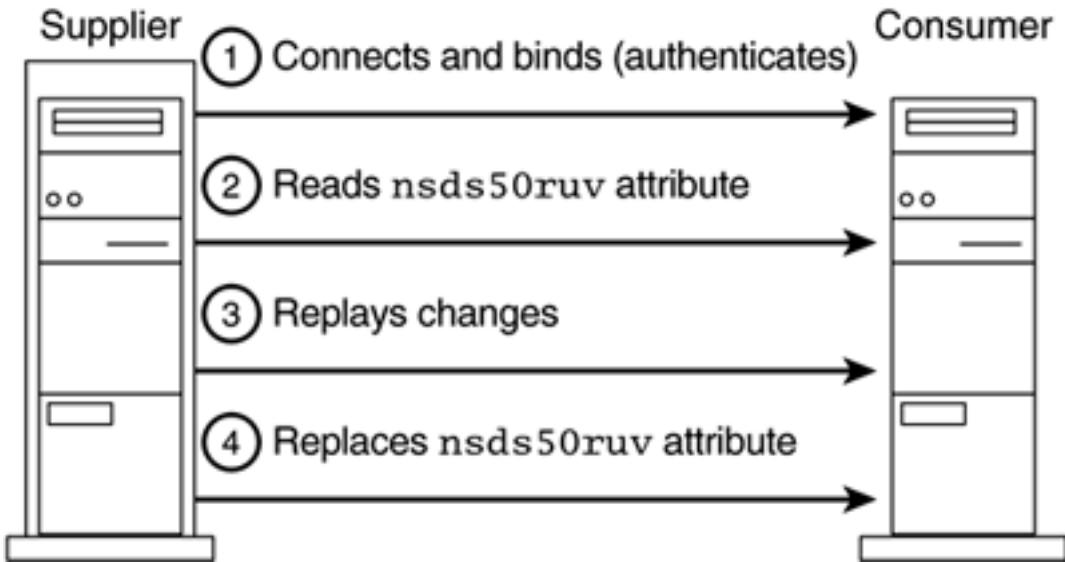
The Netscape Directory Server 6 Update Process

Netscape Directory Server 6 updates consumers by replaying changes it receives. For example, if a client connects to a Netscape Directory Server 6 server and adds a new entry, the supplier connects to all its consumers and adds the same entry. Each change, when received by the supplier, is assigned a unique *change sequence number* (CSN); the change, along with the CSN, is then logged to a *changelog*, a database that records all changes made to the server. The supplier keeps track of the changes it has replayed to the consumer by storing in the consumer's directory tree the CSN of the last change applied.

Netscape Directory Server 6 performs the following steps when incrementally updating a replica, as illustrated in [Figure 11.5](#):

- Step 1.** The supplier server connects to the consumer server and authenticates.
- Step 2.** The supplier reads the `nsds50ruv` attribute in the entry at the top of the replicated subtree. The `nsds50ruv` attribute contains the CSN of the last change replayed to this consumer from every known updatable replica.
- Step 3.** The supplier server replays to the consumer any changes that it has not yet received. It continues to do this until it runs out of changes to replay.
- Step 4.** The supplier server stores on the consumer an updated `nsds50ruv` attribute. The consumer is now consistent with the supplier.

Figure 11.5. The Netscape Directory Server 6 Update Process



The preceding sequence of steps is simplified; the process is actually more complicated. We'll discuss the update process later in this chapter when we cover multimaster replication in detail.

Initial Population of a Replica

When a consumer server is initially configured, it contains no data. The replica must somehow be populated with a snapshot of the supplier's data so that it can subsequently be made consistent. Or in the event that a consumer server has become damaged, the consumer must be made consistent, usually by removal of the damaged data and creation of a fresh copy of the directory data from the supplier.

Tip

A replica that is being initialized cannot service requests until initialization is complete. If it began servicing requests before being completely populated, it might give erroneous results. For example, it might claim that a given entry does not exist when in fact it simply has not yet received the entry from the supplier. Virtually all directory server software automatically takes care of arranging for a replica to be offline during replica initialization. The replica typically issues a referral to the master server or chains the operation to the master. Clients that connect to a replica that is being initialized may experience degraded performance during initialization of a replica.

How is a replica initialized? Directory vendors accomplish this task by various methods, although all are similar. For example, Netscape Directory Server 6 goes through the following process to perform a total update:

Step 1. The supplier server sends a special LDAPv3 extended operation to the consumer that signals the beginning of a total update.

Step 2. The consumer server takes the appropriate directory database (partition) offline and configures itself so that any client operations (searches or updates) are referred to the supplier during the update. The consumer then signals to the supplier that it is ready to accept the total update.

Step 3. The supplier server sends a series of LDAPv3 extended operations that contain the directory data—one entry per extended operation. The consumer server buffers this data and feeds it to a bulk update process that efficiently creates the directory database and all needed indexes.

Step 4. When all entries have been sent, the supplier sends a special LDAPv3 operation to the consumer that signals the end of the total update. It also replaces the consumer's `nsds50ruv` attribute.

Step 5. The consumer server places its database back in the online state and resumes servicing client requests.

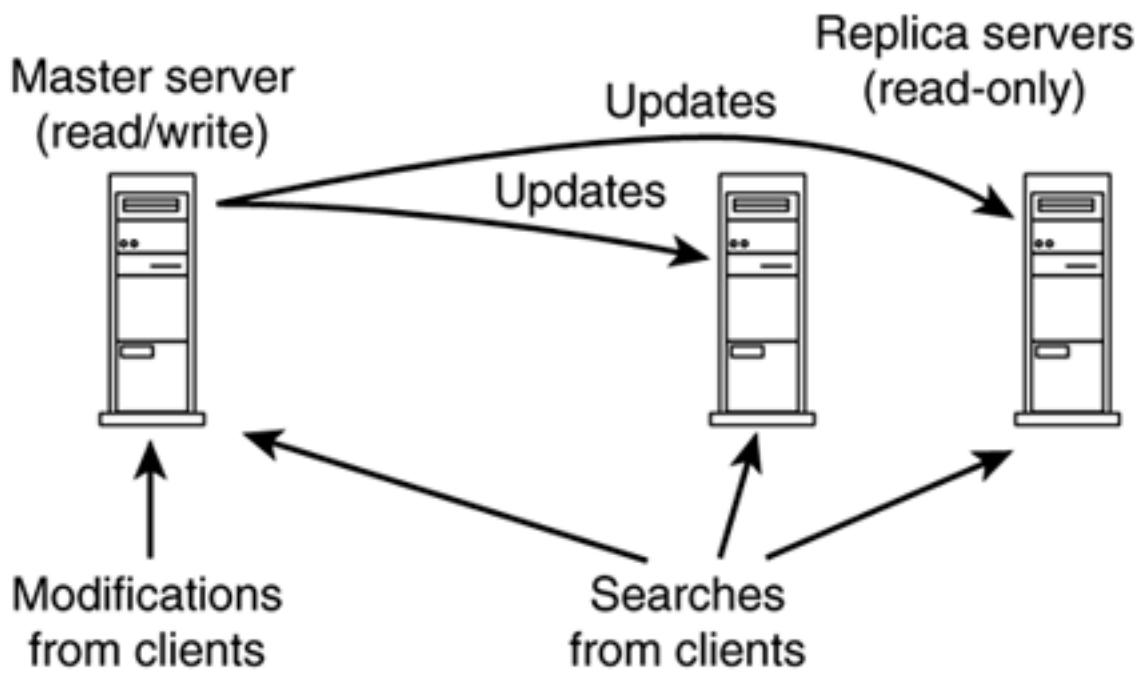
Replication Strategies

The term *replication strategy* refers to the way updates flow from server to server and the way servers interact when propagating updates. There are two main approaches: single-master replication and multimaster replication.

Single-Master Replication

In *single-master replication*, only one server contains a writable copy of a given directory entry. All replicas contain read-only copies of the entry. Whereas the master server is the only one that can perform write operations, any server may perform a search, compare, or bind operation (see [Figure 11.6](#)).

Figure 11.6. Single-Master Replication

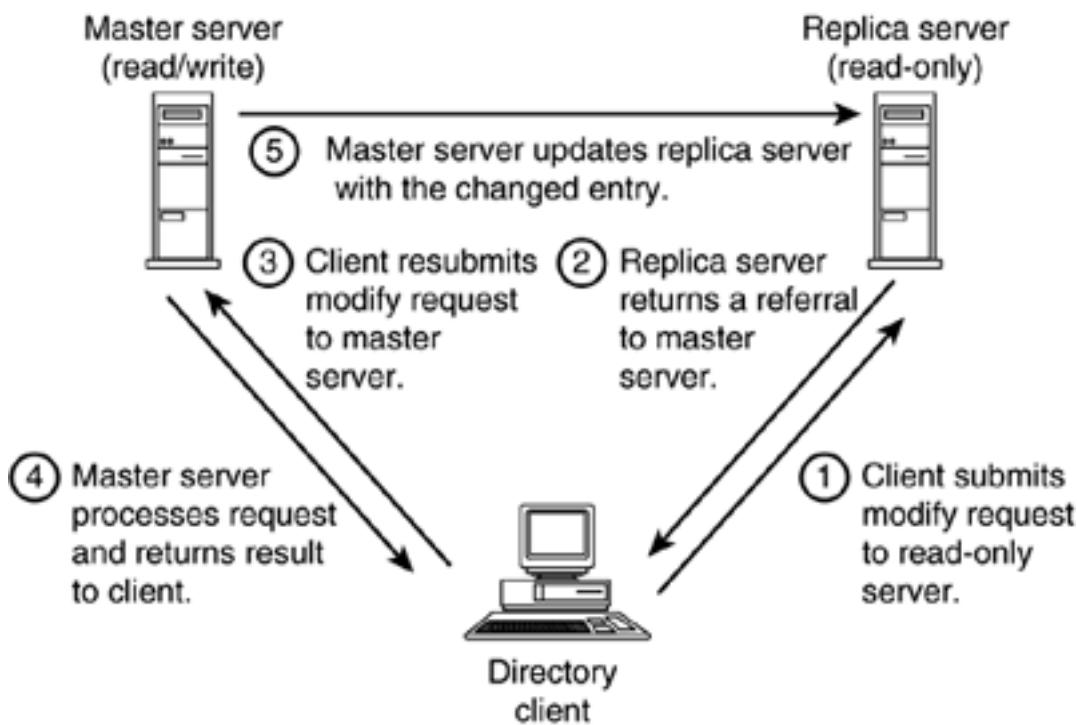


Because a typical directory-enabled application performs many more search operations than modify operations, it's beneficial to use read-only replicas. The read-only replica server can handle search operations just as well as the writable master server can.

If the client attempts to perform a write operation on the read-only server (for example, adding, deleting, modifying, or renaming an entry), we need some way to arrange for the operation to be submitted to the read/write server. There are two possibilities: The first is to submit the operation via a referral, which is simply a way for a server to say to a client, "I cannot handle this request, but here is the location of a server that should be able to."

[Figure 11.7](#) shows the steps involved when a directory client submits a change to a read-only replica.

Figure 11.7. Directing an Update to a Master Server by Using Referrals

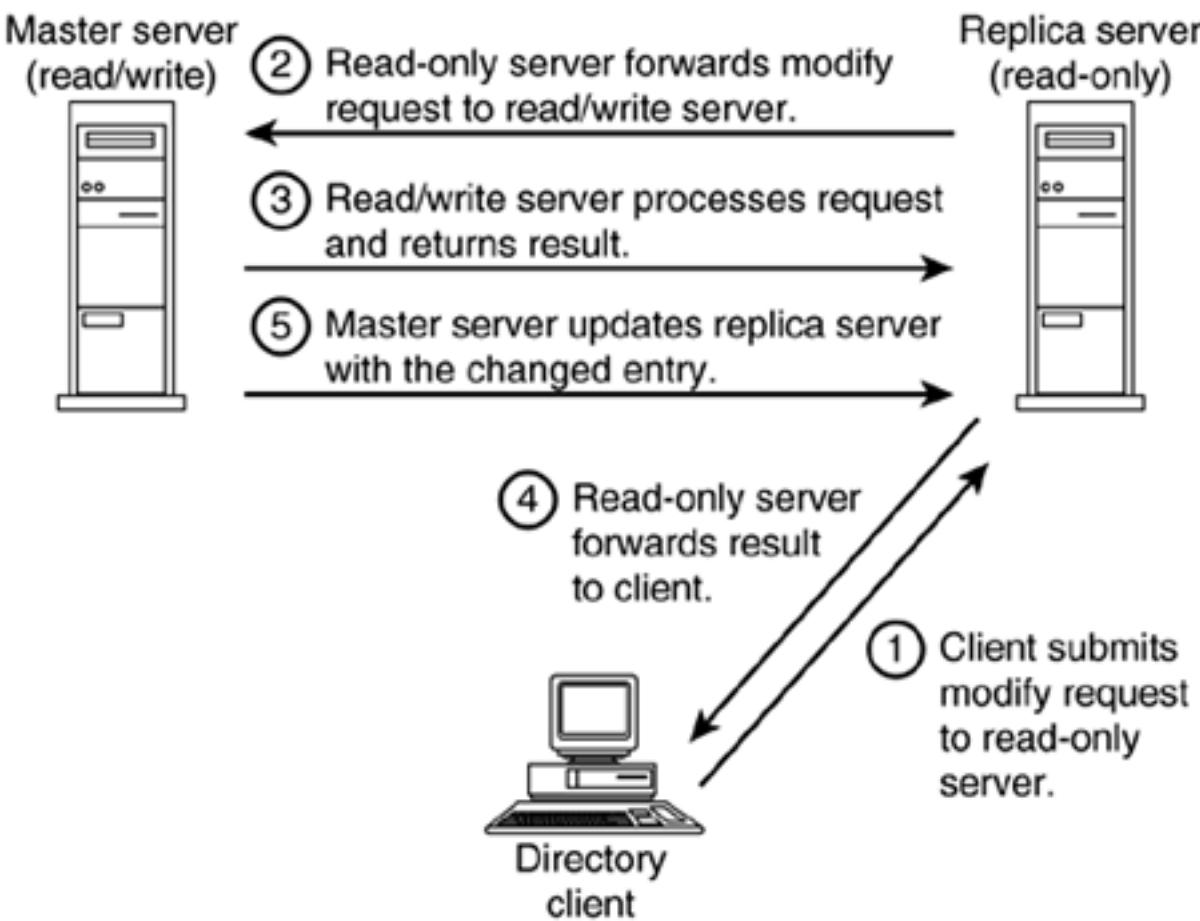


Note

Some directory clients do not automatically follow referrals or do not authenticate properly to the referred-to directory. For example, the `ldapmodify` command shipped with the Solaris operating system will follow a referral generated by a read-only consumer but will not authenticate with the user's credentials when connecting to the supplier server. Instead, the client will bind anonymously and will not have sufficient privileges to perform the update.

The other way to submit a write operation to the read/write copy is by chaining the request. That is, the server resubmits the request, on behalf of the client, to the read/write copy; then it obtains the result and forwards it to the client (see [Figure 11.8](#)). A more thorough discussion of referrals and chaining may be found in [Chapter 10](#), Topology Design.

Figure 11.8. Directing an Update to a Master Server by Chaining



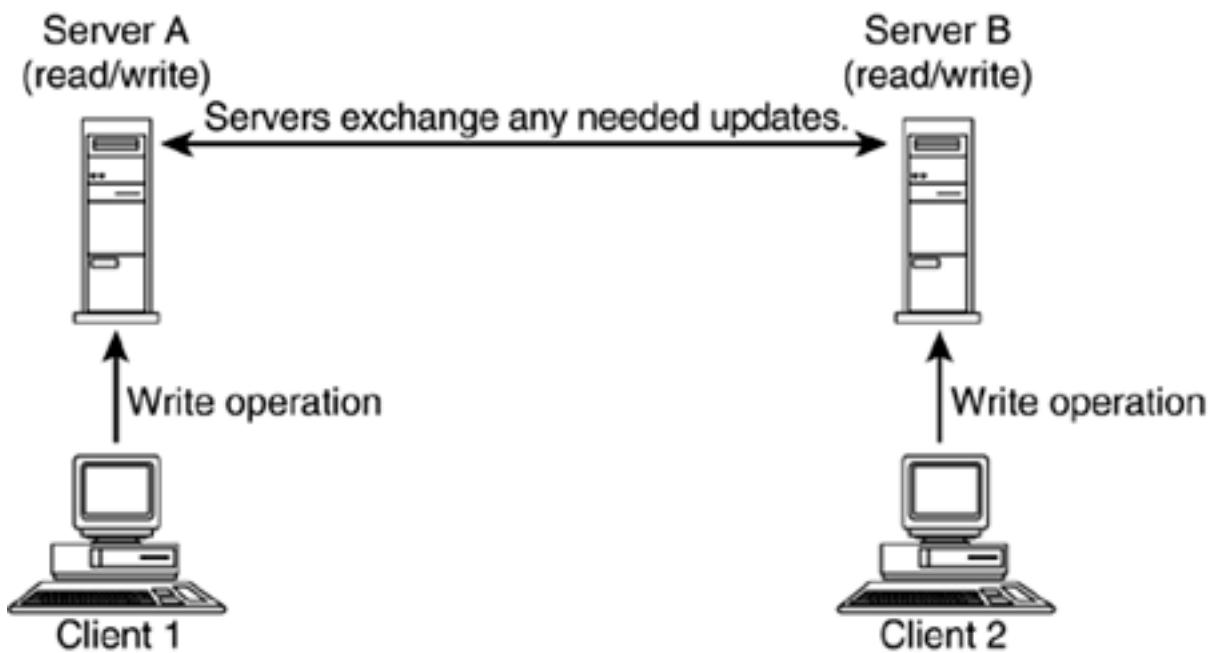
Typically, all these multistep interactions between clients and servers are handled automatically by the application software. Directory client users are unlikely to witness all this; instead they simply see the modify operation complete, and the change is eventually available on the replica. (Note that for a period of time the read/write copy of the server contains newer data than the read-only copy, as mentioned in the earlier discussion on consistency and convergence.)

Astute readers will notice that in a single-master replication system there is a single point of failure: the read/write server. Only one server can process write operations for a given entry; if it goes down, no client can modify that portion of the directory (although search and read operations can continue at read-only replicas). Depending on the type of directory client software and directory-enabled application in use, this may or may not be acceptable. If it is not acceptable, another replication strategy, discussed next, can remove the single point of failure for write operations.

Multimaster Replication

In a *multimaster replication* system, more than one read/write copy of an entry is almost always available. Clients may submit an update operation to any of the read/write replicas. It then becomes the responsibility of the set of cooperating servers to ensure that changes are eventually propagated to all servers and that consistency is maintained. [Figure 11.9](#) shows two replicated read/write servers capable of handling client write requests.

Figure 11.9. Multimaster Replication



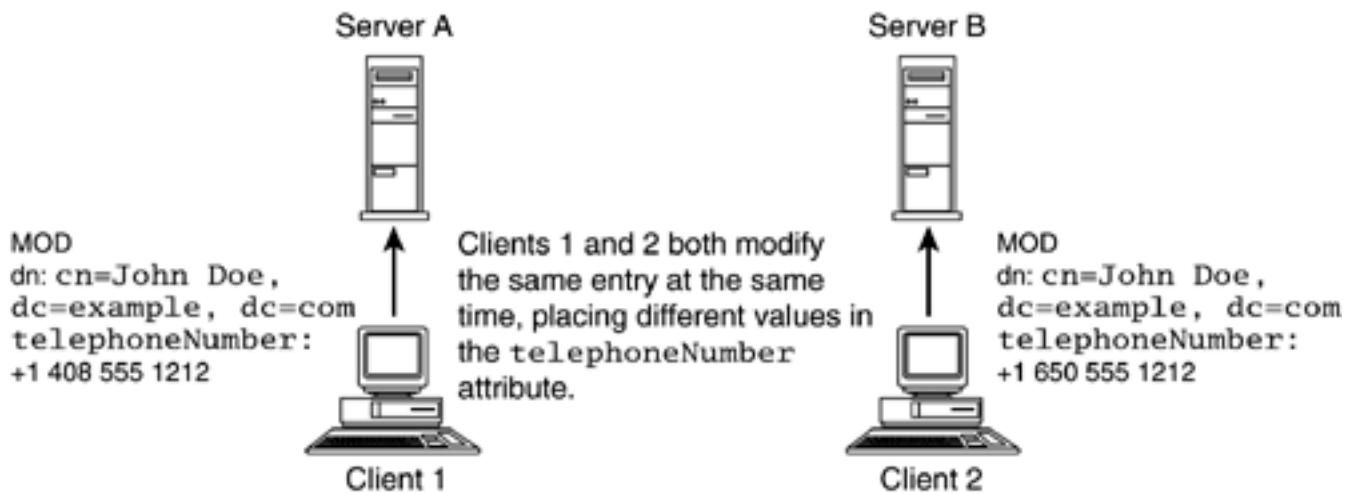
Multimaster replication eliminates the single point of failure for updates and thus offers greater reliability for directory clients. However, allowing more than one server to accept write operations brings additional complexity, most notably the need for an *update conflict resolution policy*. This policy is used to resolve an update conflict, which can occur when an attribute of an entry is modified at the same approximate time on two different master servers.

Conflict Resolution

In multimaster replication systems, more than one directory server may accept modifications for a given entry. Sometimes the result is a situation in which two directory clients modify the same entry on two different servers at the same time. But what happens when the clients modify the entry in such a way that the changes are in conflict?

In [Figure 11.10](#), Client 1 modifies the entry `cn=John Doe,dc=example,dc=com` and replaces the `telephoneNumber` attribute with the single value `+1 408 555 1212`, submitting the change to Server A. At the same time, Client 2 modifies the entry `cn=John Doe,dc=example,dc=com` and replaces the `telephoneNumber` attribute with a *different* value, `+1 650 555 1212`, submitting the change to Server B. After these operations complete on each server, the entries are in conflict: It's impossible for both changes to be retained, so one must be discarded.

Figure 11.10. Setting the Stage for an Update Conflict



Because the cooperating servers are required to converge eventually, we need to invent some way of resolving this conflict. There is really no correct way to resolve the conflict; each client's change is as good as the other's. Of course, each user thinks that his change will be made on all replicas, and he may be somewhat surprised to discover otherwise.

All currently available multimaster directory replication systems use a "last writer wins" policy to resolve such conflicts. Every attribute of every entry in the directory is marked with a unique sequence number that allows a server to determine which update should be used and which update should be discarded. In the next section we'll discuss, in abstract terms, the multimaster update process. After we've introduced some important common concepts, we'll examine how multimaster replication works in Netscape Directory Server 6.

Sequence Numbers

All multimaster replication systems assign a unique sequence number to each update operation received from an LDAP client. Sequence numbers have the following properties:

- They are unique.
- They always increase with time. Every sequence number generated by a server is larger than all previously generated sequence numbers.

These properties ensure that it's always possible to determine the ordering of two updates. To the greatest extent possible, multimaster replication systems try to make this ordering reflect the real-world ordering of events (sometimes referred to as *wall-clock time*). Most systems base their sequence numbers directly on the system clock.

All multimaster systems also provide a way to avoid a situation in which two updates on two different servers are assigned exactly the same sequence number. In some systems, such as Netscape Directory Server 6, each replica is assigned a unique, small integer identifier, and this identifier is inserted into the sequence number. In other cases, including Microsoft Active Directory, a more complicated system is used in which each attribute value is marked with versioning information and timestamps. Whatever the method used, the outcome is that no two updates are ever assigned the same sequence number. Thus, each server in a multimaster environment can make consistent update conflict resolution decisions.

All multimaster replication systems that we are aware of derive their sequence numbers from the server's clock in some fashion. When two servers detect that their clocks are out of sync, they take one of two approaches, depending on the type of directory server software in use. Netscape Directory Server 6 and Novell eDirectory use a special clock for the timestamps that can move relative to the system clock. If a server detects that its clock is behind, it advances its timestamp clock to the largest value seen from other servers. This

time is sometimes referred to as *synthetic time* because it does not accurately reflect wall-clock time. Other systems, such as Microsoft Active Directory, have a maximum allowable clock skew. In Active Directory, if two servers detect that their clocks are out of sync by five minutes or more, they will not replicate data, and an administrator must remedy the problem.

In practice, clock synchronization problems can be avoided through use of a clock synchronization utility. Novell eDirectory and Microsoft Active Directory include software that automatically keeps server clocks in sync. For systems running Netscape Directory Server 6, it is highly recommended that you install Network Time Protocol (NTP) software on the server. NTP can synchronize server clocks to highly accurate external time sources that are freely available on the Internet.

Granularity

Another important concept in a multimaster system is the granularity of sequence number assignment. Whether the system assigns sequence numbers to attributes or to the individual attribute values has an effect on the behavior of a replicated system.

If sequence numbers are assigned to attributes instead of individual attribute values, when resolving conflicts a server must choose between the values of an attribute on one server and the values of the attribute on the other server. For example, if one user adds a member to a group on Server A, and another user adds a different member to a group on Server B, when replicating, the resolution policy can choose only one list of members or the other. It's not possible to merge the lists of members because they don't have any individual sequence numbers. This means that one of the changes will disappear without a trace. The version of Microsoft Active Directory that ships with Windows 2000 suffers from this design problem. Microsoft recommends that administrators alter group membership on a single domain controller to prevent this situation. A future release of Active Directory will resolve this issue.

An alternative approach is to assign sequence numbers to individual attribute values. In this case, finer-grained update resolution is possible. In the previous example, instead of group membership updates being discarded, the member lists of the group would be merged during the conflict resolution process. In general, assigning sequence numbers to individual attribute values results in fewer unpleasant surprises for directory users, at the expense of larger database sizes. Netscape Directory Server 6 uses this approach. We will discuss the specific procedures used by Netscape Directory Server 6 in the section Update Resolution Policies later in this chapter.

Unique Identifiers

Multimaster systems assign a unique, unchangeable identifier to every entry at creation time. The entry's distinguished name (DN) is not acceptable for this purpose because it can be changed with the rename operation. Instead, an identifier is assigned that is guaranteed to be unique for all space and time. There are several variants on this theme, but all are similar to the global unique identifier (GUID) found in the Open Group's Distributed Computing Environment. Although it's technically incorrect to say that GUIDs are guaranteed to be unique for all space and time, the probability that the same GUID will be assigned to two different directory entries is extremely small.

All replication operations that flow between servers in a multimaster environment use unique identifiers when naming entries. For all practical purposes, the DN of the entry is just another attribute of the entry that can change, although there are some special considerations that we will cover in the section titled Update Resolution Policies later in this chapter.

Client Updates versus Replica Updates

In all multimaster systems, updates received from clients are treated differently from updates received from replicas. When a server processes an update from a client, it assigns the update a sequence number before committing the change to its database. It also advances the global sequence number counter so that the next change processed will be assigned a larger sequence number. Microsoft terms these types of updates *originating writes*.

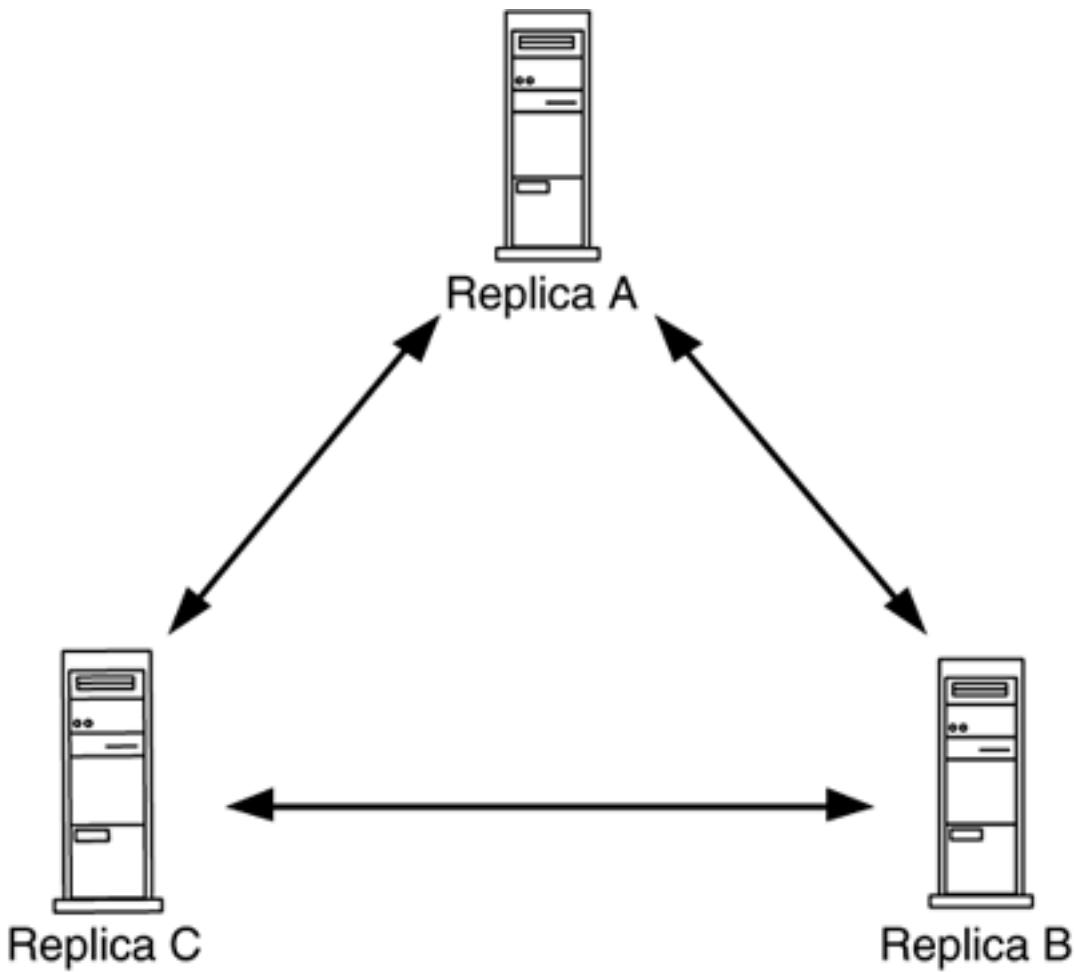
When an update is received from another replica, by contrast, the sequence number arrives with the update (because it was assigned by the replica that originally received the update from a client). A new sequence number is not assigned. This means that sequence numbers on directory data reflect the order in which the changes were originally received from clients, at whichever replicas happened to accept the updates.

Replica Update Vectors

A *replica update vector (RUV)* is a collection of information stored on each replica that describes how up-to-date that replica is with respect to every other replica of that partition. An RUV consists of a series of sequence numbers, one from each updatable replica, that describe the most recent update received from that replica. When one replica sends replication updates to another server, it first retrieves the destination server's RUV and sends only updates with sequence numbers larger than the sequence numbers in the RUV. This means that the minimal number of updates is sent to bring the replica up-to-date.

Another, more subtle benefit, is that RUVs allow the construction of more complicated replication topologies. For example, consider the replication topology shown in [Figure 11.11](#). Notice that the three replicas are fully connected; each replica has a replication agreement with the other two. An originating write made at Replica A can arrive at Replica B via one of two paths. Either it can be sent directly to B, or it can be sent to C, which forwards it to B. Depending on the scheduling of replication sessions, either case might happen.

Figure 11.11. Three Servers—A, B, C—Fully Connected



Because each server has an RUV that records the largest sequence number seen from each of the other replicas, updates are never sent unnecessarily. As illustration, suppose that the following sequence occurs:

Step 1. An originating write (w) is received at Replica A. It is assigned a sequence number (s).

Step 2. Replica A begins a replication session with Replica B. It sends w to B. The RUV on B now records the fact that it has seen all updates with sequence numbers up to s from Replica A.

Step 3. Replica B begins a replication session with Replica C. It sends w to C. The RUV on C now records the fact that it has seen all updates with sequence numbers up to s from Replica A.

Step 4. Replica A begins a replication session with Replica C. When A examines the RUV on C, it discovers that C has already received updates up to sequence number s from it. Therefore, it avoids sending update w .

Microsoft calls this property *replication dampening* because it suppresses (dampens) the unnecessary transmission of replication updates.

Another benefit of allowing redundant replication agreements is that the system is more resistant to network failures. For example, even if Replica A loses contact with Replica C, if it can still contact Replica B, and Replica B can contact Replica C, updates can still get from A to C via B.

Note

Although Netscape Directory Server 6 uses a robust multimaster replication algorithm, it is currently certified for use only where there is a maximum of two updatable replicas. The intent is to allow the updatable replicas to be configured as a highly available pair, such that the failure of one server will not prevent clients from making updates. It is not recommended that you configure more than two updatable servers in your replication configuration, although you are not prevented from doing so. Future versions of Netscape Directory Server may remove this restriction.

Update Resolution Policies

As mentioned previously, all multimaster replication systems in use today attempt to provide "last writer wins" semantics when resolving update conflicts. This policy makes sense because it most closely matches the behavior of a single-master system. If you are using a single server, and you update an entry's e-mail address at 1 P.M. and then you observe someone else updating the entry's e-mail address 30 seconds later, you will expect your change to be overwritten.

The algorithms for resolving conflicting updates to an existing entry are relatively straightforward. Generally, the state of the attribute observed by the last writer is propagated to all servers. However, other conflict scenarios are more difficult to resolve. Let's examine several of these.

Entry Naming Conflicts

Suppose that two people create the entry `uid=jsmith, ou=people,dc=example,dc=com` on two different servers simultaneously. When the servers replicate the changes to one another, the entries are in conflict. The two entries may or may not refer to the same person. Perhaps one person is Jane Smith, and the other is John Smith. It's impossible for the replication system to know. Because each entry in an LDAP directory must have a unique DN, it's impossible to keep both entries with their original names.

The update resolution policy used in this case is to rename one of the entries so that its DN is different. But which entry should be renamed? If the same scenario occurred with a single server, the attempt to add the second entry would have failed with an "entry already exists" error. Therefore, the most natural policy is to rename the entry with the larger sequence number, which would be the entry added later. Netscape Directory Server 6 renames the entry by prepending its unique ID to the RDN of the entry. For example, if the second entry added has been assigned the unique identifier `abef601f-1dd111b2-8084bc31-2dc0fde3`, then the renamed entry will have the DN `nsuniqueid=abef601f- 1dd111b2-8084bc31-2dc0fde3+uid=jsmith,ou=people,dc=example,dc=com`. The renamed entry will also be marked with the operational attribute `nsds5ReplConflict`. You can locate all entries that were altered because of a replication conflict by using the search filter `nsds5ReplConflict=*`.

It's also possible for an entry naming conflict to arise as a consequence of simultaneous add and rename operations, or simultaneous rename operations. If the new names of the entries are the same, a naming conflict has occurred. For example, if one user adds a new entry with the name `uid=jsmith,ou=people,dc=example,dc=com` on one server, and another user renames the entry `uid=jjones,ou=people,dc=example, dc=com` to `uid=jsmith, ou=people,dc=example,dc=com` on another server, the result is two entries with the same name. The same policy of renaming the entry with the larger sequence number is also used

in this case.

Conflicts Involving Deleted Entries

Some types of multimaster update conflicts arise because an entry has been deleted on one server, and before that update can propagate to other servers, the entry is deleted or has subordinate entries added beneath it. Consider the following example:

The entry `ou=Engineering,dc=example,dc=com` is added by a directory administrator. This update is submitted to Server A, and multimaster replication propagates this update to all other servers.

A departmental administrator, seeing that the Engineering department's entry is now in the directory, begins adding employees to that department. This administrator submits the updates to Server B.

The directory administrator realizes that the department name should have been "Research Engineering." She deletes the Engineering entry and creates a new entry named "Research Engineering," performing the operation on Server A.

Servers A and B now replicate with one another. The conflict is that the entries added by the departmental administrator have been "orphaned" because their parent entry was deleted. Because every entry in the directory (except the root entry) must have a parent, we need to apply a policy to ensure that the orphaned entries again have a parent.

There are several possible approaches to solving this problem. One is to resurrect the deleted parent and mark it in such a way that an administrator can discover its existence. Another is to move the set of orphaned entries into a special lost-and-found directory container that holds orphaned entries. Netscape Directory Server 6 resurrects the deleted entry in such situations.

Clearly, to allow resurrection of a deleted entry, a server must keep track of entries that have been deleted. Many vendor implementations use a special type of entry named a *tombstone entry* for this purpose. When an entry is deleted, it is not physically removed from the database. Instead, it is converted to a tombstone entry. Such entries are not visible to LDAP clients and are used only by update resolution procedures.

To prevent the database from becoming unnecessarily large, tombstone entries are typically purged on some sort of a schedule. After a tombstone entry has been purged, all records of the entry's existence are gone. Problems can arise if an updatable replica is disconnected from its replication partners for a period longer than the tombstone purge interval. In that case the disconnected replica may contain entries that have been deleted, tombstoned, and purged from the other replicas. Deleted entries can be reintroduced in this case. If you know that a replica has been disconnected for a long time, it's often a good idea to reinitialize the replica before reconnecting it.

Conflicts Involving Single-Value Constraints

When an attribute is marked as single-valued in the schema, a directory server will reject any attempt to add more than one value to the attribute. However, consider what happens when two clients simultaneously add a value to an initially empty attribute and submit those operations to two different servers. Each server individually allows the update because the single-valued nature of the attribute is not violated. When the server replicates the update, however, it's impossible to accommodate both client updates because the resulting state of the attribute would contain two values. To resolve this conflict, Netscape Directory Server 6

discards the value with the smaller CSN. The value that will appear in the entry is the one added later.

Replication Protocols

A *replication protocol* is the flow of information over the network that directory servers use to send replication updates. At the time of this writing, efforts are under way in the Internet Engineering Task Force (IETF) to develop a standard, vendor-independent replication protocol. However, there is currently no common replication standard that allows you to directly replicate from one vendor's directory server to a different vendor's. If you do need to synchronize directories from different vendors, you will need to develop your own synchronization tools or use one of the commercially available metadirectory solutions. For more information, see [Chapter 23](#), Directory Coexistence.

Because there is no replication protocol standard, each vendor implements its protocol slightly differently. Netscape Directory Server 6 uses a replication protocol based on LDAPv3. The basic LDAP protocol has been augmented with controls and extended operations that carry the extended information required for multimaster replication to operate.

Microsoft Active Directory supports two different replication protocols. The more common protocol is a proprietary remote procedure call (RPC) protocol that is very efficient, especially when Active Directory replicas have good network connectivity. Active Directory also supports a replication protocol that sends replication updates as Simple Mail Transfer Protocol (SMTP) messages. Replication updates transmitted via SMTP can pass through store-and-forward message networks and can be used when replicas are less well connected.

Novell eDirectory uses a proprietary replication protocol, and X.500 supports a single-master replication protocol called Directory Information Shadowing Protocol (DISP). X.500 does not currently support multimaster replication.

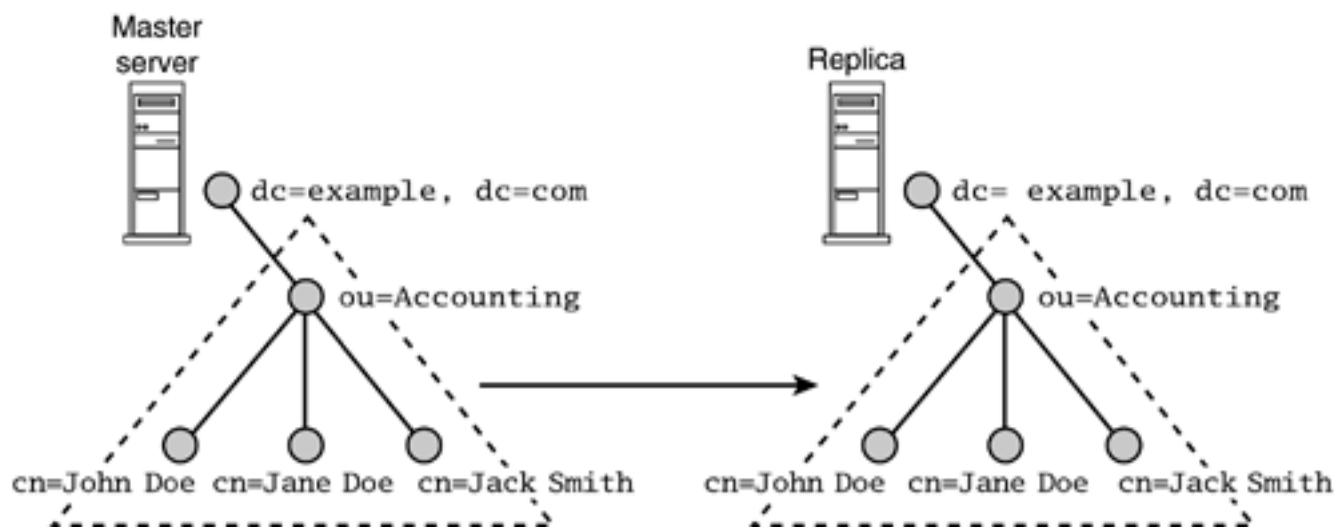
Advanced Replication Features

The advanced replication features described in the following sections are found in some, but not all, directory implementations.

Replicating a Subset of Directory Information

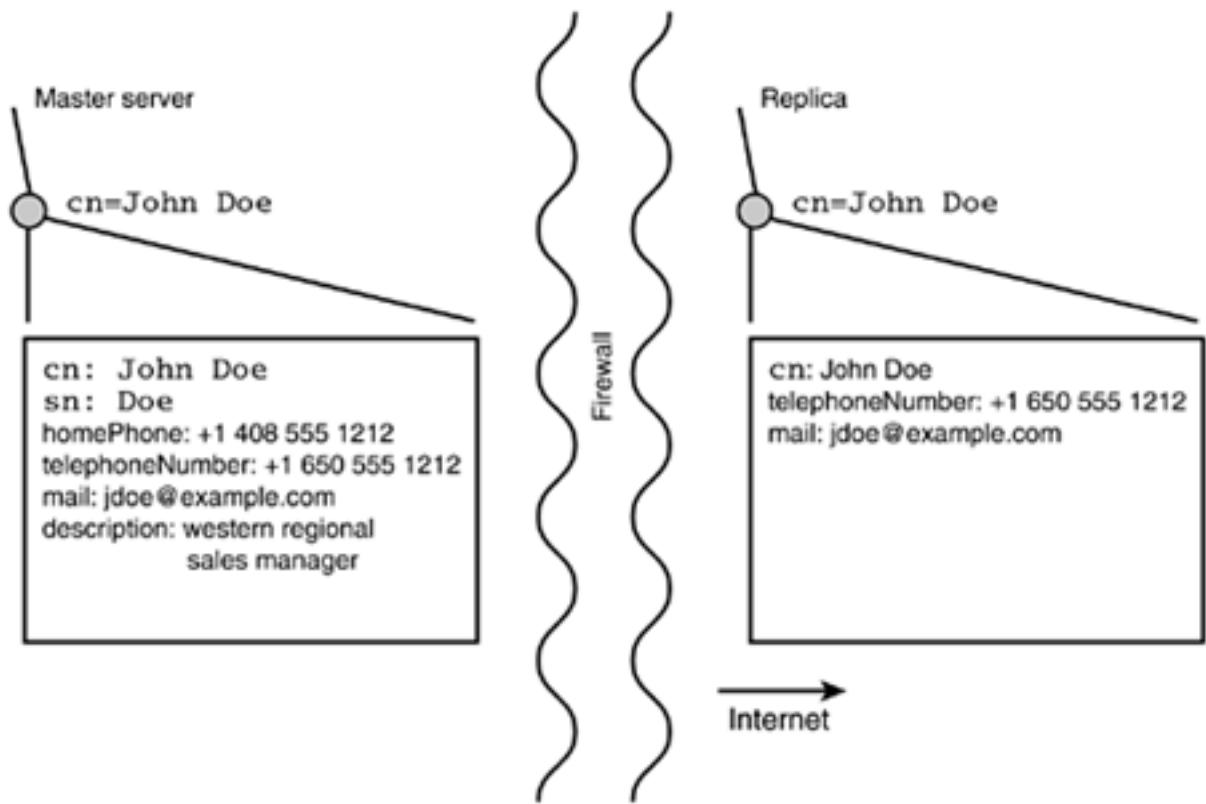
In most cases, replication is used to provide complete copies of a partition. For example, in [Figure 11.12](#) the complete subtree rooted at `ou=Accounting,dc=example,dc=com` is being replicated.

Figure 11.12. Replicating an Entire Subtree



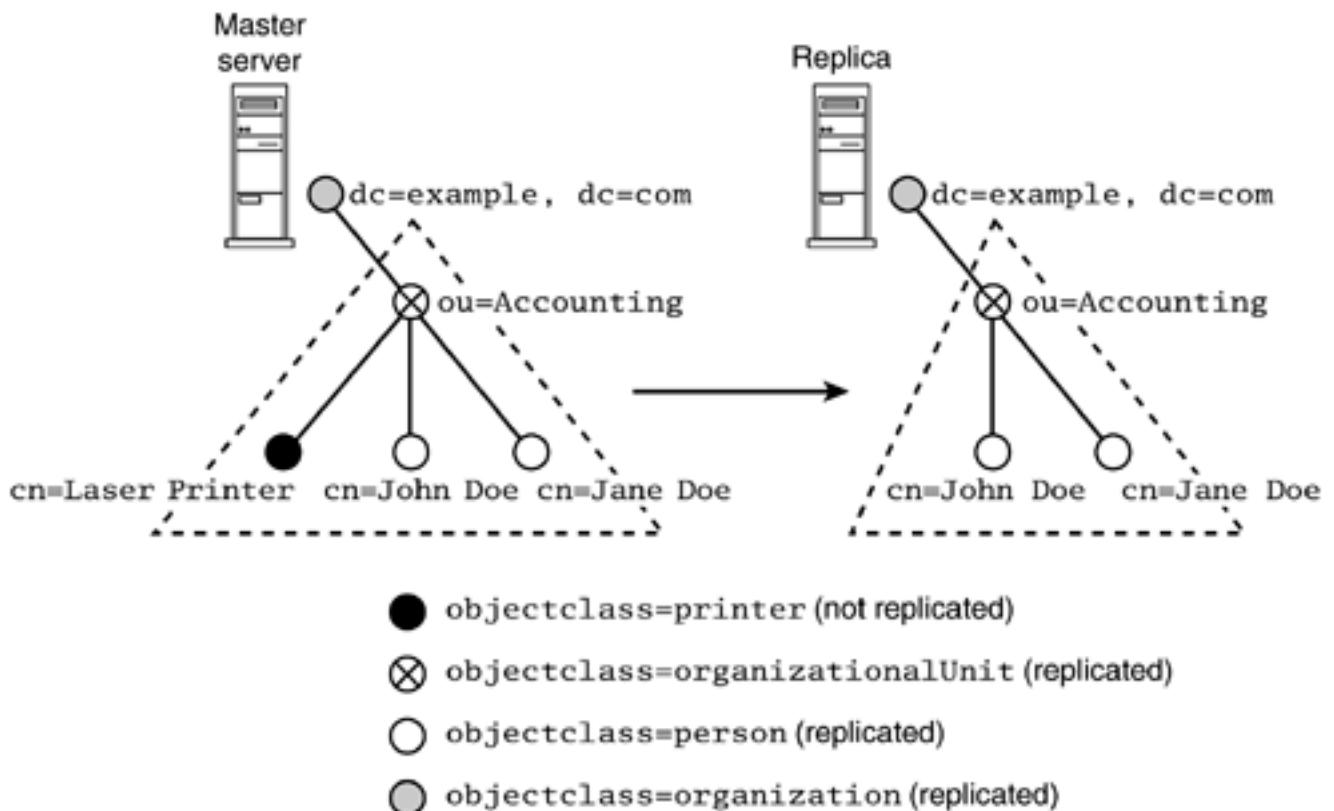
In some cases, however, we might be interested in replicating only certain attributes of the entries. For example, when providing a publicly searchable directory of employee information outside a corporate firewall, an organization might elect to replicate only full names, e-mail addresses, and office telephone numbers and omit all other personal information. Notice in [Figure 11.13](#) that the copy of John Doe's entry accessible outside the firewall contains fewer attributes than the master entry inside the firewall. A replica that contains a subset of the attributes found on a complete replica is called a *fractional replica*.

Figure 11.13. Replicating Only Selected Attributes from an Entry



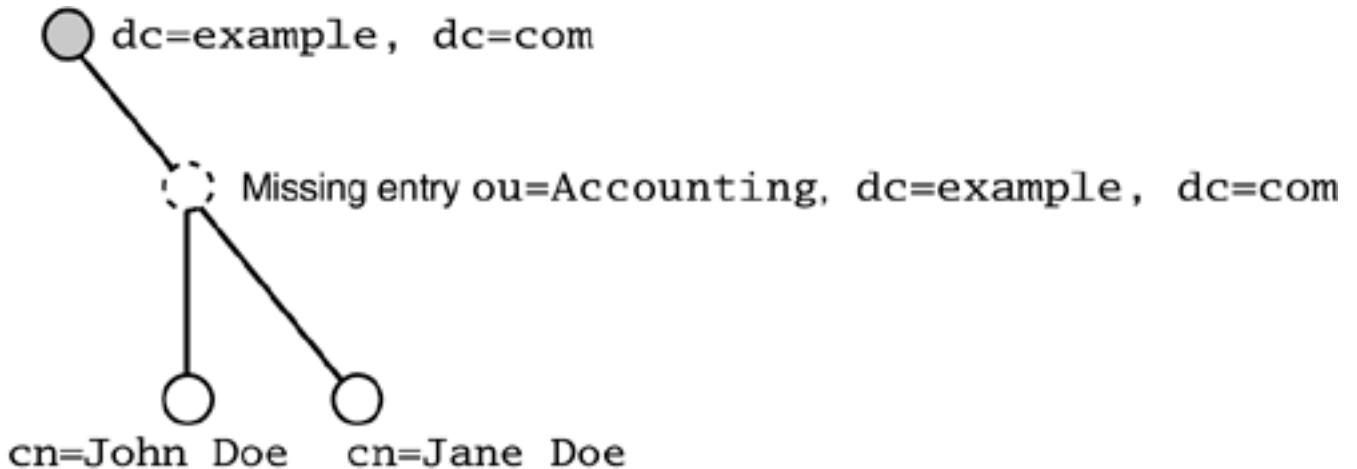
We might also be interested in selecting only certain entries from a subtree. A reasonable thing to do would be to select entries on the basis of their object class. For example, we might want to replicate only those entries that represent people or organizational units. In [Figure 11.14](#), the root of the replicated subtree is once again `ou=Accounting,dc=example,dc=com`, but only the `organizationalUnit` and `person` entries are being replicated. A replica that holds a subset of the entries of another replica is called a *sparse replica*.

Figure 11.14. Replicating Selected Entries



One potential complication with sparse replication is that the replicated directory may contain "holes." In the example depicted in [Figure 11.14](#), if entries of `objectclass organizationalUnit` had not been selected, the replicated tree would look like the one shown in [Figure 11.15](#). To be a valid directory tree, every entry except the root entry must have a parent; however, the consumer's directory tree violates that rule. To remedy this situation, the supplier could create on the consumer a placeholder in place of the entry that was not replicated. The X.500 model describes a specific type of placeholder, termed a *glue entry*, used for just this purpose.

Figure 11.15. A Hole in the DIT Arising from Sparse Replication



Although sparse and fractional replication are simple concepts, they are quite difficult to implement in a multimaster environment. Novell eDirectory is the only directory we are aware of that supports general-purpose fractional and sparse replicas. Microsoft Active Directory, however, supports a special type of fractional replica called the global catalog (GC).

Active Directory GC Servers

Microsoft Active Directory uses fractional replication to create a special replica called the *global catalog*, or *GC*. The GC is a directory that contains a subset of the attributes in each directory entry in all the Active Directory domains. Typically at least one GC is installed at each Active Directory site (an Active Directory site is a zone of high-speed network connectivity within a Windows 2000 network). The GC is used by Active Directory clients within the site when they need to search across the entire Active Directory installation.

Without the GC, an Active Directory client that wants to search across all domains must contact each domain (via referrals) and aggregate the results. With the GC, however, the client can direct its search operations at the GC server, as long as the attributes it needs to search have been replicated to the GC. For common Windows operations, such as logging in and evaluating group membership and access control rights, the GC contains all required attributes.

Scheduling Replication

In some cases a directory administrator might not want changes to be sent to all replicas immediately. For example, a remote office connected via a dial-up link might be better served if updates could be transmitted in one batch to save connect-time charges. Alternatively, a remote office connected via a slow network link might be configured to have updates propagated to a consumer server during off-hours to improve network response for other applications during working hours.

Unfortunately, scheduling replication in this fashion means that users connecting to the consumer will see old data for potentially long periods. For many applications, this is perfectly acceptable; but think carefully about any requirements you might have down the road. For example, if an employee is terminated, it may be necessary to reset his password immediately so that he cannot log in. However, if the directory is not scheduled to be updated for another eight hours, the password revocation will not make it to the consumer immediately, as required.

Scheduling Update Latency by Attribute Type

One solution to this latency problem is to incorporate a scheduling policy that propagates changes to certain attributes immediately and others less rapidly. With Novell eDirectory, for example, whereas certain attributes, such as login passwords, are propagated on a fast synchronization schedule, other attributes, such as last login timestamps in user entries, are scheduled for update on a slow synchronization schedule. This feature seeks to improve update time for critical values and defers other updates to conserve network bandwidth and server processing time.

Schemas and Replication

The purpose of replication is to provide copies of directory data in multiple physical locations. It makes sense, therefore, that supplier and consumer servers should have the same schema. Serious problems would arise, for example, if a supplier server attempted to add an entry of an object class not allowed by the consumer's schema (the operation would be rejected, and the consumer could never be brought into synchronization).

Different vendors take alternative approaches to solving this problem. Netscape Directory Server 6 has one global schema for all replicas held by a server. The schema is marked with a sequence number that represents the last time the schema was updated. Before commencing a replication session, a Netscape Directory Server 6 supplier checks whether the consumer's schema sequence number is older than its own. If it is, it sends the updated schema to the consumer and then commences the normal replication update. Schemas may be changed at any updatable replica. If two schemas are in conflict, the newer schema is used in its entirety (attribute-specific value conflict resolution is not performed), and an updated schema will eventually propagate to all servers.

Microsoft Active Directory solves the schema replication problem by using single-master replication for schemas. One server in the domain is designated as the schema master. All updates must take place at the schema master. No conflict resolution policies are required because they cannot occur in a single-master replication system.

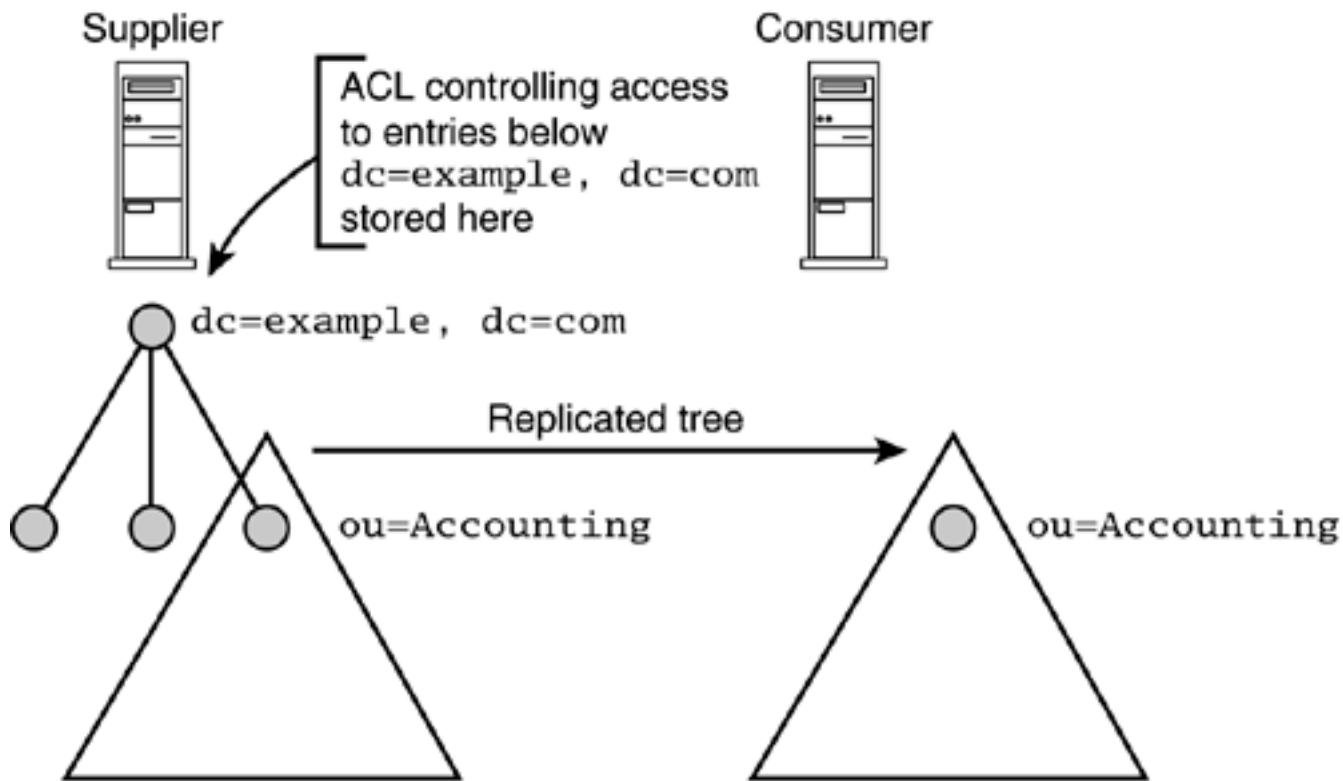
Access Control and Replication

Virtually all directory products offer some way of controlling access to the data contained in the directory tree, usually via an *access control list (ACL)* mechanism. When the contents of the tree are replicated, it is desirable also to replicate any associated ACL information so that the same protections apply to both the replicated data and the original data. More information about access control can be found in [Chapter 12](#), Privacy and Security Design.

Most, if not all, directory software stores ACLs as attributes of entries. Usually this means that the ACLs merely need to be replicated along with other directory content. As long as the supplier and consumer server use the same ACL syntax, and as long as those ACLs mean the same thing on both servers, their directory entries will have the same access control.

Again, in most cases it's sufficient to replicate ACLs along with directory content. The one time this gets a bit tricky is when ACLs have a scope that extends down the directory tree and crosses a unit of replication. In [Figure 11.16](#), the entry `dc=example, dc=com` on the supplier server contains an access control directive that applies to all entries below it. However, this entry is not contained within the replicated subtree, so the consumer server lacks the access control information it needs to properly control access to its replicated subtree.

Figure 11.16. Access Control Information Stored above a Replicated Subtree



Netscape Directory Server 6 allows you to configure replication in this manner, so be sure when designing your replication strategy that you include ACLs at the top of all replicated subtrees. (The default in Netscape Directory Server 6 is to completely deny access in the absence of ACL information, so a problem like the one in [Figure 11.16](#) would not expose any directory information to unauthorized access.)

Designing Your Directory Replication System

To maximize the reliability and performance of your directory service through replication, spend some time planning for both your current requirements and future expansion. To begin your planning, gather the following information:

- A network map of your organization, showing all locations and the types of network connectivity between them. Label your network map in terms of zones of high connectivity and low connectivity, and label the reliability of long-haul network links.
- An inventory of any directory-enabled applications you are running or intend to run. For example, LDAP-enabled messaging products such as Sun ONE Messaging Server and Enterprise Web Server use the directory for authentication and access control. If you know the physical locations of these applications and clients, note them on your network map. If you don't yet have a plan for deploying these servers and clients, that's OK; estimate as well as you can and revisit your assumptions as you gain experience through the piloting phase of your deployment.

Each vendor offers deployment and planning guides that help you plan the optimal replication design for your environment, and you should consult those guides. However, a few general principles apply to all replication designs:

- Understand your zones of high connectivity and high network reliability, and how they are connected.
- Place enough replicas in each high-connectivity zone to handle the load in that zone.
- Try to minimize traffic over the links that connect the zones.

As we work through the design process, realize that it's easy to overengineer a replication solution. Think about airplanes: Very few have ten engines; most commercial jets these days have two because that's been found to be an optimal number, given fuel costs, plane sizes, and engine reliability. Now think about your directory: Although it's certainly possible to put three redundant directory servers on every Ethernet segment in an office building—which would definitely increase your directory's reliability—don't forget that someone has to set up and manage all those replicas!

In an office in which all workstations and servers have 10Mbps or better connectivity, reliability is probably so good that it's unnecessary to worry about network failures. Even if that's not a valid assumption, it's probably still cheaper to fix any network problems than it is to set up and manage many directory replicas. On the other hand, if your organization is spread over a wide geographical area and your sites are linked by slower, less-reliable network links, you should definitely pay close attention to how you allocate replicas across your network.

Finally, view this design process as iterative. Make a pass through it, thinking about maximizing reliability and performance. As your solution evolves, you may find that you want to revisit a design decision. In addition, as you think about replication, you may want to revisit some of your design decisions about directory topology and even your namespace design. Don't be afraid to do this; it will be time well spent.

Designing for Maximum Reliability

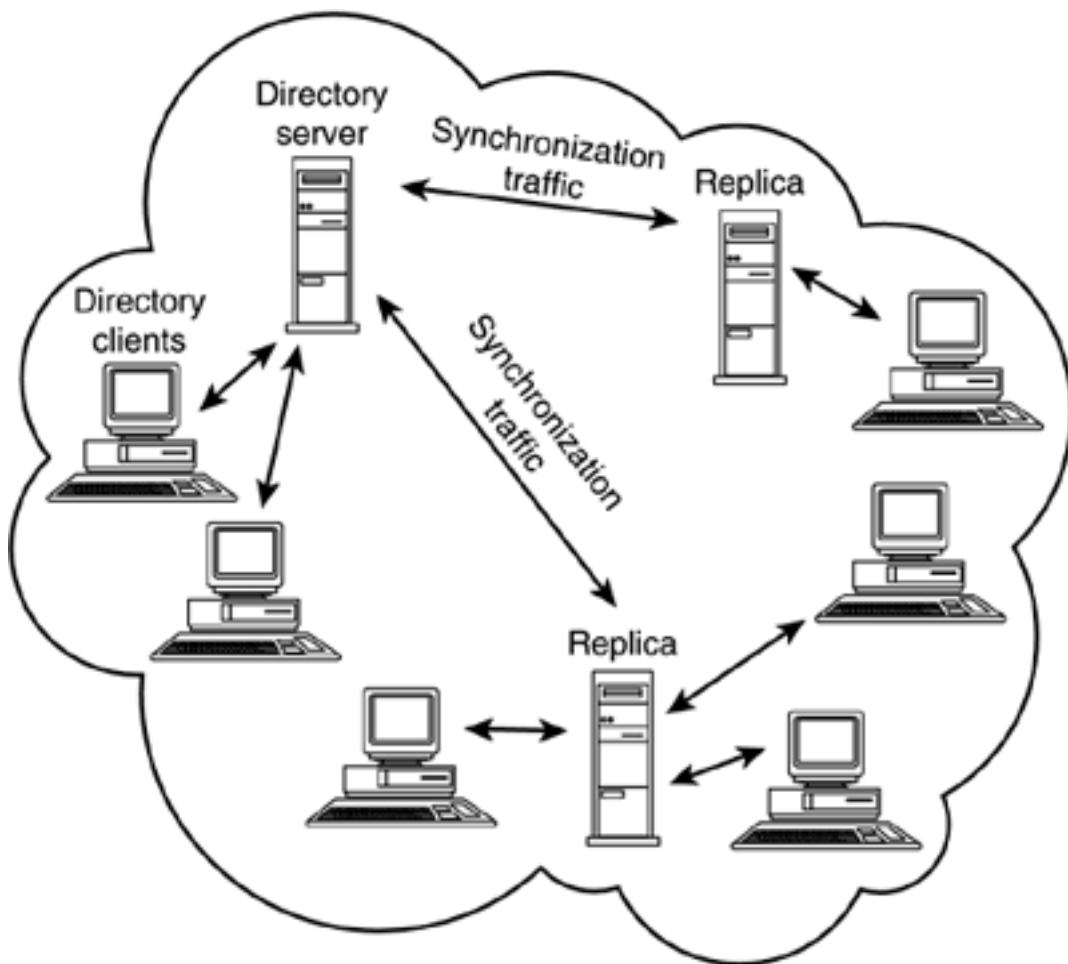
When you design for maximum reliability, you make your directory impervious to the failure of a single directory server. Then if one of your servers fails, directory clients can use another replica to obtain their directory services.

How exactly do directory clients deal with the failure of a particular server? LDAP client applications are responsible for detecting the failure of their primary server and reconnecting to an alternate server. At present, there is no standard method for locating an alternate server that can provide service, so it's useful to ask the supplier of an LDAP-based application how this is handled.

For example, client applications that use the Netscape LDAP C software development kit (SDK) can provide multiple server names when establishing a connection; if a given server is unavailable, the SDK tries another. Another option is to use a hardware failover device such as Cisco Systems' LocalDirector or Nortel Networks Alteon Link Optimizer, which can balance client load across multiple servers. It can also detect when a server has failed and avoids directing clients to that server until it is returned to service. Of course, the hardware failover device itself can fail, as can any of the network devices connecting it to the directory servers and the network. A complete solution for high availability makes all components redundant, including servers, load balancers, network switches, network routers, and power supplies. Whether the cost for this level of redundancy is justified will depend on your particular needs.

To maximize reliability, locate within each major zone of your network at least one replica connected via a network link of less than 10Mbps. (If you have a single well-connected network, you have only one zone to worry about.) For example, if your network comprises a single set of buildings connected by high-speed fiber-optic links, you might choose to deploy two replicated servers. If either server fails, client requests will be handled by the remaining server (see [Figure 11.17](#)).

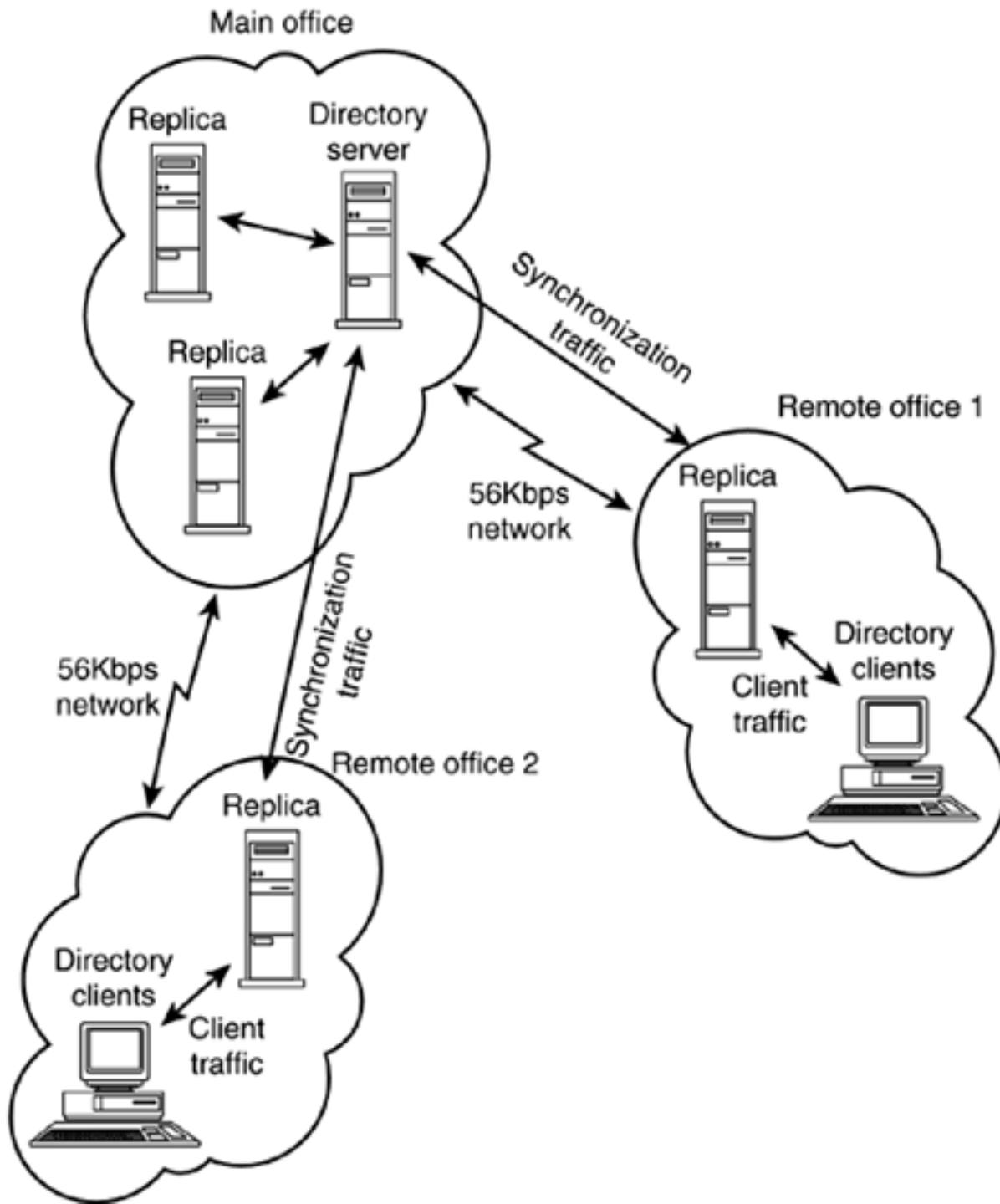
Figure 11.17. Multiple Replicas Located at a Site



Suppose now that your organization consists of a central office with a series of remote offices connected by slower, 56Kbps network connections. In this case you might choose to

place a replica at each remote site (see [Figure 11.18](#)) so that you can avoid wasting your scarce network bandwidth on LDAP client traffic.

Figure 11.18. Replicas Placed to Limit Client Traffic on WAN Links



Under normal circumstances, the directory clients shown in [Figure 11.18](#) in the remote offices contact the onsite replica for directory operations. If the server fails, the remote clients can contact one of the servers in the central office. You could even place more than one replica in each remote office so that even if one of the onsite servers fails, the clients can obtain directory service without sending requests across the slower interoffice network link. If you use such a configuration, you may want to schedule replication updates to occur only during off-peak hours, if your directory server software supports that option.

When you design replication to enhance performance, you should strive to design a system that can handle your existing client load today but can be expanded easily to handle a larger load in the future.

Strive to provide a sufficient number of replica servers to handle your client load. Estimating your client load involves a bit of guesswork, but you should be able to get a good idea by understanding how often a given client makes a request and how a typical request from that client would affect the server. Then multiply this figure by the number of clients you expect to use the directory.

For example, suppose that 1,000 users will use a Web-based address book application to look up other employees in the directory. Begin by making some assumptions about how often people will use the service. (These assumptions can be verified, and adjusted if necessary, during the pilot phase of your deployment; see [Chapter 14](#), Piloting Your Directory Service). Let's assume that each person will perform 10 lookups during the 8 A.M.–to–5 P.M. workday; that means you should expect to see 10,000 queries in eight hours. This translates into 1,250 queries per hour—approximately one query every three seconds. If additional knowledge leads you to believe that usage will not be uniform, but instead will have spikes during the day, consider that as well.

Next understand the impact that the load will have on the directory server. Are the queries typically made on indexed attributes—which perform well and place a lighter load on the server—or might some of the queries be on unindexed attributes? Will the searches typically return a single entry or many entries? Will the retrieved entries contain large attributes, or will the amount of transmitted data be small (perhaps up to a kilobyte or two)?

Finally, determine how many replicated servers you will need to handle the load. You may be fortunate enough to have some data from the software vendor that tells you how many typical queries the software can service while running on standard hardware in a given time period. For example, Netscape Directory Server 6 can perform from several hundred to thousands of searches per second on a typical server-class computer, so a single server would be entirely capable of handling the client load in the previous example. If, however, you don't really know how many operations the server can perform per second, you should measure that factor when evaluating server software or piloting your directory.

Tip

When you're doing capacity planning, always take vendor-supplied performance figures with a grain of salt. Performance figures that appear in vendor data sheets reflect the performance measured by the vendor on one system with one particular set of assumptions. Performance may be affected by many other factors, including the operating system platform, amount of available memory, speed of disk drives, namespace design, complexity of access control rules, characteristics of LDAP clients, and much more. If possible, measure your systems under the types of loads you expect them to handle to get a better idea of individual server performance capabilities before planning your total number of replicas.

Consider write performance of your directory separately from search performance. In a replicated environment, each change made by a client needs to be made at each replica as well. For this reason, write performance in a replicated environment does not scale as well as search performance does. If your application modifies the directory frequently, you may find it advantageous to partition your directory so that each server needs to handle fewer modifications. Remember that directories are typically optimized for searching, so don't be surprised to discover that your software can perform only a few write operations per second.

—even if it can perform hundreds or even thousands of indexed searches per second.

Another way to improve your directory's performance is to provide and tune dedicated servers for particular applications. For example, a busy Sun ONE Messaging Server imposes a heavy load on a server as it goes about its business of delivering mail. However, it uses only LDAP equality search filters (as opposed to substring or approximate filters). You could create a Netscape Directory Server 6 replica that is dedicated to servicing the messaging server's queries. Because you know that substring and approximate indexes are not required on the server, you could remove them from the server's configuration, therefore improving its write performance and reducing the memory requirements.

One other consideration is how your directory clients will know that multiple servers are able to handle their requests. Some directory services handle this situation automatically via their proprietary protocols, but LDAP does not currently provide a way for a server to inform clients about other replicas of it. One way to achieve load balancing in the absence of this capability is to use the Domain Name System (DNS) round-robin capability to map a given host name to all the IP addresses of hosts containing a copy of the replicated data. With DNS round-robin, the DNS server reorders the list of Internet Protocol (IP) addresses each time it responds to a query for the host name. In this way, client load can be divided among any number of servers. As previously mentioned, it is also possible to use a hardware load-balancing and failover device such as Cisco Systems' LocalDirector or Nortel Networks Alteon Link Optimizer to distribute load across multiple servers.

Using DNS round-robin to distribute clients across a set of replicas has a major drawback, however. If one of the directory servers fails, the DNS server will continue to direct clients to it. If the server will be out of commission for an extended period of time, you will need to remove the server's IP address from the DNS until the server is brought back online. Hardware load balancers typically notice when a server fails, and they stop directing clients to it until the server comes back online.

Other Considerations

In the discussion of reliability and performance, we've focused primarily on where to put replicas. However, we haven't considered some other factors that may be important when you're designing your replication system.

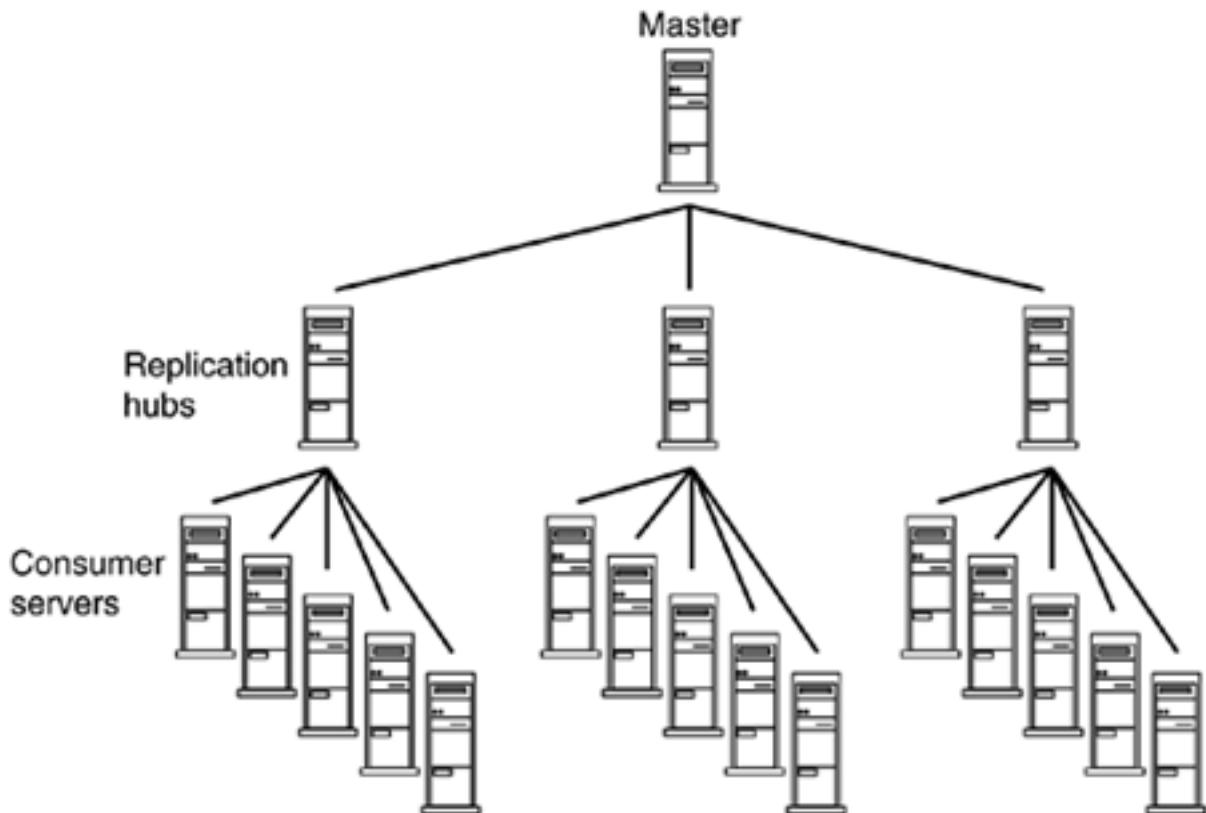
First you need to consider the maximum number of replicas your software can gracefully handle. This number is highly dependent on the software in use and the number of updates that your system receives. In general, it's a good idea to try to limit the number of replicas supplied by a single server to somewhere between five and ten. If you try to manage a larger number of replicas, the servers may spend so much time propagating updates that they are unable to answer client requests in a timely fashion. If your directory sees very few modifications, you can probably use more replicas; if your directory handles many modifications, you may need to use fewer replicas.

What if you find that your directory service is bogged down with synchronization traffic? One option is to partition your directory tree among a larger number of servers. When you do this, each server needs to handle fewer update requests and therefore needs to send fewer updates to replicas. Of course, the total amount of network traffic would still be the same (in fact, it might actually increase somewhat because of the partition management overhead of some directory systems), so if you find that your network is the bottleneck, a network upgrade may be in order. In practice, however, the network is rarely the bottleneck. More often the limiting factors are server CPU, disk I/O, and memory usage.

Another option for reducing the replication burden on a master server is to use a cascaded replication configuration. In a *cascaded configuration*, a change propagates from a supplier

to a small number of consumers, and then from each of those consumers to a larger number of consumers, and so on until all replicas have been updated (see [Figure 11.19](#)). This approach lengthens the time it takes for a given change to propagate to all replicas, but it does make it possible to feed a larger number of replicas. Your directory server software may or may not allow this type of configuration, so consult your documentation.

Figure 11.19. Cascaded Replication



The second factor to consider is the overhead associated with managing a complex replication system. What if a replica goes down? How difficult is it to bring it back up? How do you monitor the system to be sure it's working properly? How difficult is it to find out whether a user's complaint results from a replication problem? In general, use of the KISS principle (keep it simple, stupid) is a wonderful idea: The simpler you can make the replication configuration, the better off you'll be. It'll be easier to troubleshoot, simpler to fix, and probably more reliable overall. If your boss is unimpressed because the system looks too simple, you can describe it as "elegant." That usually works.

Choosing Replication Solutions

If you need to provide a reliable, high-performance directory service, you should evaluate the software you're considering in terms of its replication capabilities. The following are some general questions to ask when performing this evaluation:

- Does the software support incremental updates, in which only the minimum set of changes needed to bring a replica into synchronization is sent?
- Does the software support single-master replication, multimaster replication, or both?
- How do client applications behave when a replica becomes unavailable? Do they automatically select a new server?
- How easy is it to replace a master server that has failed?
- How easy is it to manage replication? Does the software provide tools for determining the state of replication and whether replicas are up-to-date?
- Does the software support the features you need? Does it, for example, allow

replication to be scheduled at a particular time of day?

- Can the software send updates securely (for example, in encrypted form) to prevent network eavesdropping?

More information about selecting appropriate directory server software can be found in [Chapter 13](#), Evaluating Directory Products.

Team LiB

◀ PREVIOUS

NEXT ▶

Replication Design Checklist

To recap, you should perform the following tasks when designing replication for your directory service:

- Create a map of your organization's physical network.
- Inventory your existing and planned directory-enabled applications, and understand the load that they will generate.
- Note on the map the locations of your directory-enabled applications.
- Locate at least one replica within each network location connected via a slower and/or unreliable network link.
- Add additional replicas to handle the client load. Optionally, create dedicated replicas for directory-enabled applications that make extensive use of the directory.

Further Reading

Active Directory Branch Office Guide Series: Planning Guide. Microsoft, Inc., 2001. Available on the World Wide Web at <http://www.microsoft.com/technet>.

IETF LDAP Duplication/Replication/Update (LDUP) Protocols Working Group documents. Available on the World Wide Web at <http://www.ietf.org/html.charters/ldup-charter.html>.

Microsoft Active Directory documents. Available on the World Wide Web at <http://www.microsoft.com>.

NDS eDirectory 8.7 Administration Guide, Novell, Inc., 2000. Available on the World Wide Web at <http://www.novell.com/documentation/lg/edir87/index.html>.

Netscape Directory Server 6 Administrator's Guide. Available on the World Wide Web at <http://enterprise.netscape.com/docs/directory>.

Netscape Directory Server 6 Deployment Guide. Available on the World Wide Web at <http://enterprise.netscape.com/docs/directory>.

Network Time Protocol (Version 3) Specification, Implementation and Analysis (RFC 1305). D. L. Mills, 1992. Available on the World Wide Web at <http://www.ietf.org/rfc/rfc1305.txt>.

Novell's Guide to NetWare 5 Networks. J. Hughes and B. Thomas, Novell Press, 1999.

Looking Ahead

Now that you've used directory replication to provide your users with a reliable and high-performance directory, it's time to think about how to protect the data in your directory service and ensure that persons can view and modify only the data they are authorized to work with. [Chapter 12](#), Privacy and Security Design, will describe directory privacy, security, and access control in detail.

Chapter 12. Privacy and Security Design

- Security Guidelines
- The Purpose of Security
- Security Threats
- Security Tools
- Analyzing Your Security and Privacy Needs
- Designing for Security
- Privacy and Security Design Checklist
- Further Reading
- Looking Ahead

No discussion of directory design would be complete without at least one chapter explaining how to secure the information in your directory and protect the privacy of your users. Without such safeguards, the types of directory applications that can be supported by your directory are severely limited.

If the information in your directory is not secured and you cannot be sure whether it has been tampered with, the applications that use the data are limited to those unconcerned with the accuracy or completeness of the information. If access to the information in your directory is not secured and the information itself cannot be kept private to those authorized to view it, only public information may be stored in the directory. These concerns are paramount to providing an industrial-strength directory service that can be trusted by applications and users.

This chapter describes the purposes of security and outlines the threats posed to it in a typical directory environment. We describe how to analyze your environment to determine your security and privacy needs, and we explain how to design your directory service to meet these needs. A separate section on user privacy explains the importance of keeping data about your users private, and a section on the trade-offs between security and deployability analyzes some common decisions you will have to make.

Security Guidelines

One of the most important points to understand before you begin to design your directory's security infrastructure is that there is no such thing as "secure" or "private" in an absolute sense. Instead, there are degrees of security and privacy that come with various trade-offs and apply only in well-defined contexts.

A good analogy is the security of your house. It probably has one or more doors and windows, each with some kind of lock on it. The security-minded among us lock the doors and windows to our house in an effort to make it secure from unauthorized entry. Clearly, we can achieve only a modest level of security. A window can easily be broken. A lock can be picked. A door can be broken down. Adding bars on the windows and doors increases your level of security, but at the expense of your own convenience. Such trade-offs are typical in the security world and may well be worthwhile if you live in a neighborhood where threats are common. The lengths to which you should go to protect yourself generally should be proportional to the security threats you face—a principle you should consider when designing your directory.

Another important security lesson is that a system is only as secure as its weakest link, so it is important to think of the whole product and protect against every avenue of likely attack. Continuing our analogy to your house, consider the futility of installing a steel-reinforced door with triple dead bolt locks if you're going to leave your windows wide open. Similarly, making your directory system secure in one dimension while leaving other areas wide open leads to a false sense of security. Be sure to consider every aspect of security you can think of that might be related to your service.

On the other hand, these concepts can be taken too far. Why have windows on your house at all? If they can be broken so easily, they provide no real security. Better board them up. Why bother locking your door when anyone who really wants to get in could easily break it down? Better go live in a bank vault. But what good does that do? Even bank vaults get robbed.

The answer is that every effort you make to secure the weakest link improves the overall security of the system. Although no security system is guaranteed to thwart a determined and capable attacker, every additional security measure you employ increases the difficulty of attack. Every time you add a level of security, you filter out more attackers. The more difficult it is to break your security, the more likely it is that an attacker will give up or move on to someone else's house—or directory service.

So how far should you go to protect the security of your directory? The answer depends on the kinds of threats you face—and the consequences you would suffer in case of a security failure. For example, if your directory contains name and e-mail address information, unauthorized access to the directory might result in a lot of junk e-mail being sent to your users—which can be miserably annoying. But the most serious consequences usually are lost time and a waste of system resources.

On the other hand, consider a directory used in a banking site that contains names, account numbers, credit card numbers, and other sensitive financial information. Unauthorized access to this directory might result in far more serious consequences, including improper access to bank accounts, unauthorized use of credit cards, damaged credit histories, and worse. Clearly, this information needs to be protected more strongly.

All these principles are fundamental to security design. Keeping them firmly in mind during your design process will go a long way toward keeping you on track and will help make your service secure and successful. Here is a quick summary of these security and privacy design

principles:

- There are different levels of security and privacy. Your job is to choose the level appropriate for your needs and the threats your directory faces.
- Your system is only as secure as its weakest link. Remember also that the strength of a link in the security chain should be evaluated with respect to the likelihood of an attack.
- Different types of information require different security precautions; similarly, different types of users require different levels of privacy. Don't try to devise a one-size-fits-all solution.

Team LiB

◀ PREVIOUS

NEXT ▶

The Purpose of Security

At its most basic level, the purpose of security is to protect the information in your directory so that you can access it with confidence. The obvious next question is, *Protect it from what?* In the following section, we give an overview of the kinds of threats you should guard against. For now, it's enough to think of these threats as being unauthorized access to or tampering with directory information, or causing users of the directory to be denied service.

If security is breached, often it is important to know exactly what was breached and how. Auditing provides this capability. Auditing also can be useful in determining why the system is not performing as it should, what the directory is being used for, and other interesting and useful bits of information.

Auditing information is invaluable in determining how to secure your system after a break-in. If you don't know what went wrong, it's difficult to know how to fix it. Maintaining an adequate audit trail provides information such as who accessed the server, what operations were performed, when those operations were performed, how long they took, and other information about errors and unusual conditions. Analyzing these logs can give you insight into many problems, including the following:

- **Break-in attempts.** For example, many repeated authentication failures in the logs might alert you to a break-in attempt. This information could help you track down the attacker or take preventive measures.
- **Trawling attempts.** *Trawling* is any technique used to perform unauthorized bulk downloads of directory data. Look for repeated searches that download successive portions of the database in an attempt to defeat the administrative limits you have imposed. This auditing information could help you track down the trawler or take preventive measures.
- **Misconfigured applications.** For example, you might notice an application performing searches that make no sense or aren't optimal, placing unnecessary load on the directory. In extreme cases, by consuming all available directory resources, a misconfigured application can cause others to be denied service. Auditing information can help you identify and fix the misbehaving application or configure your directory to handle the searches better.

There are also nontechnical reasons for securing your directory. It's important for the users of your directory to be confident that the information they consider private is being safeguarded in an adequate manner. Users often have concerns that go well beyond what you may consider a security or privacy threat. For example, you may consider a user's name or gender to be public information, but the user may have legitimate reasons for wanting this information kept private (for example, having a fear of stalking, or being a member of a witness protection program). Such perceived threats are as real as any others as far as your users are concerned, and they should be dealt with accordingly.

Another nontechnical reason to secure your directory is public relations. In some situations this can be the most important reason. A break-in reported in the newspaper or on TV can be devastating to your company's business. The popular press seldom digs deep enough to discover the real consequences of a break-in. If your business is banking or securities trading, or a similar business in which trust plays a vital role, a security breach can be fatal. Your customers (not to mention your competitors) usually won't distinguish between a break-in of your publicly available corporate phone book directory and the bank vault itself. The damage from this kind of a security problem can take a long time to repair.

Security Threats

There are many potential threats to security, and an entire science and industry have grown up around this important area. Several good books on the subject, which we mention in the Further Reading section at the end of this chapter, provide excellent coverage of security in general and treat the subject in a more complete and formal manner than we will here. Because the subject of this book is directories, not security, we will take a more pragmatic and focused approach toward describing the range of security threats. This section provides an example-driven overview of the most typical threats to directory security. We've divided the threats into three categories: unauthorized access, tampering with information, and denial of service.

It's important to understand that an attacker does not necessarily have to be particularly clever to launch one of these types of attacks. With the popularity of the Internet and the growth of the bad-guy community along with it, the advantages of shrink-wrapped software have come to computer security attacks. For most of the threats we describe in this chapter, you can find ready-made software that will exploit them. People trying to compromise your security are often just running shell scripts and programs they downloaded off the Internet. These "script kiddies," as they are known, may have less of an idea how the programs operate than you do! Of course, there are exceptions, too: Wily hackers who discover security holes and write the programs that exploit them are still in abundance.

A commonly held security myth is that most attacks are made by hackers operating out of their basement computing lairs. In reality, most attacks, especially successful ones, are made by your own employees, administrators, and users. In practice, the inside job poses by far the greatest threat to your directory's security in most environments. When designing your security solution, be sure to consider threats both inside and outside your organization.

Unauthorized Access

The threat of unauthorized access may seem simple to protect against. You should authenticate clients accessing your directory and provide access control restricting the information that these clients can view. Problem solved, right? Unfortunately, it's not quite that easy.

Think about the way directory information is delivered to authorized clients. There are several opportunities along this path for an unauthorized client to gain access to the data. Here are several breaches that can occur:

- **Credential forging.** If a client's credentials can be forged, an unauthorized client can fool the directory into thinking it is authorized. For example, suppose that your directory's authentication scheme is based on plaintext passwords, and no other steps are taken to protect the password as it is transmitted to the server. An attacker who is able to watch a legitimate client in an authentication exchange may be able to replay the exchange later, successfully masquerading as the legitimate client. Encryption technologies can reduce this risk by making it difficult for an attacker to obtain the information necessary to mount a replay attack.
- **Credential stealing.** This threat is closely related to credential forging but more low-tech in nature. If your users write down their passwords on notes stuck to their computers, anyone who walks by can steal them. If you use token-based security that requires the user to present a physical token to access the directory, this token can be stolen.

The same is true for schemes based on public key cryptography. Each user has a public key that is shared with others, and a private key that is kept secret. If a user's private key is stolen, the thief can impersonate the user. Credentials can be stolen in a variety of ways—many of them not technology based. Make sure that your users know never to give their password or key to anyone, not even an employee of your own Help Desk.

- **Connection hijacking.** Is it possible for an unauthorized client to hijack an authorized client's connection when the authorized client has authenticated itself? Yes, barring any connection-level protection that prevents it. For this to happen, the hijacker usually has to have access to the same physical network that the victim is on. Methods of attack vary somewhat, but they all involve the hijacker responding to requests meant for the authorized client and preventing the client from responding. General connection protection mechanisms, such as Secure Sockets Layer (SSL) or its successor, Transport Layer Security (TLS), can prevent this. SSL and TLS are discussed in more detail in the next section, Security Tools.
- **Network sniffing.** If it is possible for attackers to eavesdrop on the information exchanged between a legitimate client and the server, they can learn things they are not authorized to know. To guard against this possibility, steps must be taken either to physically protect the network between clients and servers so that no one can listen in, or to protect the information exchanged so that an eavesdropper who does listen in cannot get useful information. SSL and TLS provide these benefits by encrypting the information transferred. Other schemes have this property, but SSL and TLS are by far the most widely used.
- **Trojan horses.** Remember that there can be a lot of software between an authorized directory user (or other agent) and the network that conveys that information. A *Trojan horse* is software that masquerades as a legitimate program but, when run, performs some illicit functions that compromise security. A popular kind of Trojan horse program is a *keystroke sniffer*, which disguises itself as a legitimate login program. This nasty piece of software hooks into the low-level routines taking input from your keyboard, recording all keystrokes you make. These keystrokes can be analyzed later to determine such private information as your password. To protect against Trojan horses, you can use a system integrity verifier, which we discuss in the next section, Security Tools.
- **Backdoor access.** Are there possibly other ways to access the data you want to protect—ways not subject to your directory's authentication and access control safeguards? The answer is almost certainly yes. Directory data lives on one or more server machines, probably residing in some kind of database or file system. If an intruder gains unauthorized access to the directory server machine, he or she has a wide variety of opportunities to access the data. If your directory data comes to your server in a feed from the Human Resources relational database, the information in your directory is only as secure as the source database. Many other avenues of access to your data are possible, in addition to the directory access methods you are designing.
- **Physical access.** Obvious, perhaps, but worth mentioning is the fact that if an attacker has physical access to the directory server machine, he or she can cause a whole host of problems. These range from gaining increased privileges by logging in via a console or other trusted terminal, to just being able to unplug the disk drive containing the directory data and walk out the door with it. Keeping your server machines in locked rooms with limited access is a good guard against this possibility. Encrypting directory data as it lives on the server is another good approach, although this can be expensive. Performance suffers because of the time it takes to encrypt and decrypt data. Extra hardware may be required to make performance acceptable. Also be sure to secure any backup copies of your directory data, including those stored in off-site locations.

- **Software bugs.** This category is a bit of a catchall, representing today's most commonly exploited security problem: bugs. Bugs in the directory server software, the operating system, shared libraries, and even unrelated systems can often be exploited by an attacker to gain unauthorized privileges. Hackers often use prepackaged scripts designed to locate servers that are vulnerable because of a known bug. These scripts often exploit the vulnerability and install a special backdoor access method for the hacker. Systems compromised in this way can be exploited, or used as a launchpad for further attacks mounted from inside your corporate network. To protect against these types of attacks, stay abreast of security patches available from vendors and install them as soon as they become available. You can also reduce your risk by using firewall technology to close off access to all network ports except the ones that need to be accessible via the Internet. For more information on this important subject, consult one of the general texts on security mentioned in the Further Reading section at the end of this chapter.

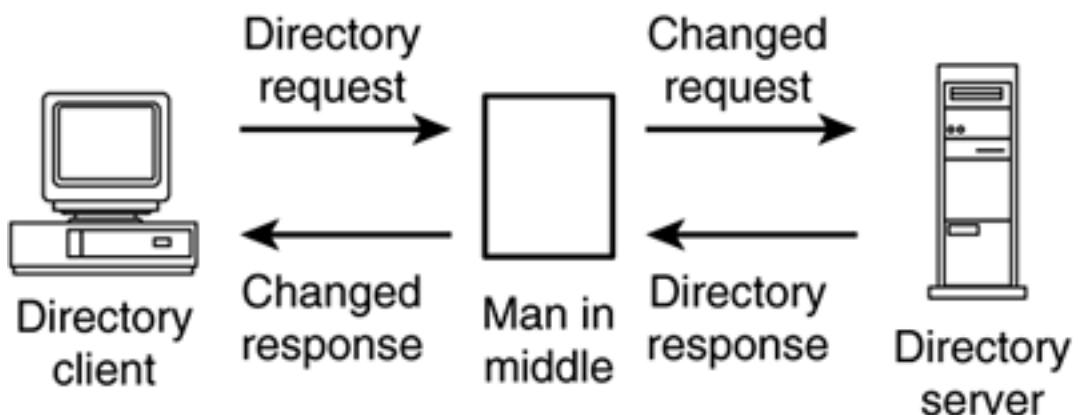
Unauthorized Tampering

Access to directory data is one thing, but if an attacker can actually change directory data—either as it resides in the service itself or en route between client and server—a new set of problems arises. If that were to happen, clients could no longer trust the information they received from the directory, servers could no longer trust the modifications and queries they received from clients, and the directory service would soon become useless.

Many of the attacks described in the previous section could result in data tampering, as could other new attacks. Here are the attack methods you need to be concerned about:

- **Man in the middle.** In one common attack of this kind, an attacker is able to insert himself between the directory client and server. Without any means to detect tampering, the man in the middle could change the client's requests to the server (or not forward them at all) and change the server's responses to the client (see [Figure 12.1](#)). SSL and TLS can solve this problem by signing information at either end of the connection. If the signature is invalid when the data arrives, the data has been tampered with.

Figure 12.1. The "Man in the Middle" Security Attack



- **Trojan horse.** As with unauthorized access to data, a Trojan horse attack on a directory client (or server) can easily facilitate unauthorized tampering. The same countermeasures as for man in the middle apply, as do the same difficulties in applying them.
- **Masquerading.** In the preceding section we described several ways that a client can

fool a server into thinking that the client is someone else. The same problems apply here, for both client and server. A client that can masquerade as somebody else can insert false information into the directory. An evil server that can masquerade as a legitimate server can send back incorrect information to clients and prevent legitimate client modifications from being made.

Denial-of-Service Attacks

Another kind of security threat to your directory does not involve stealing or changing data at all. Instead the attacker's goal is to prevent the directory from providing service to its clients. Such an attack is called a *denial-of-service (DoS)* attack. DoS can be one of the hardest security problems to guard against and detect. There are two main types of DoS attacks:

1. **Direct resource consumption.** This is a general kind of attack in which the attacker simply uses the system's resources to prevent them from being used by someone else. For example, someone could write a directory client that continuously performs expensive directory operations, tying up the resources and making them unavailable to other users. Someone could also write a directory client that stores large amounts of information in the directory in an attempt to exhaust available disk space. There are many other forms of this attack as well.

Placing limits on the number of directory resources that any single client or user can use is a good way to guard against direct resource consumption. Keeping good audit records for the directory is also a good idea. Although an audit record does not prevent the attack, it does allow you to determine when it is happening, and perhaps who is perpetrating the attack and how to stop it. Monitoring can also help fight this problem. If you know the normal level of resource consumption by your directory, monitoring can alert you to any unusual events. Hopefully you will be able to take action before the problem gets out of hand. Monitoring is discussed in more detail in [Chapter 19, Monitoring](#).

2. **Indirect resource consumption.** This attack is similar to the direct attacks just described, but it is often more difficult to detect and guard against. Indirect resource consumption is the use of resources that the directory server or directory clients need, thus denying those resources to the legitimate users. The difference between this and the direct attack is that the directory service itself is not involved, so directory auditing capabilities often don't help in detection and prevention.

For example, an attacker could write a program that uses inordinate amounts of bandwidth. Another attacker with access to the directory machine could write a program to consume CPU, disk bandwidth, memory, or other resources. Much more clever and insidious attacks are possible, too, such as initiating half-opened connections to the directory machine until the machine runs out of system resources. The list of possibilities goes on—and can get nasty.

This kind of attack is difficult to defend against, but you can take some precautions. Isolating your directory machines as much as possible is a good start. Reducing the number of nondirectory processes and users on the directory server machines (or, ideally, eliminating them) is another good idea. Employing firewall filters and other network-level safeguards is also possible. No amount of prevention can eliminate the threat, but it can be reduced.

In addition to traditional DoS attacks, a more sinister variant, the *distributed denial-of-service (DDoS)* attack, has become increasingly common. In a DDoS attack, the perpetrator exploits known security holes to attack vulnerable servers. If the attack is successful, a

special piece of software is installed that can mount a DoS attack and can be remotely controlled by the hacker. When the hacker has tens or hundreds of compromised servers at his disposal, he can instruct all of them to attack a single target system simultaneously.

There are several reasons someone might conduct a DoS attack on your directory. The first and probably most likely is simply by mistake. Bugs in directory client software, misconfigured software, or simply a lack of awareness of the consequences of certain actions can all lead to DoS. Your best guards against this kind of attack are education, monitoring, and auditing.

The second reason someone might conduct a DoS attack on your directory is simple maliciousness. The attacker might be out to ruin your day or the days of your users. The attacker might have a specific problem with you or your service, or your service might simply provide a convenient target for wreaking general havoc. Either way, you would do well to guard against this kind of attacker.

The final and most insidious reason someone might conduct a DoS attack on your directory is to help him compromise another system that depends on the directory. For example, if your Web server depends on the directory to authenticate users, attacking the directory can effectively disable the Web service. This kind of attack can be difficult to defend against because the real motivation behind the attack may never be known.

Security Tools

Now that we've described some of the security threats your directory service may face, it's time to turn our attention to the tools available to help combat them. We do not attempt to provide complete coverage of all the tools out there, but we do give an overview both of the general protection mechanisms and their embodiment in specific technologies.

First, these are the general security methods at your disposal:

1. **Authentication.** *Authentication* is the means by which one party verifies another's identity. Authentication can be one-way or two-way (the latter is sometimes called *mutual authentication*). In *one-way authentication*, a directory client presents a password to a directory server in an LDAP bind operation, or a directory server presents its certificate to a directory client during an SSL connection negotiation so that the connection can be encrypted. In *two-way authentication*, both the directory client and server exchange certificates during SSL connection negotiation.
2. **Signing.** *Signing* is the means by which the authenticity and integrity of information is ensured. If information is signed, the recipient can determine that it was in fact sent by the indicated party and that it was not tampered with in transit. An example of signing occurs when an LDAP connection is made over SSL: The SSL layer divides the stream of data being sent into a series of blocks, and each block is accompanied by a cryptographic checksum that allows the receiver of the packet to determine whether it's been tampered with. In another example of signing, an application stores a signed value within an attribute in a directory entry; the authenticity of the value can then be verified regardless of the security of the server itself.
3. **Encryption.** *Encryption* is the means by which the privacy of information is protected. If information is encrypted, it is scrambled in a way that only the recipient (and possibly the sender) knows how to undo. Encrypted information intercepted by anyone else is not useful. In SSL LDAP sessions, all packets transmitted are encrypted via the method negotiated during connection setup.
4. **Auditing.** *Auditing* is the means by which you track what happens to your directory. Auditing is a key element to the overall security solution because it is often the only way to determine whether your security has been compromised and in what manner. Auditing of data handling and other procedures is also important to make sure that there are end-to-end security protections for your data. The log files maintained by most directory server products are an example of auditing.
5. **Firewalls.** *Firewalls* and other network security technologies are used to prevent unauthorized access to resources on your networks. For example, a firewall can restrict access to your corporate network so that outbound connections from employee workstations are allowed but inbound connections are not. E-commerce sites usually use a sophisticated firewall configuration that creates different zones of network security so that multiple firewalls exist between the public Internet and database servers containing sensitive information. The topic of constructing and maintaining a secure network is beyond the scope of this book, but it is vitally important. For more on the subject, we suggest you refer to one of the texts mentioned in the Further Reading section at the end of this chapter.
6. **Intrusion detection systems.** *Intrusion detection systems (IDSs)* are used to detect when an intruder is attempting to infiltrate your computing systems. In some cases, these tools can detect an attack in progress. In other cases, these tools look for telltale signs that an attack has occurred and report their findings. IDS includes

network intrusion detection systems (NIDSs) that examine network packets for suspicious activity, system integrity verifiers (SIVs) that verify the integrity of critical system resources such as files and registry settings, and log file monitors (LFMs) that check for telltale signs of intrusions in log files generated by network devices and servers. These specific IDS types will be covered in a bit more depth shortly. For more information on IDSs, see the Further Reading section at the end of this chapter.

These six concepts form the basis of most practical security procedures in modern directory systems. There are other, more esoteric security concepts, such as nonrepudiation, that we will not bother to cover in this book.

Now we turn our attention to some specific technologies that provide one or more of the abstract services we just described:

- **SSL.** *SSL* is the *Secure Sockets Layer* protocol. Originally developed by Kipp Hickman of Netscape, SSL is a generic mechanism designed to make connection-oriented protocols like LDAP secure. SSL is based on public key cryptography and can provide high security. It includes strong authentication, signing, and encryption services. SSL lets two communicating parties negotiate a level of security that is appropriate and acceptable to both. A variety of different security algorithms and strengths, or security levels, can be negotiated.
- **TLS.** *TLS* is the *Transport Layer Security* protocol. When the Internet Engineering Task Force (IETF) formed a working group to standardize an SSL-like protocol, it started with SSL version 3.0 and changed the name to TLS. There is little difference between SSL 3.0 and TLS 1.0; in this book we use these terms interchangeably. At the time of this writing, TLS 1.0 is a Proposed Internet Standard, documented in RFC 2246.
- **Kerberos.** *Kerberos* is a security technology originally developed at the Massachusetts Institute of Technology as part of Project Athena. Kerberos provides an authentication service and can be used to provide encryption services as well. Kerberos version 4 was the first widely used version of Kerberos. Later attempts to standardize Kerberos (although not in the IETF) with version 5 have achieved limited success. Various groups have splintered off from the core standard, producing different, incompatible versions, including Microsoft, which used Kerberos-based technology in its Windows 2000 and Active Directory products.
- **SASL.** *SASL*, the *Simple Authentication and Security Layer*, was developed by John Myers of Netscape. SASL (pronounced "sazzle") is a generic framework for negotiating authentication and security layer semantics in application-layer protocols. SASL enables support for authentication, encryption, and signing services. Although it provides no security itself, SASL allows application protocols such as LDAP to negotiate security parameters. LDAP version 3 includes native support for SASL. At the time of this writing, SASL is a Proposed Internet Standard, documented in RFC 2222.
- **IPsec.** *IPsec* stands for *Internet Protocol Security*. A relatively new proposed standard at the time of this writing, IPsec is a generic network-layer security mechanism designed to secure transport-layer connections between machines (such as with TCP, over which LDAP runs). Like SSL, IPsec is based on public key technology, but its focus is on securing connection endpoints, not users or applications.
- **SSH.** *SSH*, the *Secure Shell* protocol, is a generic, end-to-end security package for protecting login sessions, file transfers, and other connections between machines. SSH provides strong security and is relatively easy to use. It is a good general-

purpose tool you can use to secure many of the daily administrative tasks that must be performed on your directory and other systems. SSH version 2 is the most recent version at the time of this writing.

- **SATAN and ISS.** *Security Administrator Tool for Analyzing Networks (SATAN)* and *Internet Security Scanner (ISS)* are generic host security-checking packages. When you point these tools at a host or network, they probe for various well-known security holes that they have been programmed to look for. When finished, they produce a report about all the problems found and how to fix them. Tools like these are invaluable to system administrators (including directory administrators) who need to ensure the security of a host on the network. However, these tools are also invaluable to attackers who are trying to break into those systems. You should beat the attackers to the punch and scan your networks before they do.
- **NIDSs.** *Network intrusion detection systems (NIDSs)* monitor network activity in real time and attempt to determine when an attack is in progress. For example, a NIDS may be able to detect when a DoS attack is in progress and generate an alert. Snort is a widely used open-source NIDS package. See <http://www.snort.org> for more information.
- **SIVs.** *System integrity verifiers (SIVs)* are packages that detect when critical system resources have been tampered with. They accomplish this by taking a snapshot of these resources (system files and Windows registry settings) and periodically comparing the resources against the snapshot. Tripwire is a widely used open-source SIV package. See <http://www.tripwire.org> for more information.
- **LFMs.** *Log file monitors (LFMs)* are tools that examine system logs and perform an action such as paging an administrator when a particular message is logged. Swatch is a widely used LFM package. See <http://oit.ucsb.edu/~eta/swatch> for more information.

The list could go on and on, but we don't have room to mention all the possibilities here. This list should give you a basic idea of the kinds of tools available to you. Check the references in the Further Reading section at the end of this chapter for more on this subject.

Analyzing Your Security and Privacy Needs

Now that you have an idea of the kinds of threats your directory may face and the tools at your disposal to protect against them, it's time to turn your attention to analyzing your environment to determine your specific needs. It's important to tailor your security design to your environment and the needs of your users. Failure to address security needs can compromise your entire service and make an otherwise successful deployment fail. On the other hand, if you design a system that imposes needless and cumbersome security constraints on your users, the service can easily become unpopular, go unused, and fail.

The key to finding the right balance is to understand your other directory design requirements (and how they affect your security design) and the security needs of your environment and your users.

Directory Requirements

Many aspects of the directory design we've discussed up to this point have an impact on security, and vice versa. Some of the more important design decisions that affect security are covered in the following sections.

Read or Write

If you allow users or applications to update the directory, you may require more security. Directory write operations are often held to a higher standard of authentication than directory read operations. For example, you might decide that simple password-based authentication is sufficient to read information from the directory, but to update that same information, a client must authenticate using a public key certificate over a protected SSL connection. With this setup, you can be confident about the integrity of the data in your directory and changes to that data received from clients. You can be less confident that only authorized users are accessing the directory, but you can still maintain some level of assurance.

Sensitivity of Data

Understanding the type and sensitivity of data in your directory is one of the most critical aspects affecting security. If your directory contains publicly readable information, there may be no need for access control. You still may need to be concerned about the integrity of the information, however.

If there are different sensitivity classes of data in your directory, you need to consider how best to protect each one. For example, you may consider name data to be public data and therefore require minimal protection. On the other hand, a directory that contains Social Security numbers or salary information calls for a much greater level of protection.

A useful design exercise is to list all the attributes that will be held in your directory and try to categorize them according to their sensitivity and the type of access and protection required for each one. [Table 12.1](#) shows an example of this kind of categorization. After you choose your directory software, you can use this table to help design your directory access control.

Replication and Synchronization

If your directory is replicated or synchronized with other data sources, you need to be concerned about the security of the data when it is in transit. It's not much good to protect the data if you pass it among servers without similar protection. Also, if your architecture calls for copying data outside your own domain of control (for example, onto every LAN), many copies of the data will exist, not all of which will necessarily be held to the same stringent security requirements you place on your own machines. In other words, it's not much good to use directory access control to protect entries if they will be synchronized to machines controlled by some of the people against which you're protecting the entries!

Think about the network links between replicas. If they are secure, or at least free from access by users, it may be acceptable to pass directory replication updates in the clear. On the other hand, if replication updates must traverse networks accessible to users who may be potential security threats, you'll probably want to protect replication exchanges from eavesdropping, tampering, and other security threats. Replicating over connections secured by SSL might be a good choice in this situation.

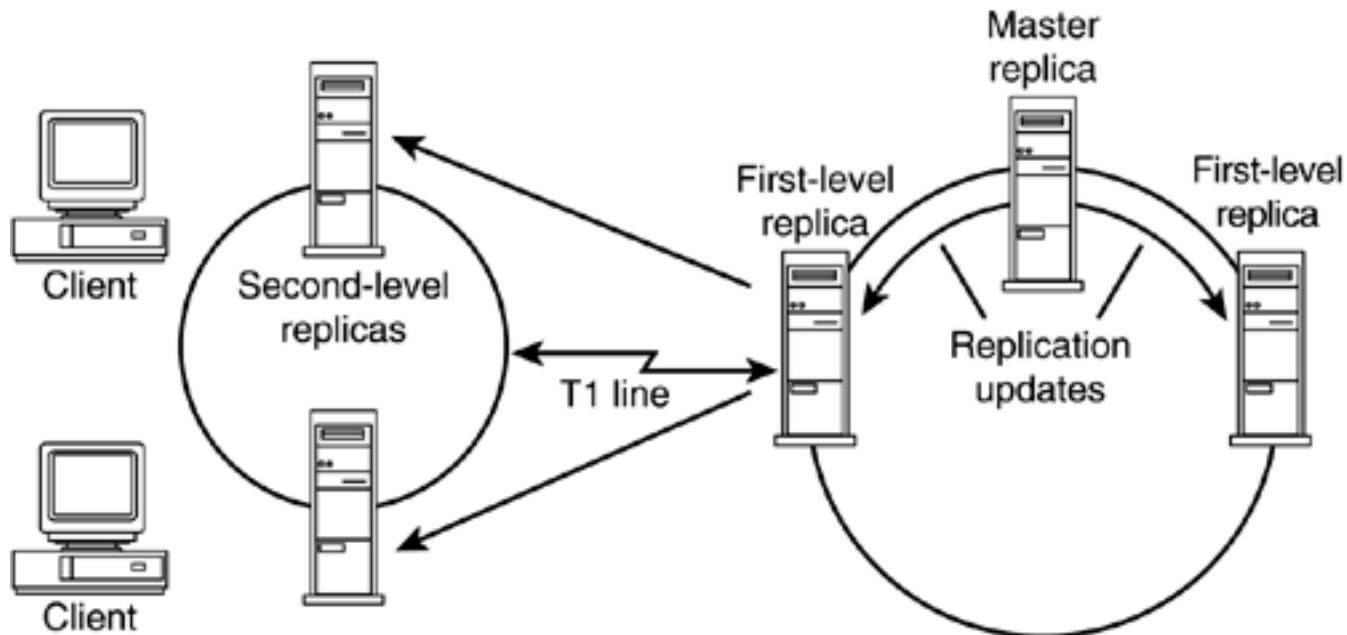
Table 12.1. A Sample Categorization of Attributes by Sensitivity

Attribute	Sensitivity	Accessed by	Updated by
cn	Low	Everyone	Administrator
mail	Medium	Everyone	User, administrator
salary	High	User, administrator	Manager
uid	Medium	Everyone	Administrator

For planning the security of a replication architecture, we've found that network diagrams are helpful. Start by drawing a diagram representing the replicas or other copies of directory data you plan to create and indicate the security level of each network link involved. Then label each copy with its physical and administrative security levels.

[Figure 12.2](#) shows an example of such a diagram. In this case replication updates between the master and first-level replicas may not need protection from eavesdropping because they occur over a network within the corporate data center that is physically inaccessible to users. However, replication updates between the first- and second-level replicas probably do need protection because they occur over user-accessible networks.

Figure 12.2. Replication and Network Topology



Administration

If your directory administrative model calls for delegation, you must be concerned with the privilege level granted to subordinate administrators and their ability to increase their own privilege levels.

You also need to be concerned with other aspects of the administrative environment of your directory. Recognize that you will probably need administrators of the service itself who are different from the administrators of the content held by the service. *Service administrators* start and stop the service and make sure that it runs smoothly, is configured correctly, and is replicated properly. Typically, this function is performed by an experienced system administrator. *Content administrators* are responsible for day-to-day administration of directory content, adding and deleting users, resetting passwords, and similar tasks. Typically, this function is performed by data-processing staff, administrative assistants, and sometimes users themselves. The two types of administration require different kinds of security considerations.

Understanding Your Environment

Analyzing your environment is another important prerequisite to understanding your security requirements. If you understand your environment, you will have a good idea of the kinds of threats your directory will face. On the basis of these threats and the other information you collected previously, you'll be able to choose appropriate safeguards for your directory. At a minimum, consider the user community, directory accessibility, network environment, and physical security.

The User Community

Think about the community of users who will be using your directory. Are they employees of your company who have employment contracts and other incentives not to misbehave and create directory mischief? Are they students with far too much time and cleverness on their hands? Or are they perhaps e-commerce partners who would dearly love to obtain information about each other? Obviously, different user communities require different security measures.

Directory Accessibility

Think about how widely accessible your directory needs to be. Does your directory need to be directly accessible from the Internet? If so, your firewall will need to allow users to connect directly to the server's LDAP or LDAP-over-SSL ports. When you make a server directly accessible to the Internet like this, you need to be vigilant and apply vendor security patches whenever they are made available.

In the Internet case, your directory is available for the world to access. Even if no information in your directory is accessible to users who have not authenticated, there is still a significantly increased risk of exposure to security problems. Attackers have free reign to probe your directory for weaknesses, your directory is more vulnerable to DoS attacks, and any security holes in your directory can be more easily exploited.

On the other hand, if your directory is accessed by only a set of application servers that provide the business logic for an e-commerce application, you can put the directory servers and application servers behind a firewall and prevent direct attacks on your directory servers. This arrangement increases the security of your system and is a common configuration today. However, it is not a panacea, and it effectively shifts the burden of network security to those maintaining the Web servers and application servers.

Your decision about how much security is required depends more on the consequences of a security breach and the attractiveness of your data and organization as a point of attack. If your organization is prominent (especially among the computer-literate), you can count on being a more likely candidate for attack by the mischievous or malicious. Typically, attacks of this kind are motivated by a desire for notoriety (or sometimes revenge, in the case of a disgruntled ex-employee, for example). There is little chance that a security breach will go unnoticed; one of the goals of the attacker is to make his or her breach of your security known.

If the data held in your directory is especially valuable, you can count on being a prime candidate for attack. This kind of attack motive can be more problematic because it is often in the attacker's best interest to conceal the fact that any security breach has taken place. Indeed, attackers in search of credit card numbers, competitive analysis information, or other sensitive information may find the information they steal worthless if the fact that it has been stolen is revealed.

The Network Environment

One of the most vulnerable points in your directory service can be the network over which clients access the directory and over which directory servers communicate with each other to pass replication updates or perform other server-to-server communication. You need to understand these vulnerabilities and their implications for security.

To improve your understanding, create or obtain a map of your network's topology, such as the one shown in [Figure 12.2](#). Make sure you understand all the paths between your servers and between servers and clients. If these paths are susceptible to eavesdropping by unauthorized entities, you must find another way to protect directory data as it travels on the network. One such vulnerable configuration that needs protection is a network in which your server and client machines all share a single Ethernet segment. Any client on this network can easily listen in on traffic not intended for it. A good option might be to require SSL for client/server and server/server communication. With this requirement, eavesdropping does an attacker no good because the traffic is encrypted.

If these paths are not susceptible or at least are not wide open to access, SSL may not be required. An example of this situation might be a configuration in which your server network is physically secure. Perhaps a fiber-optic link (difficult to tap) connects your server network

with your user networks, and your user networks consist of switched Ethernet hubs connecting user machines. In this environment it is much more difficult to eavesdrop on traffic not intended for you. However, it is not impossible, especially if one of the other servers on the server network is compromised and a hostile network sniffer program is installed.

Physical Security

The physical security of your servers is important. As we pointed out previously, securing your network and administrative procedures does not necessarily do you a lot of good if the physical security of your directory server machines is not protected. You can take various steps to ensure a high level of security.

Make sure that your server is kept in a room accessible only to authorized users. In high-security environments, you may want to employ cryptographic smart cards or biometric security procedures to permit entry to the room and use of the server. A retinal scanner is an example of a biometric security device.

Understanding Your Users

A factor often overlooked in security design is the needs of your user community. Understanding how your users will use the directory and view the data it contains, and what their security expectations are, will go a long way toward helping you design a security infrastructure that will make them happy. And their satisfaction, in turn, will go a long way toward making your directory service successful and yourself happy.

How your directory will be used, both by users and applications, can have a significant effect on the security needed by your directory. For example, a directory used for phone book searches and other noncritical tasks certainly has lower security requirements than a directory used to support e-commerce transactions for a high-volume business-to-business Web site.

How your users view the data in your directory is also a relevant factor. A directory that contains personal, but not secret, information about users may not need high security in your point of view, but your users may feel differently. You should be sensitive to these needs. Your users may have a sense of ownership about the data in the directory and strong feelings about how it should be protected from unauthorized access and certainly from unauthorized tampering. They may also feel differently about specific attributes such as home addresses and phone numbers. Be prepared to defend your security policy for each attribute in your directory, or be prepared to have a flexible policy that can be changed by request on a per-user basis. Document your policy and share it with your users.

Designing a flexible access control policy can be helpful. For example, suppose that your directory is accessible both inside and outside your organization and contains both home and work address and phone information. The possible access combinations for these sets of information, for clients inside and outside your organization, are given in [Table 12.2](#). If you were to poll your user community, you would probably find out that there are plenty of users who want each combination.

Table 12.2. Possible Access Combinations for Home and Work Information Inside and Outside an Organization

	Accessible Inside?	Accessible Outside?
Home information	Yes	Yes
	Yes	No
	No	Yes
	No	No
Work information	Yes	Yes
	Yes	No
	No	Yes
	No	No

Some of these combinations may appear strange, but chances are that each one is desired by one or more of your users. For example, it may seem unusual to want home information accessible to outsiders but not to your fellow employees. But one reason might be that publishing a home telephone number to colleagues could encourage them to call you at home—which you might want to discourage; however, you might want friends across the world to have access to your home phone number.

How do you provide a flexible access control system that allows your users to choose the model most appropriate for their needs, yet without creating undue administrative burden for either them or you? Although there is no single answer to this question, it is important to understand the capabilities of your software's access control features and how they deal with situations like this.

Understanding Your Corporate Policies and Applicable Laws

When designing security and privacy policy for your directory, you need to be aware of corporate policies that apply, as well as any relevant state or federal laws.

For example, your corporation may have a written policy that describes who may have access to sensitive employee information. Your directory's access control configuration needs to comply with this policy. In addition, legal requirements may be imposed by state and federal governments concerning the privacy of your directory data.

It's always a good idea to sit down with the relevant departments in your organization, most likely the Human Resources department and your Legal department, and get them to sign off on your security policy. If your organization has a chief security officer (CSO), be sure to involve him or her as well. Maintain a good working relationship with these groups and be

sure to involve them when any unusual events occur. For example, if you become aware of a security breach, or you are approached by a law enforcement official with a search warrant, you should immediately inform all these groups and keep them apprised throughout the resolution of the situation.

Team LiB

◀ PREVIOUS

NEXT ▶

Designing for Security

Up to this point we've covered the basics of security threats, the tools you have in the directory world to combat them, and the effects that your environment and other directory design decisions have on your security design. By now, you should have a good idea of your overall security needs. For example, you should know the level of authentication required, whether you need access control, and whether you need to protect the privacy of client/server or server/server connections. You also should have thought about the relative consequences of various kinds of security breaches.

In short, you should have a good idea what your security policy should be. Now it's time to turn our attention to some more concrete steps you can take to implement your policy.

Authentication

LDAP provides several choices for authentication, which are explained in the following list. Your task is not only to choose the method or methods best suited for your directory, but also to make sure that other unsuitable authentication methods are not used. Recall that authentication is useful only in the context of access control. First you authenticate a client, and then, on the basis of the identity established, you interpret access control information to grant or deny access to certain resources.

Here are the choices for authentication in an LDAP environment:

- **Anonymous authentication.** This term may seem like a bit of an oxymoron. After all, authentication is all about establishing identity, and *anonymous* means no identity. In some circumstances, however, you may want to make data available to any user of the directory, authenticated or not. Users who do not authenticate, or users who authenticate without providing any credentials, are called *anonymous users*. Such users are treated by the LDAP server as if they did not authenticate at all and are given corresponding privileges. You can set access control on information in your directory, permitting it to be read by anonymous users.
- **Simple passwords.** With this option a client authenticates to an LDAP server by sending the server a simple, reusable password. No effort is made to protect the password while in transit. This choice might be appropriate for low-security environments in which the set of users is restricted and relatively trusted (for example, behind a firewall). It may also be appropriate for certain secure networking environments in which eavesdropping is not a concern or in which the information being protected is of little value.

The advantage of passwords is their simplicity, of course. Most people know and understand passwords, how to use them, and the administrative costs associated with them. Password security is also simple to implement, so there are few, if any, performance implications inherent in choosing a password-based solution.

- **Digest MD5 authentication via SASL.** With this option a client authenticates to an LDAP server without sending the password over the network in the clear. Instead, the server constructs a challenge that is sent to the client. The client must respond to this challenge in a way that proves the client knows the password. Even if an attacker is able to observe the authentication exchange, the information learned is of no use. Digest MD5 authentication offers the ease of simple passwords while preventing theft of credentials or replay attacks.
- **Simple passwords over SSL.** With this option, SSL (or TLS) is used to encrypt the

connection over which a simple password is transmitted. The simplicity and convenience of passwords can be preserved, but the added security of SSL means that this option can be used in relatively high-security environments in which networks are open to eavesdropping attacks and the information protected is of relatively high value.

There are several downsides to this choice. SSL is more complex than a password-based solution. You must obtain certificates for your servers (but not for your clients) and renew them when necessary. Both clients and servers have to include SSL implementations, which involve a substantial amount of code, increasing the cost and size of software. Not all clients and servers support SSL, which means that choosing this solution restricts your choice of software. SSL may not be much of a problem on the server side; after all, you are in charge of this choice, and if you can live with it there should be no problem. The corresponding client choice may be more problematic because users may want to use a variety of clients that do not support SSL to access your service. Think about whether supporting such clients is important for your directory. Also think about the kinds of operations for which you want to require SSL.

One more important consideration is the impact your choice will have on the performance of your system. Using SSL to authenticate a directory session consumes a substantial amount of extra computing power; encryption also requires extra cycles. Whereas the client side of this burden is distributed across the many client machines on your network, the server side is more centralized and therefore more of a concern. At the time of this writing, an LDAPv3 extension named StartTLS is nearing completion that will allow encryption and other TLS features to be turned on and off during a session. This feature will allow your password to be protected during authentication before an update operation without degrading the performance of the rest of the system. It is likely that not all servers and clients will support this extension, however, so check with your vendor.

Be sure to perform benchmarks during the piloting phase of your service to ensure that performance is not degraded too much. If it is, consider adding hardware acceleration to the server. A cryptographic hardware accelerator is usually a special board or other device you can add to your computer. It has processing hardware that performs cryptographic operations much faster than your computer's general-purpose processor can. Keep in mind that not all LDAP servers support hardware acceleration (Netscape Directory Server 6 is one that does support it).

- **Certificate authentication over SSL.** With this option you are using the full power of SSL. This power allows you to do three things:
 1. Protect the privacy of data
 2. Ensure the integrity of data
 3. Authenticate clients

Government Restrictions on Encryption Technology

A separate but important consideration is the effect, if any, that U.S. or other national laws may place on the strength of your security solution. The U.S. government considers encryption technology to be like a weapon potentially useful to foreign countries, international terrorists, and other nasty people intent on wreaking havoc. Restrictions on the use and export of cryptographic systems vary widely from country to country and are likely to be in a state of flux following the terrorist acts perpetrated on the United States on September 11, 2001. Before using any cryptographic system, be sure you understand the laws that apply to you. The remainder of this sidebar focuses on U.S. encryption policy as it was before September 2001.

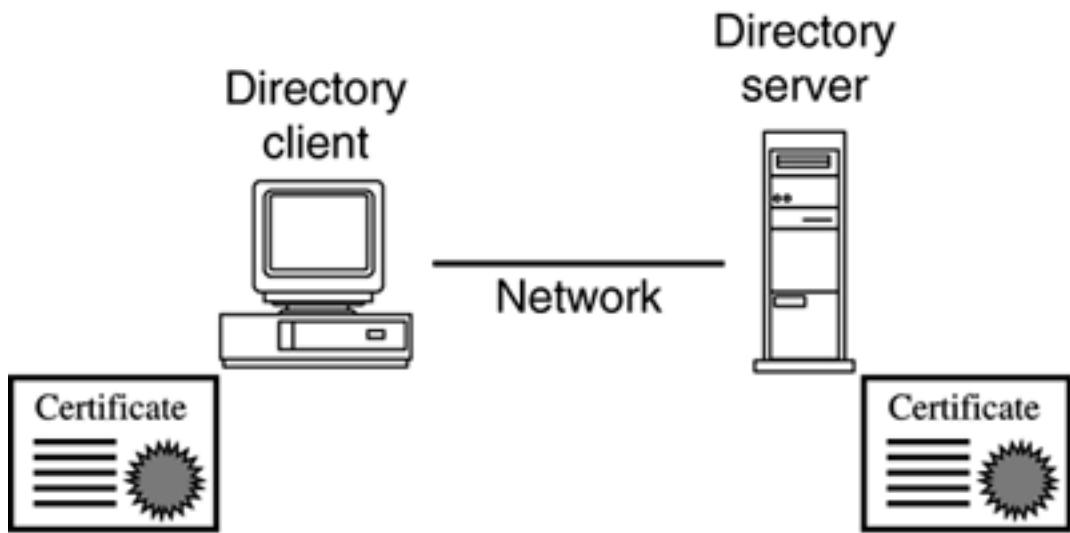
The United States restricts the export of strong encryption technology (you can interpret "strong" as "hard to crack"). Products with strong encryption capabilities can be exported only to selected countries in the European Union and several other industrial democracies. Export to other countries is possible only after technical review by the federal government.

In today's world, the preferred key length for very secure encryption is 128 bits, which is classified as strong encryption by the U.S. government. However, with processors getting faster all the time and, more importantly, with people getting more clever in their methods of attack, the bar is continually being raised. For example, recent well-publicized cracking "challenges" (in which Internet users are challenged to decrypt a secret message as a test of security) have been met with many clever people harnessing the power of thousands of computers on the network to work in parallel on cracking the code.

What does this mean to you? Well, if your country is not on the approved list, it may mean that you cannot get very secure encryption technology from a U.S. vendor. Your choices are to live with less secure technology, pursue a more secure solution with a non-U.S. vendor (not bound by U.S. export laws), or wait for the U.S. government to adopt a more liberal encryption export policy.

You end up with a much stronger level of assurance as to the identity of the client based on public key cryptographic algorithms. [Figure 12.3](#) shows the components of an SSL-based system supporting certificate-based client authentication. The downside is that instead of distributing the plain old passwords that we all know and understand, your clients must generate private-public key pairs and be issued certificates. Those certificates and keys must be protected and managed throughout their life cycle. You must deploy and maintain a public key infrastructure (PKI) to provide this life cycle management. Historically, certificates have not been sufficiently easy to use or manage.

Figure 12.3. Components of a PKI-Based SSL System



Another disadvantage of this approach is the generally higher processing cost of public key security. As discussed earlier, more computing power is required than with other approaches. You may need hardware acceleration to make a public key solution feasible for a system with any kind of scale. The upside, of course, is that you get a high level of security.

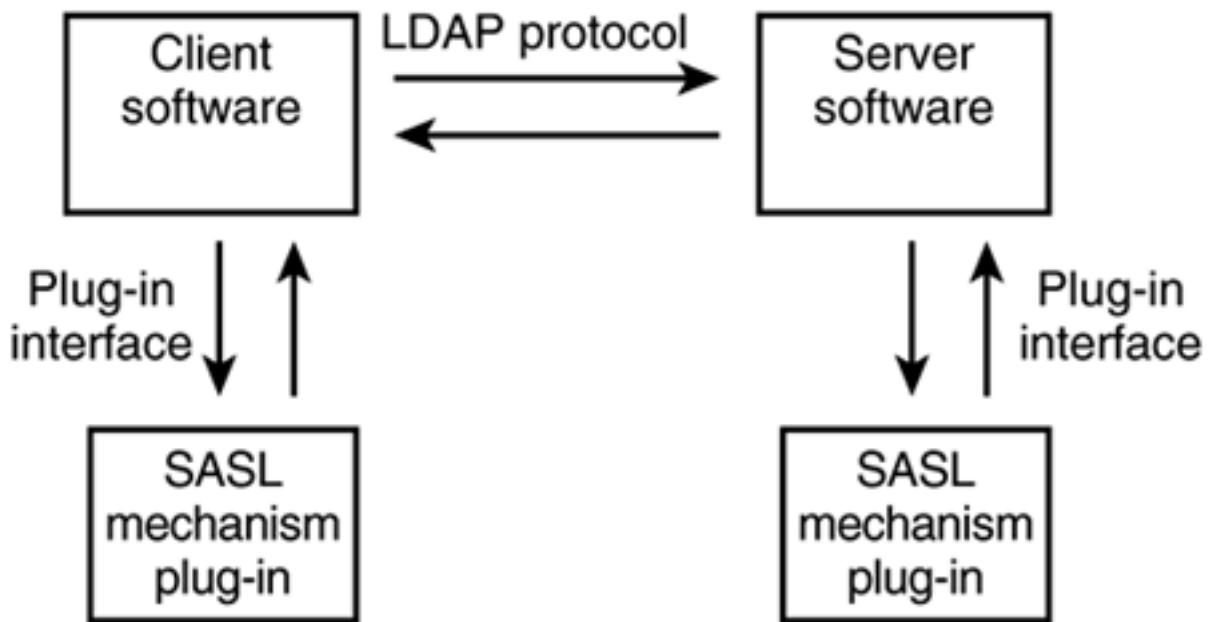
If you decide you need this level of security, be sure the software you choose supports it and that the support is flexible enough to serve your other needs. For example, what kind of solution does the software provide for certificate life cycle management? During authentication, how is the association between a subject in a certificate and an entry in the directory made? You probably want this mapping to be as flexible as possible to prevent too tight a link between your certificate authority and your directory service.

- **Another scheme via SASL.** This is really a whole set of options because SASL is a generic mechanism for hooking in just about any authentication and security mechanism you can think of. You might want to use this mechanism if, for example, you have an existing Kerberos authentication database that you want to leverage. By obtaining or writing SASL modules for Kerberos on your server and all clients that will be accessing the directory, you can continue to use your current authentication database, administration procedures, and so on.

Realize that by choosing this option you are probably putting your fate in your own hands, unless the vendor you choose supports a SASL module for your particular authentication scheme. At the time of this writing, Netscape Directory Server 6 supports a SASL interface that allows you to write your own plug-in modules, but Netscape does not provide any prewritten modules. The Netscape server also does not yet support security-layer plug-ins, only authentication plug-ins.

Another concern with writing your own SASL plug-in is that each client that accesses the directory must have a similar plug-in that understands the security mechanism you've chosen. Also you should check with your server vendor to see how well a new SASL mechanism you define can be integrated with the directory's access control scheme. As with Netscape Directory Server 6, you can often explicitly reference the type of authentication used when creating an access control list, but only for predefined authentication mechanisms. The pieces fit together as shown in [Figure 12.4](#).

Figure 12.4. Components of a SASL-Based Security Scheme



Access Control

When you've decided on one or more authentication schemes to establish the identity of clients, you need to decide how to use them to protect information contained in your directory. This protection is usually provided in the form of access control, allowing you to specify that certain identities have access to certain information, whereas certain other identities or groups of identities do not.

Overview of Access Control Models

As we discussed in [Chapter 2](#), Introduction to LDAP, there is no standard LDAP access control model today. The IETF is working on defining one, but it will probably take a while for the design to be finished, and even longer for the standard to be implemented in products and become available for you to use. In the meantime, you have to examine each directory server vendor's access control capabilities independently in the context of your needs.

Even though there is no standard access control model, most vendors' implementations of access control have these common characteristics:

- Access can be controlled down to the attribute level. For example, it's possible to allow read access to one attribute type and write access to a different attribute type.
- Access control rules can be specified for an entire subtree at once. Generally, an access control rule included in an entry applies to that entry and all entries subordinate to it.
- Subjects may be individuals (named by distinguished name, or DN) or groups (named by the group's DN). Thus it is possible to specify access control policy for a group in your organization rather than for each individual. This capability reduces the number of access control policies you must manage, increasing security and decreasing management time.

Access control is usually specified as one or more access control lists (ACLs). Each ACL specifies three things:

1. **One or more resources in the directory.** Resources, also called *objects*, are the things to which you control access. They are typically entries, attributes, or values. An example might be the `cn` attribute in all the people entries in your directory.

2. **One or more clients (users or applications) accessing the resources.** Clients, also called *subjects*, are the entities to which you grant or deny access. A client may be specified as the name of a directory entry or other descriptive information. For example, a client might be specified as any authenticated directory user with a connection originating within your organization's Domain Name System (DNS) domain.

3. **One or more access rights.** Access rights determine what the subject can do to or with the object. For example, an access right might be "search read," meaning that the given clients can search for and read attributes from the entry.

Implementing an Access Control Policy

As we advised earlier in this chapter, the first step in determining your access control needs is to understand your data and the users who will access it. Make a table that shows who can access each data element or attribute, the level of access that should be given, and any other relevant restrictions, such as the authentication level required, time of day the access is to be allowed, and whether the attribute can be retrieved only over an encrypted connection. Try to group attributes that have related access characteristics.

[Table 12.3](#) shows an example of such a table, with attributes and access control information for a directory intended to contain some white pages data. These attributes apply to people entries. Keep in mind that you should make similar tables for each kind of entry in your directory. If your directory stores entries for people, groups, printers, and projects, you should construct four tables.

Table 12.3. A Sample Data Access Policy

Attribute(s)	User(s) or Application(s)	Access Level	Authentication and/or Connection Protection Required
cn, sn, givenName, middleInitial, name	All	Read	None
cn, sn, givenName, middleInitial, name	Administrator	Read/write	Password- or certificate-based over SSL
mail	All authenticated	Read	Password
mail	Administrator	Read/write	Password- or certificate-based over SSL
mail	Self	Read/write	Password- or certificate-based over SSL

<code>homeAddress, homePhone</code>	All	Read or none, depending on user's choice	Password
<code>homeAddress, homePhone</code>	Self	Read/write	Password
<code>postalAddress, telephoneNumber</code>	All	Read	Password
<code>postalAddress, telephoneNumber</code>	Administrator	Read/write	Password- or certificate-based over SSL
<code>salary</code>	Self	Read	Password
<code>salary</code>	User's manager	Read/write	Password- or certificate-based over SSL

We will not discuss the design of ACLs for the entire table, but we will do so for three interesting parts of it using the ACL syntax understood by Netscape Directory Server 6. The Netscape approach to representing ACLs is typical, although with some interesting capabilities not found in other approaches. An ACL is represented by an attribute called `aci`, which stands for *access control instruction (ACI)*. An ACI can be placed anywhere in the directory tree. Depending on the ACI, it can apply to the entry in which it is placed and any of that entry's descendants.

ACL Example 1

The first access control rule we will codify is the one for the various name attributes (`cn`, `sn`, and so on). The intention of this generic rule is to make the naming attributes readable by anyone, authenticated or not. The attributes are to be updated only by the administrator. With LDAP Data Interchange Format (LDIF) to represent the Netscape syntax for ACLs, the following `aci` attribute does the job:

```

aci: (target ="ldap:///dc=example, dc=com")
(targetattr="cn||sn||givenName||middleinitial||name")
(version 3.0;acl "Anonymous read-search access";
allow (read, search, compare)
userdn = "ldap:///anyone";)

aci: (target ="ldap:///dc=example, dc=com")
(targetattr="cn||sn||givenName||middleinitial||name")
(version 3.0;acl "Admin write access";

```

```
allow (write)

userdn = "ldap:///cn=directory manager";
```

We are assuming that this ACI is placed in a directory subtree rooted at the entry `dc=example,dc=com`. It applies to that entry and every entry below it in the tree, and only those attributes listed in the `targetattr` line of the ACI. The first value of `aci` grants read, search, and compare access rights to anyone accessing the directory, whether authenticated or not. The second value grants write privileges to the directory manager. If you were going to use this ACI in your directory, you would change the `dc=example, dc=com` part to point to the base of your tree. You would also change the `cn=directory manager` part to reference your directory manager's entry.

ACL Example 2

The next access control rule we will examine is for home address and phone information. The interesting thing about this rule is that users have a choice of which access applies to their information. You might want to do this because some of your users won't want their home address and phone information published. One approach would be to allow users to write their own access control lists. This approach is fraught with peril—from users accidentally denying themselves access to things they should be able to access, to users accidentally granting access to things they should not, to various kinds of malicious behavior. Allowing users to modify their own ACLs is almost never acceptable.

A way around this problem is to make ACL changes go through a mediator. The mediator might be a Web application that you develop specifically for the purpose of allowing users to make changes to the access control information in their entry. The mediator has access to make changes but also has the knowledge of what should and should not be allowed. This approach may be acceptable to you, and it is supported by most ACL schemes. The downside is that users have to go to a special application (that you must maintain) to make ACL changes.

Another approach, supported by Netscape Directory Server 6, solves this problem in a more elegant way. The Netscape server allows you to predefine access rules that apply only to entries satisfying some arbitrary criteria expressed as an LDAP search filter. These entries can live anywhere in the directory. This arrangement allows you to specify the access control rules in a single location, and then allow your users to update a different attribute in their entries, causing a new set of access control rules to apply. For example, consider the following pair of `aci` attributes, shown in LDIF format:

```
aci: (target="ldap:///dc=example, dc=com"
      (targetfilter="(homeaccess=private)")
      (targetattr = "homeAddress|homePhone")
      (version 3.0; acl "keeps users' home information private";
       allow (none)
       userdn = "ldap:///anyone";)
      aci: (target="ldap:///dc=example, dc=com")
```

```

(targetfilter="(homeaccess=public)")

(targetattr = "homeAddress|homePhone")

(version 3.0; acl "publishes users' home information";

allow (read)

userdn = "ldap:///anyone";)

```

Notice the additional `targetfilter` construct in the second line of each ACI. This is a search filter used to select the entries to which the ACI should apply. In the case of the first ACI, we've chosen a filter that selects entries with the hypothetical `homeaccess` attribute set to the value `private`. Entries that satisfy this filter will have their home address and phone information attributes protected by this ACL.

The second ACL contains a similar filter, but this one tests for entries with the `homeaccess` attribute set to the value `public`. In this case the attributes can be read by anyone. Strictly speaking, the first `aci` value is not needed. By default, no access is granted unless specifically granted in an ACI. In this case we assumed there could be other, more general ACIs granting access to attributes by default.

With this access control scheme, a user has only to change the content of an attribute in her entry from `public` to `private` to enable a new access control policy. This kind of ACL manipulation—essentially flipping a switch in an entry—makes maintaining multiple access control policies convenient. This capability is powerful; it allows a compact specification of complex access control policies to be made independently of directory tree structure. Use of this feature inevitably results in access control rules that are easier to understand. You can use this feature for many other policies, some examples of which are outlined here:

- Apply different access control to various types of entries by using a filter to test for different values of the `objectclass` attribute. For example, you could have one set of ACIs that applies only to entries satisfying an `(objectclass=person)` filter. Another set of rules could apply to `(objectclass=group)` entries.
- Apply policies based on attribute values entered by your users. For example, you could apply an access control to all entries with mail addresses not ending in the DN of your organization by using a filter such as `(!(mail=*>@your.domain.com))`. Thus you could automatically prevent the routing of mail to entries that leave your domain.
- Apply policies based on reporting structures. For example, you could have an ACI that applies only to entries with `(manager=<supervisor's DN>)`.
- Apply policies that quickly and easily grant and revoke access to users on the basis of the content of their entries. For example, you could effectively rotate a particular role among users simply by changing a value in their entry. Any entry satisfying `(role=boss-for-a-day)` could be given special privileges.

As you can see, the applications of the target filter access control feature are many, and the power and flexibility are great.

ACL Example 3

The final ACL example we will investigate in detail is represented by the `salary` rows in [Table 12.3](#). Our goal is to allow only an employee's manager the right to update the employee's salary. The employee should have only read access to the salary, and no one

else should have any access. The latter two conditions are not much different from the simple naming attribute examples we discussed earlier. The first goal, allowing only the manager to update something, is more interesting, and we will focus our attention on that.

You could easily come up with an ACI to satisfy this goal, assuming that you know the DN of the manager in question. The problem is one of management. Taking this approach, you would need to create a separate ACI for each manager in the company, and you would need a way to apply that ACI to only those entries managed by that manager. The target filter feature mentioned previously is helpful here, but it does not help solve a basic management problem: If a user changes managers, or if a manager changes roles, you need to update the corresponding ACIs. You could use ACIs that refer to group entries for this, but then you would have to update these groups all the time.

It is also a problem that the number of ACIs required by this approach is proportional to the number of managers in your company—potentially a large list! More ACIs means more maintenance, more chance for error, and more things to change in the event of reconfiguration or reorganization. The existence of many ACIs can also adversely affect the performance of your directory. Because all these ACIs are essentially identical, except for the value listed in one field, there must be a better way.

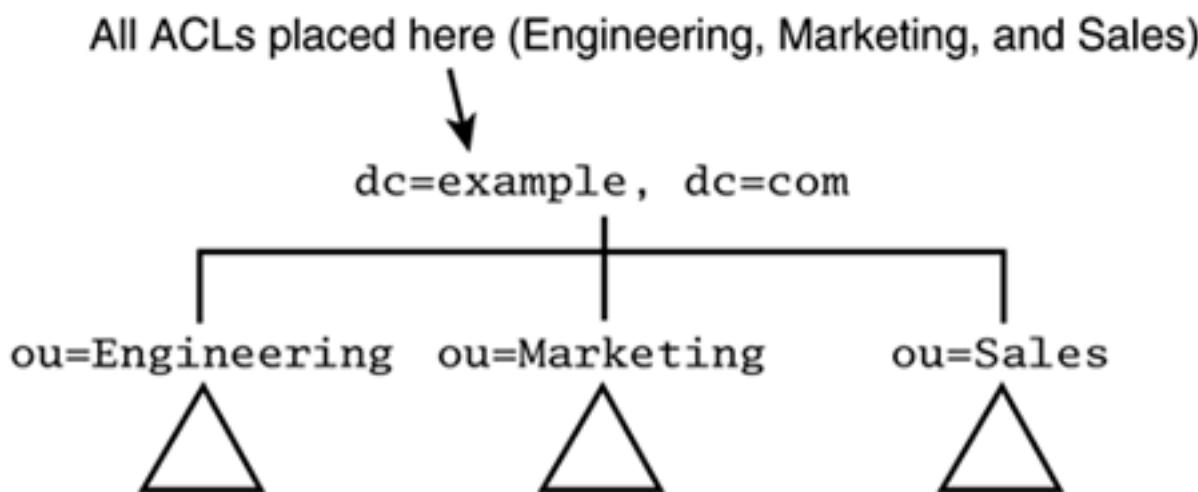
Fortunately, there is. Netscape Directory Server 6 implements a powerful feature aimed specifically at reducing this management overhead and the number of ACIs you need to store in your directory. Consider the following `aci` attribute:

```
aci: (target="ldap:///dc=example, dc=com ")
      (targetattr = "salary")
      (version 3.0; acl "allow manager access to salary";
       allow (read, write)
       userdnattr = "manager";)
```

The interesting part of this ACI is the `userdnattr` construct in the last line. This means that the access is granted to the entry or entries listed in the `manager` attribute of the entry to which the ACI applies. So, for example, if the ACI applies to the entry shown in [Figure 12.5](#), the `salary` attribute in the entry is readable and writable by the entry mentioned in the `manager` field—namely, `uid=managerjoe,dc=example,dc=com`:

```
dn: uid=workerbabs, dc=example, dc=com
uid: workerbabs
cn: Worker Babs
sn: babs
manager: uid=managerjoe, dc=example, dc=com
salary: 9000
... other attributes ...
```

Figure 12.5. ACL Placement in a Directory That Allows Separation of ACLs and Namespace



The power of this capability should be clear now. It can reduce the number of ACIs you need to maintain in your directory and reduce management costs by avoiding the requirement to maintain duplicate information. Furthermore, it can transfer ACL decisions to directory users in a safe, efficient, and error-free way, freeing up directory administrators for more important tasks. Consider these additional applications of the `userdnattr` ACL construct:

- Maintain an `owner` attribute for groups. A single ACI can be used to allow the users listed in the owner attribute the ability to update the group. This approach avoids the need for a separate ACI for each group that must be changed each time the owner of a group changes.
- Maintain a `proxy` attribute for entries. A single ACI can be used to allow the users listed in the `proxy` attribute the ability to update the entry. Such a capability might be used, for example, to give an administrative assistant temporary access to an entry that the assistant will update on behalf of his boss. Again, this functionality is provided while the number of ACLs involved and the administrative effort required are minimized.

No doubt, you can think of other interesting applications for this feature as well.

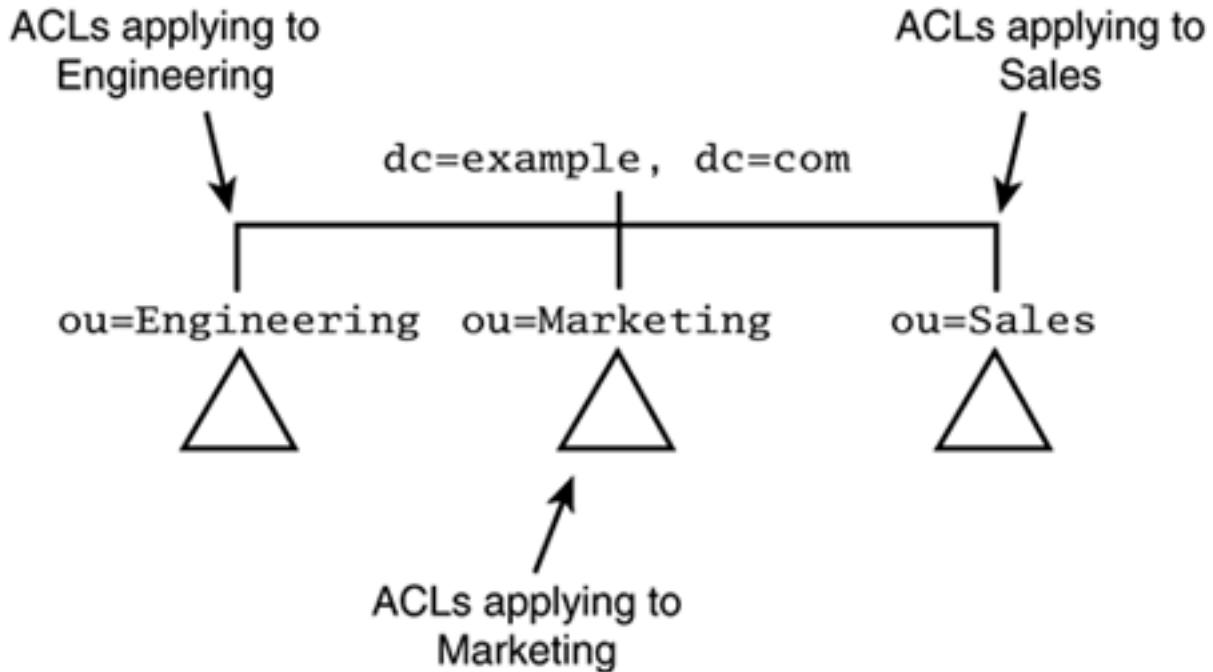
As with any powerful scheme, this one comes with some risks. Just as you can easily change much access control information in your directory for the better, you can also change it for the worse. For example, if you were not careful, you could remove access to public attributes with a simple change. More seriously, you could grant access to information that should remain private. Always be careful and triple-check your work when modifying access control information.

ACL Placement

Finally, you must consider where in the directory to place the ACLs you have designed. This placement depends on the design approach you take and the capabilities of your software. These two things are related. If your design takes advantage of some of the features we've been discussing, and one of your goals is to limit the number of ACLs in your directory, you need directory software that supports this. If, on the other hand, your design approach is to put an ACL in each entry, your choice of software is probably expanded. You should be sure to determine the scale and performance implications of this approach. For example, some directory server implementations do not handle large numbers of ACLs well.

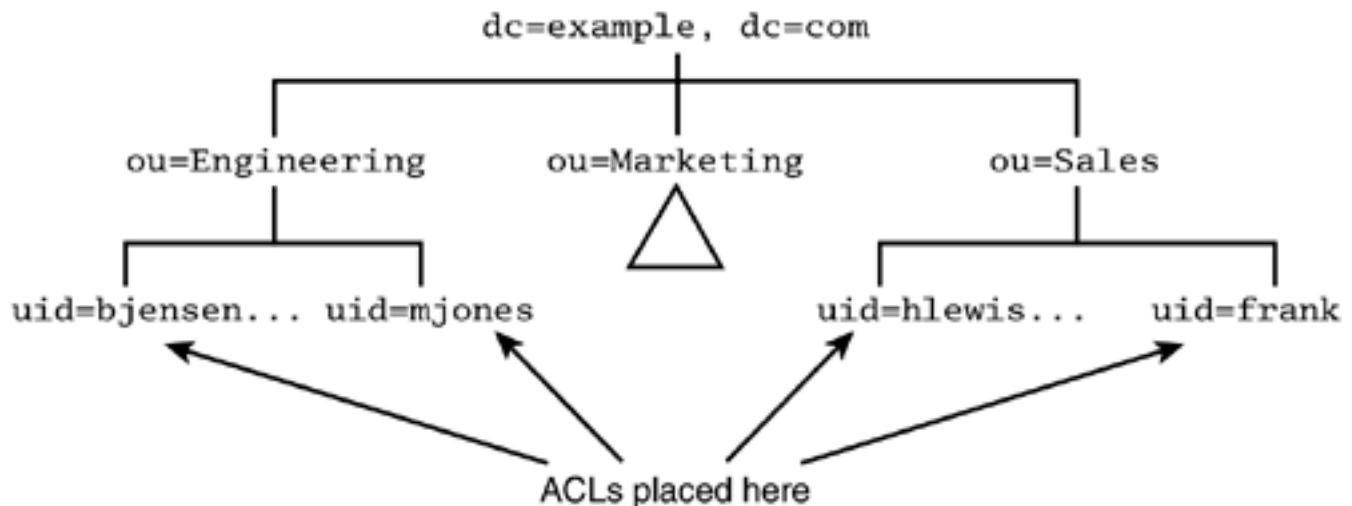
Where you place ACLs in a tree also depends on the organization of your namespace and the capability of your directory software to separate ACL placement from the namespace. If you use software such as Netscape Directory Server 6, you can separate ACL placement from namespace; your ACL placement might look something like [Figure 12.5](#). If you use software that does not allow such placement, you might end up with an arrangement like that in [Figure 12.6](#).

Figure 12.6. ACL Placement in a Directory That Does Not Allow Separation of ACLs and Namespace



If you place an ACL in each entry, your tree might look something like the one depicted in [Figure 12.7](#). We discourage you from taking this approach. In our experience, the number of ACLs in your tree is directly proportional to the administrative burden to maintain them. When things go wrong, it's much more difficult to find the offending ACL when there are many ACLs. Overall, fewer ACLs are better, and anything you can do to reduce the number of ACLs in your tree will make your life easier.

Figure 12.7. ACLs Placed in Each Entry



A final consideration is the relationship between ACL placement and replication. If you place ACLs at some point in the tree but replicate only a subtree below the entry containing the ACLs, some directory server software, such as Netscape Directory Server 6, will not replicate the ACLs (because they are not within the subtree selected for replication). If this is the case, you can either place the ACL within the replicated subtree so that it is replicated along with the data, or you can create an ACL on the consumer server in the same entry that contains the ACL on the supplier server.

Information Privacy and Integrity

You have designed your authentication scheme for identifying users and your access control scheme for protecting information in your directory. Now it's time to design a way to protect the privacy and integrity of the information as it is passed among servers and between clients and servers. Recall that earlier in this chapter we discussed the various threats to data privacy and integrity and how the threats you face and other aspects of your environment affect your design choices. In this section we detail these effects and lay out your design options in this area.

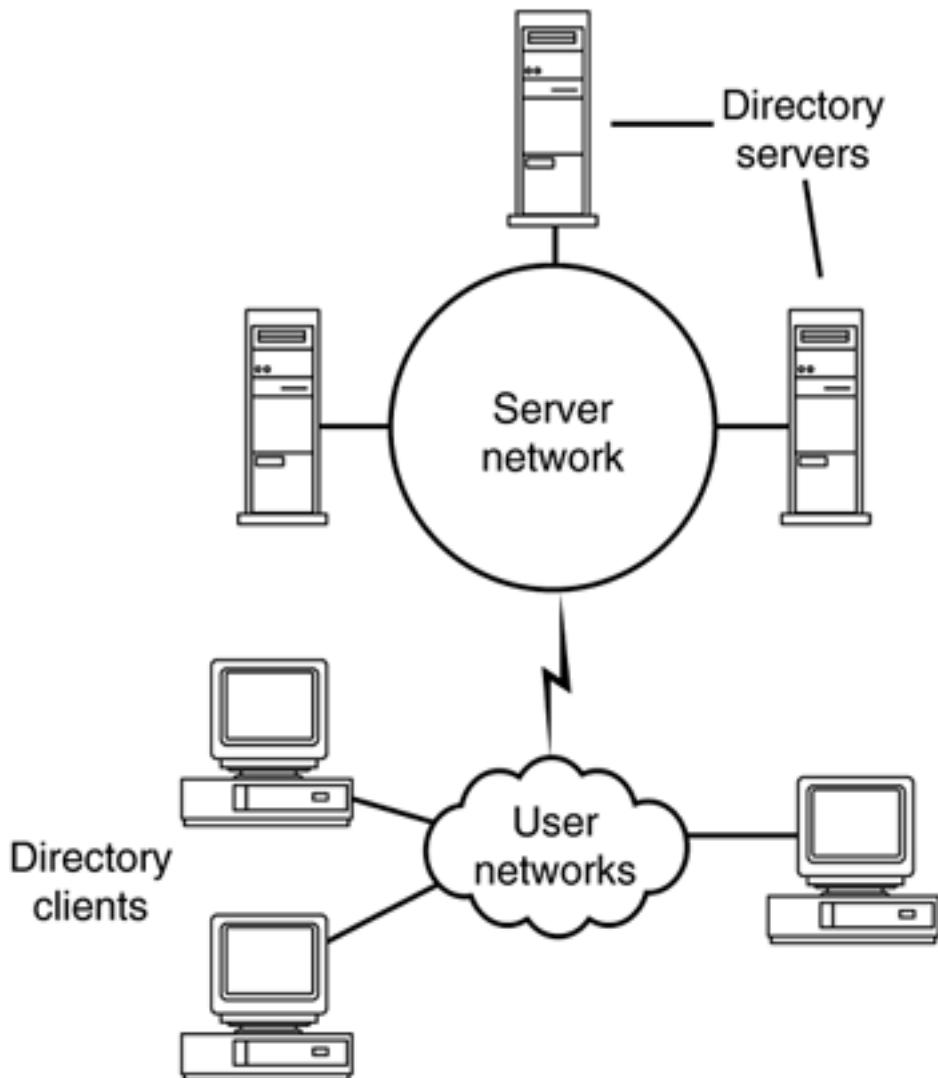
If your environment is such that you need to ensure the privacy and integrity of connections at a directory software level, your only real choice is SSL. Not every vendor supports SSL, and not every vendor that supports SSL for client/server connections supports it for server/server replication connections. Be sure to check the capabilities of your software, and make SSL support part of your evaluation criteria if this kind of protection is important to you.

When deciding whether you need this kind of protection, think about the different kinds of information you'll be keeping in your directory. Think about the consequences of each piece of information having its privacy or integrity compromised. You will find different answers for different attributes, of course. Consider making a table, like [Table 12.4](#), listing categories of attributes in your directory and how their privacy and integrity need to be protected. In practice, you might want to combine this table with [Table 12.3](#). We've separated them here for clarity in this discussion.

After you create a table like this, you also need to decide whether to protect privacy and integrity for client/server access, for server/server access such as replication, or for both. Protecting client/server access is usually a higher priority than protecting server/server access.

When making the decision whether to protect server/server connections, consider the topology of your replication solution. If you use replication in a tightly controlled manner, all within your authority to protect, you may not need to protect the privacy of replication connections. For example, this kind of environment might be found in a centrally administered information services (IS) department with good network connectivity throughout the organization. The directory might be replicated among three servers, all located within close proximity and on networks to which no users have access (see [Figure 12.8](#)). In this environment there is less risk that anyone will eavesdrop on a replication exchange, so SSL may not be required. On the other hand, you might want to take a more secure approach and use SSL everywhere by default, unless there is a compelling reason not to.

Figure 12.8. A Replication Configuration That Does Not Require Privacy Protection



In environments without this kind of control, or with networks that are not protected from eavesdropping, SSL may be an important component of the replication solution. For example, consider an organization without reliable network connectivity. Such a situation could require wide directory replication onto each LAN so that the directory is always available even if the WAN link goes down. Consider also an application that has high performance requirements. This case could require directory replicas to be widely dispersed onto each LAN to reduce latency of directory queries (see [Figure 12.9](#)). In this environment, protecting replication connections may be as important as protecting client/server connections.

Figure 12.9. A Replication Configuration That Requires Privacy Protection

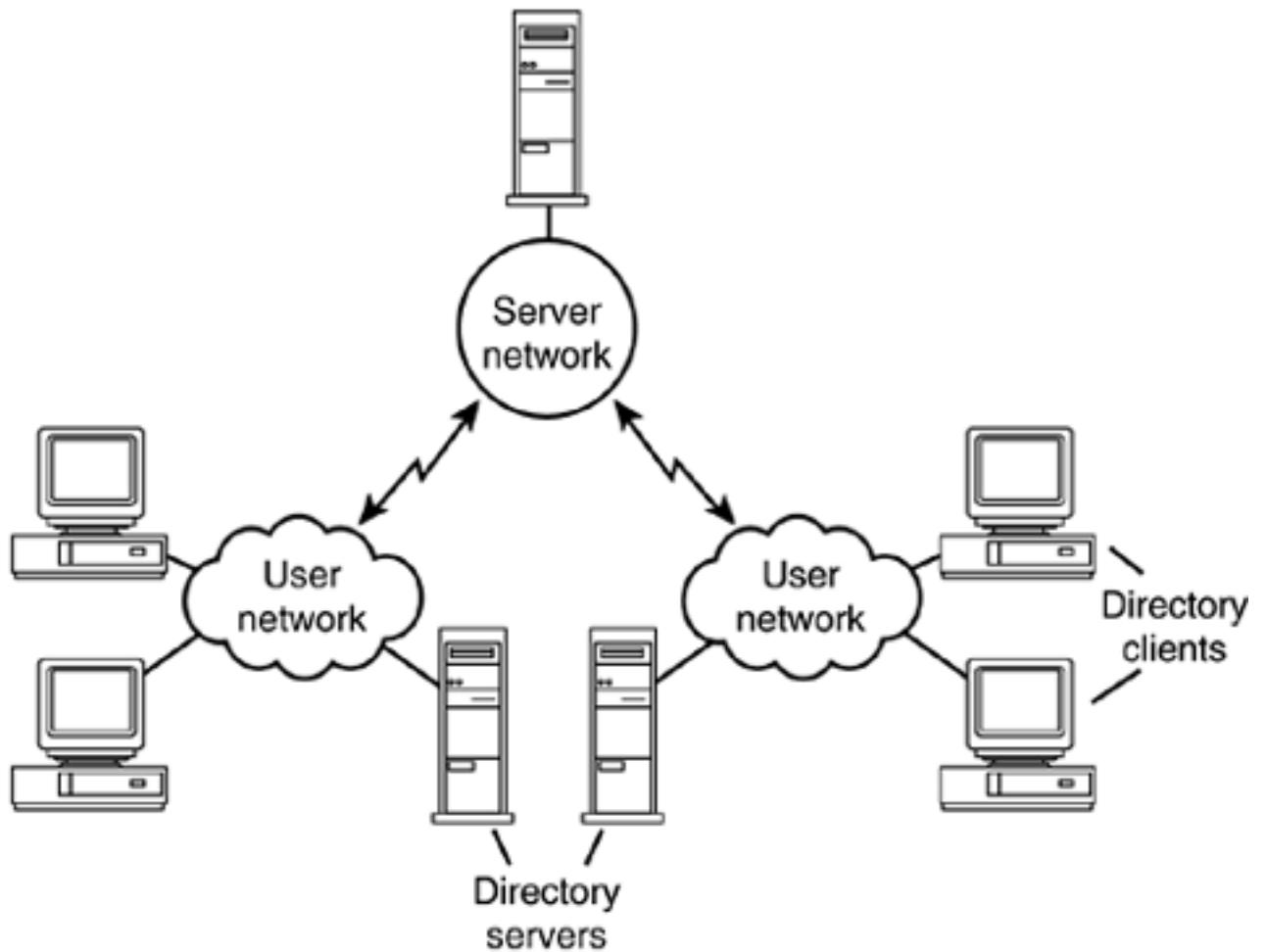


Table 12.4. A Sample Attribute Privacy and Integrity Table

Attribute	Privacy	Integrity
<code>cn, sn, givenName, name</code>	No need for protection	No need for protection
<code>mail</code>	No need for protection; e-mail addresses are public information	Must be guaranteed to avoid misrouting e-mail
<code>salary</code>	Must be protected; salary information is private to an employee and the employee's manager	No need for protection; attribute is read-only in the directory

The design decisions concerning SSL include how you will address the performance issues discussed earlier and how you will handle the components of the certificate life cycle: certificate creation, distribution, renewal, and revocation. Performance is one of the most important issues to be addressed. Making your directory secure at the expense of performance is not a good solution. Performance with security can be expensive; it takes a lot of CPU power to encrypt and decrypt data that traverses the network. Public key operations involved in initial authentication exchanges are even more expensive.

One solution is simply to buy bigger, more powerful servers. A better and often more cost-effective solution is to use hardware acceleration specifically targeted at cryptographic operations. Such solutions are readily available today, but many directory providers do not support them. If performance is important to you, be sure to make hardware cryptographic acceleration part of your selection criteria.

SSL is based on public key technology. Although a detailed explanation of this technology is beyond the scope of this book (there are some excellent references listed at the end of this chapter), we do want to cover some of the administrative implications of choosing SSL or any solution based on a PKI. The important thing to know is that maintaining the PKI is different from maintaining other kinds of shared-secret or private key infrastructures. With the latter, an administrator maintains a centralized database of passwords (or the equivalent). This database exports an authentication service used by both clients and servers. The idea is that both clients and servers trust the authentication database and therefore can use it as a trusted third party during authentication exchanges.

The details can be significantly more complicated than this, but the principle remains the same: A centrally trusted administrator is in charge of all aspects of issuing identities and credentials, verifying trust, revoking credentials, imposing password and other policies related to authentication, and so on. When a user shows up for work the first day, the administrator creates an entry for that user in the authentication database. If a user forgets her password, the administrator simply resets it for her. When an employee leaves the company, the administrator removes her from the database. Credentials can be temporarily suspended, defined as usable during only certain hours of the day, or placed under other restrictions—all under the control of the administrator.

With a PKI, things are a little trickier. The basic difference is that clients and servers do not need to consult a trusted third party to perform authentication. Instead, the credentials used to authenticate are self-verifying. A user is in charge of generating his own credentials. A trusted third party is used just once, after the credentials are generated, to verify that they belong to the user in question. Subsequent authentication exchanges need not involve the third party.

PKI is helpful in terms of scale and performance because otherwise the trusted third party can often become a bottleneck in a shared-secret system. However, PKI is often not so great in terms of deployability and manageability. For example, how does a user create his credentials? What happens when a user leaves the company and his credentials should be revoked? What happens when the user's credentials expire? What happens when credentials are lost? How are new credentials issued? These questions about the key and certificate management life cycle have made PKI much more difficult to administer than private key schemes.

A Look at PKI

Full coverage of public key technology is beyond the scope of this book. The purpose of this sidebar is to give you a 50,000-foot view of this important and complex technology.

The basic principle of public key technology is a mathematical concept that can be used to relate certain pairs of large numbers (called *keys*) in a special way. If one of the keys is used to encrypt a message, the other key can be used to decrypt the message, and vice versa. Fundamental to this scheme is another property: Only these two keys (called a *key pair*) are related in this way. In other words, if a message is encrypted with one key, it can be decrypted only by the matching key in the pair (or at least, decrypting would be very difficult without the matching key). One key is called the *private key*, and the other is called the *public key*. The idea is that you are the only one who knows your private key, and you publish your public key only as widely as you want.

Given this background, you can see how I might send a private message to you: I use your public key to encrypt the message, which I then send to you, and you use your private key to decrypt it. I know that only you can read the message because to decrypt it requires your private key, which nobody else has. There is only one flaw in this scheme: How do I know that it's really your public key I'm using to encrypt this message? This is the purpose of a certificate.

A certificate binds a public key to an identity (and possibly other information about that identity). The idea is simple: You and I share a trusted third party (for example, a mutual friend, an organizational administrator, a government agency). If you go to that trusted third party and prove your identity and present your public key, that third party wraps up and signs your public key along with your identity and any other appropriate information. This neat little package of information is called a *certificate*, and the process of getting one is called *certificate issuance*.

A certificate has two interesting properties. First, it is signed by the trusted third party (called a *certificate authority*, or *CA*). This means that if the certificate is tampered with, I'll be able to tell. Second, it is possible for me to look at the certificate and verify that it was signed by the party we both trust. This is the mechanism by which I can be assured that your public key really belongs to you, at least to the level that I trust the certificate authority.

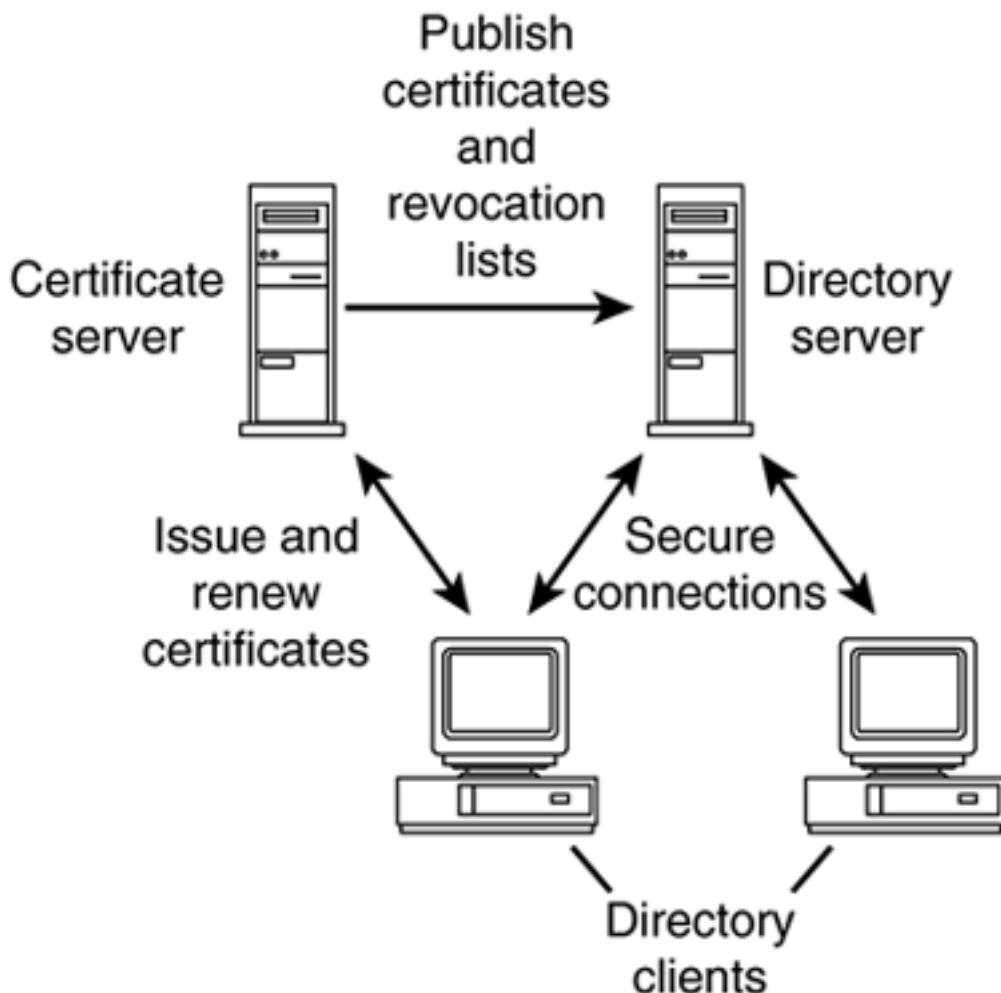
Just as in real life, you can imagine building chains of trust between certificate authorities. For example, you and I might not share a trusted third party, but the entity I trust and the entity you trust might share a trusted third party. Depending on the level of trust we both have in our trusted entities, this might be enough for us to trust each other. The area of trust management is an important and often complicated aspect of PKI.

Another important concept in PKI is revocation. What happens if I lose my private key or if it falls into the wrong hands? If your credit card is stolen, the thief can rack up charges on your account. Similarly, but with potentially more serious consequences, if your private key is stolen, the thief can read messages intended only for you and—perhaps even worse—send messages that look like they came from you. What makes this situation more serious than losing your credit card is that you might not be able to tell that your key has been compromised; you don't get a monthly statement detailing its use. Nor do you typically have any protection from the malicious things someone might do with it, whereas your liability for a stolen credit card is limited.

In this situation the solution is to revoke your certificate. This means that the CA that signed your public key wants essentially to "unsign" it. Because that's not possible, the CA might resort to publishing a list of revoked certificates—a *certificate revocation list*—that should be consulted by security-conscious applications. Another approach is to have applications ask the CA (or its agent) each time the certificate is used whether it is still good. This solution, although attractive in many ways, reintroduces the central bottleneck that we thought we got rid of by moving to PKI!

[Figure 12.10](#) depicts the major components of a typical PKI solution.

Figure 12.10. Components of a PKI Solution



Lest you be scared away from PKI by this bleak presentation of the management challenges it poses, we should quickly add that products exist that improve the deployability and manageability of PKI by a quantum leap. The Netscape Certificate Management System (CMS) is a good example. By using the directory as a central hub for managing the PKI life cycle, CMS achieves the best of both worlds. The more scalable advantages of PKI are retained and combined with the centralized administration model of private key schemes.

The point is simply this: Although SSL comes with many benefits that often make it worth deploying, administrative and performance costs are involved too. You can reduce administrative costs by choosing a complete solution that includes a significant manageability component. You can reduce performance costs by using special hardware to

help speed up expensive cryptographic operations. Be sure to include PKI deployment and manageability, as well as performance requirements, in your design and evaluation criteria if you choose to deploy SSL.

Administrative Security

All the security technology in the world is not worth much if it is improperly applied and administered. The most elaborate home security system in the world, for example, does little or no good unless it is properly installed and activated, sensible precautions are followed in its operation, and the alarm passcode to defeat the system is not written on a note stuck to the wall near the system's keypad. Computer security and directory security are similar. This section examines some of the more common administrative directory service pitfalls that can contribute to security problems.

Enforcing a password policy is important for password-based security to be effective. If you have ever run a password-cracking program on your users' passwords, you will know that people tend to choose horrible passwords that are easy for an attacker to guess using either a little knowledge of the person or a brute-force attack. There are three things you can do to mitigate these problems:

1. Educate your users.
2. Enforce a password policy.
3. Run password crackers.

First, you can educate your users on how to choose a good password. Users should follow these rules when choosing passwords:

- Passwords should not be related to the user's name or login ID.
- Passwords should not be words that can be found in the dictionary.
- Passwords should be at least eight characters long. The longer the password, the better.
- Passwords should contain a combination of letters, numerals, and punctuation.

Also teach your users the consequences of losing a password, and the minimum precautions they should take in daily password management. Often users simply don't realize the consequences of their password being compromised. If a user is aware of these things, she still might not think of the nontechnology-oriented ways in which her password can be compromised (for example, the sticky note she has stuck to her computer monitor screen). Education can go a long way toward raising awareness in your user community—and therefore reducing problems.

Second, you can select software that comes with some kind of password policy capability. For example, Netscape Directory Server 6 and Microsoft Active Directory let you specify things such as the minimum length of a password, the mix of characters it must contain, how often it must be changed, and whether it's acceptable for users to choose passwords that they have used in the past. Having such guidelines in place helps enforce the policy when user education has failed because either you didn't reach users with your education campaign or because users willfully ignored all the fine advice you gave them.

Third, you can ferret out bad passwords by regularly running a password-cracking program such as Crack on your directory. The reason for doing this is simple: If you can do it, you can bet that people attacking your system will do it also. Your goal is to crack and change bad passwords before an attacker does. An even more effective approach is to enforce

minimum password quality criteria and reject passwords that are easily crackable before they are entered by users. Keep in mind that password cracking can be a controversial subject; it often takes careful reasoning to explain its usefulness to management. And you should be careful with the results of your cracking activities to avoid upsetting your users and making them distrust you and the directory service. For more information on Crack, see <http://www.users.dircon.co.uk/~crypto/download/c50-faq.html>.

Administrative data-gathering procedures are another area in which security mistakes are commonly made. Think about the path your data takes from start to finish. If you are bulk-loading data from a corporate human resources relational database, consider the total path the data takes from the time it leaves the database to when it arrives in your directory. Is the data transferred via File Transfer Protocol (FTP)? If so, how is the data protected on the way? Who has access to the data at the relational database source? Do they follow the same level of security procedures that you have instituted for your directory?

Ideally, the security of your whole system would be equal at each component. For example, if your data is protected via certificate-based authentication and encryption via SSL in your directory, you would like it to be protected in the same way in the source as it is when transferred to your directory. But be careful not to fall into the trap of viewing your solution as either secure or not secure, with nothing in between. In evaluating the total security of your system, you must consider not only the possible weak links, but also how difficult and probable it is for an attacker to exploit them.

Respecting Your Users' Privacy

We've talked about privacy as it relates to the overall security design problem. In this section we talk about privacy from the perspective of the directory user. It's important to keep this perspective in mind as you design and deploy your system. Failure to do so can create irate users, headaches for you, and, ultimately, a directory service that fails to win the hearts and minds of your user community.

If your directory is used solely as a store of configuration and preference data for your users, the intention probably is that one user should never be able to see information about another user. In this case protecting the privacy of user information involves primarily protecting the data from unauthorized access. In addition, your users may have concerns about how you will use the data. Many organizations that collect data about their users publish a detailed privacy policy that describes what will and will not be done with the data, with whom it will be shared, and how to opt out of that sharing if possible.

If your directory data is intended to be shared and viewed more widely than this, more planning is required. Our experience suggests that you will come across different kinds of users with different kinds of privacy expectations and requirements.

On one extreme is the user who does not want any information about him published in the directory at all. This user is often angry when information about him is published—angry not just with you, but with those who maintain other information sources on the network, including Web-based phone books, credit reporting services, telemarketing databases, and others. This user feels these information sources are an invasion of his privacy, and he may not settle for anything less than being removed from the service.

Think carefully about how important it is to your service to have complete coverage within your organization. If it's not critical, perhaps the best approach is simply to remove such users from the directory. You can often head off such problems by requiring users to sign up for service in your directory rather than publishing information about them without their permission. This latter approach is clearly to be avoided if possible, but often it is simply not possible. Company policy usually wins out over the desire of your users. Regardless of which

approach you choose, be aware of the consequences and have an answer ready for complaints like this.

Another type of user might have a specific concern about having personal information published in the directory. For example, she might have been the victim of a stalking incident and might be trying to keep as low a profile as possible. This was a real example we encountered while running the directory service at the University of Michigan. We felt we had two choices to address the situation: We could remove the user entirely, cutting off her access to computing services she wanted to use, such as e-mail; or we could convert her entry into an anonymous entry in which all identifying information had been removed but e-mail, and other nonidentifying information remained. We chose the latter approach, and the result was a happy user.

Pay particular attention to cases like this. Users in this category have legitimate concerns about their privacy, and you should respond swiftly to remedy the situation. Do not be surprised if you receive requests that seem bizarre at first. After talking to the user, you will usually understand his perspective, and the request will no longer seem so odd.

Security versus Deployability

We end this chapter by discussing the important topic of security versus deployability. Many security schemes have failed because they were not deployable. You would do well to keep in mind that the most secure service in the world is the one that is never deployed and used. Unfortunately, this is also not a useful service. Similarly, a solution that is deployed but provides no real security will soon become useless because the information it provides cannot be trusted.

Your job when designing your directory's security infrastructure is to find the balance between these two extremes. Where that balance lies depends on the needs of your users, the security threats your system is likely to face, and the consequences that will result from a security breach. A related factor is the amount of time and money you are willing to spend to deploy and maintain your system.

It's easy to get caught in one of the two extremes of the security spectrum. At one extreme, your only interest is deploying your directory service; security seems to be only a roadblock in the way of achieving that goal. Such thinking leads to a directory that will probably not be useful or trusted for long. Worse consequences are possible if the directory contains sensitive information that is compromised.

At the other extreme, you can become wrapped up in security for its own sake; you forget that security is only a means to an end. This extreme can lead to one of two places: One possibility is that you get bogged down trying to make the perfect security system, and meanwhile your project is canceled or taken away from you. Another possibility is that you design and deploy such a system, but none of your users show up to get their fingerprints taken so they're able to use the system, and the system fails. Either way, you end up without a directory service.

You would do well to keep the security versus deployability trade-offs firmly in mind throughout your design process. This advice may seem obvious, but we have seen many directory projects go astray in this area.

Privacy and Security Design Checklist

To review, here are the important points to consider when you're designing for privacy and security:

- Understand the information in your directory, how it needs to be protected, and from whom it needs to be protected.
- Think about who might want to compromise the security of your directory and their motives for doing so.
- Understand the ways in which the security of your directory can be compromised.
- Familiarize yourself with the various tools and techniques you can use to secure your directory.
- Understand the sensitivity of the data stored in your directory.
- If replication or synchronization is used, understand the places to which directory content is being replicated or synchronized.
- Understand the network environment and the capabilities it might offer a hostile person.
- Consider the physical security of the directory servers and backups of the directory data.
- Be aware of any company policies and legal requirements that may apply to you.
- When designing your directory, strike a balance between security requirements and usability.

Further Reading

Building an X.500 Directory Service in the US (RFC 1943). B. Jennings, 1996. Available on the World Wide Web at <http://www.ietf.org/rfc/rfc1943.txt>.

Computer Emergency Response Team (CERT) Coordination Center, <http://www.cert.org>.

Counter Hack: A Step-by-Step Guide to Computer Attacks and Effective Defenses. E. Skoudis, Prentice Hall PTR, 2002.

Hacking Exposed: Network Security Secrets & Solutions. S. McClure, J. Scambray, and G. Kurtz, McGraw-Hill Professional Publishing, 2001.

Internet Cryptography. R. Smith, Addison-Wesley, 1997.

Intrusion Detection Hotlist, <http://www.cerias.purdue.edu/coast/ids>.

Network Intrusion Detection Systems FAQ. Robert Graham, 2000. Available on the World Wide Web at <http://www.robertgraham.com/pubs/network-intrusion-detection.html>.

Privacy on the Line: The Politics of Wiretapping and Encryption. W. Diffie and S. Landau, MIT Press, 1998.

Simple Authentication and Security Layer (SASL) (RFC 2222). John Myers, 1997. Available on the World Wide Web at <http://www.ietf.org/rfc/rfc2222.txt>.

The TLS Protocol Version 1.0 (RFC 2246). T. Dierks and C. Allen, 1999. Available on the World Wide Web at <http://www.ietf.org/rfc/rfc2246.txt>.

Understanding Digital Signatures: Establishing Trust over the Internet and Other Networks. G. Grant, McGraw-Hill, 1997.

Understanding the Public-Key Infrastructure. C. Adams, S. Lloyd, and S. Kent, New Riders Publishing, 1999.

Looking Ahead

This chapter is the last in [Part II](#), Designing Your Directory Service. At this point you should have a complete directory system designed! Now it's time to move on to the deployment phase, which includes validating your design, running pilots, and going into production. [Part III](#), Deploying Your Directory Service, will take you through these steps, showing you how to change your design as you go. First up is [Chapter 13](#), Evaluating Directory Products.

Part III: Deploying Your Directory Service

[13. Evaluating Directory Products](#)

[14. Piloting Your Directory Service](#)

[15. Analyzing and Reducing Costs](#)

[16. Putting Your Directory Service into Production](#)

Chapter 13. Evaluating Directory Products

- Making the Right Product Choice
- Categories of Directory Software
- Evaluation Criteria for Directory Software
- Reaching a Decision
- Evaluating Directory Products Checklist
- Further Reading
- Looking Ahead

In [Part III](#), Deploying Your Directory Service, we turn our attention to the process of turning your directory design into a concrete, functional service. In this chapter we discuss how to select the right directory software. We will then focus on how to pilot your directory service to validate and fine-tune your design in [Chapter 14](#), Piloting Your Directory Service. We will pause for a moment in [Chapter 15](#), Analyzing and Reducing Costs, to look at the costs of the directory service and how they can be minimized. Finally, we will throw the switch and put the directory service online in [Chapter 16](#), Putting Your Directory Service into Production.

It is important to choose the right software to power your directory deployment. Directory server and client products are available from a variety of sources, varying widely in capabilities and cost. If you choose the wrong products, your directory service deployment may fail or at least be more cumbersome, costly, and less effective than it should be. If you choose the right products, your deployment should succeed with maximum effectiveness and minimum pain and cost. Your challenge is to pick the products that will best meet your directory needs and work well in your environment.

In this chapter we help you choose the right products. We focus mainly on directory server software because that is likely to be the first and most important software you choose. We also touch briefly on client software, directory-enabled application software, and software development kits (SDKs) for LDAP. Because the directory software market is evolving rapidly, we do not cover specific software packages.

We begin by presenting some reasons why it is important to choose the right directory products, then continue by looking at some general categories of directory services software. Next we talk about and illustrate how to create a good set of evaluation criteria. Finally, we present some advice that will help you devise a good decision-making process.

Making the Right Product Choice

You are probably already convinced that choosing the right directory server and client software is important. Anyone who has spent any amount of time working with computer systems knows that software often makes all the difference. Let's take a brief look at why choosing the right software for your directory service is so important.

Next to developing a good directory design (which we have already discussed extensively, in [Part II](#), Designing Your Directory Service), choosing good software does the most to ensure the success of your directory deployment. A bad directory design can't be repaired by good directory software, but a good design can be ruined by bad software. By "bad software" we mean primarily software that does not meet your needs, although you should also avoid software that is buggy, is poorly supported, or has another serious defect.

There is no one-size-fits-all directory solution; directory software varies widely in its capabilities. This is a good thing. Although your directory requirements may overlap greatly with those of other organizations, nearly every deployment has some unique requirements. The greatest challenge in choosing directory software is to find the best match between your needs and the capabilities of the available products. (Directory service requirements were introduced in [Chapter 6](#), Defining Your Directory Needs, and were discussed throughout [Part II](#).)

You may run into serious and expensive problems if you do not realize right away that you chose the wrong software. The situation is most common with a growing directory service in which your performance, scalability, and applications needs grow steadily over time. In the worst case, you may not notice that you have a problem until it is too late to fix it gracefully, and the result may be dissatisfaction with your service.

For example, if the use of your directory increases over time, you may notice only a gradual degradation in performance. If gradual performance loss is acceptable, there is no problem. However, if your directory server software reaches a cliff at some point and performance falls off abruptly instead of degrading gracefully, some directory-enabled applications may be unable to cope. If applications just stop working one day (which is possible), you will find yourself really scrambling to devise a quick fix.

Many hidden costs may be associated with choosing the wrong software as well. Small deficiencies in performance, scalability, ease of administration, and quality of support for applications and standards can lead to increased costs for maintaining your service and all the applications that surround it. Conversely, a directory server product that can accommodate 10 percent more traffic may allow you to deploy five servers rather than six, which could result in a savings of hundreds of thousands of dollars over the lifetime of your service.

Another factor that makes choosing the right software crucial is that it can be expensive and time-consuming to replace one software product with another, even though open standards such as LDAP tend to reduce such costs. Some costs you may still incur when replacing directory services software include the following:

- Money spent to purchase the new software. If you can't recoup the money you spent on the software you're replacing, your software costs may be double what they would have been if you had made the right choice in the first place. You may also need to purchase new hardware or upgrade existing hardware to accommodate the new software, because of differences in either supported platforms or performance characteristics of the replacement software.

- Time and money spent redesigning your directory service and directory-enabled applications to fit the new software's capabilities.
- Time and money spent to learn the ins and outs of the new software.
- Time and money spent to pilot and deploy the new software. This can be very expensive for directory server software, especially if you have already deployed a large number of replicas or a widely distributed directory across many servers.
- Lost productivity because end users and administrators have to learn how to use yet another set of software.

As you can see, this is a pretty scary list. Therefore, it is important to perform a proper evaluation of the available software products to make the best choice up front.

Team LiB

◀ PREVIOUS

NEXT ▶

Categories of Directory Software

Because it is difficult to provide all things to all people, software vendors tend to focus their product development efforts on meeting specific needs. By comparing a vendor's focus to your directory needs, you can often shorten your list of candidate products quickly.

Although LDAP is a general-purpose protocol, the requirements of one directory deployment may be different from those of another. For example, an LDAP server implementation that provides strong security features might be well suited for deployment on the public Internet, whereas another product that provides minimal security might be a great product for small workgroups. These two products are appropriate to use in different situations.

There are many ways to categorize the available products. One of the most useful ways is to look at the various applications they aim to support:

- Network operating system (NOS) applications
- Intranet applications
- Extranet applications
- Internet-facing hosted applications
- Lightweight database applications

Each category is described in the following sections.

NOS Applications

From the directory software perspective, a NOS such as Microsoft Windows 2000 Server or Unix NIS is just another application with a specific set of needs. Directories that work well with NOSs are generally focused on basic network services such as logon, access control, and management of LAN services (such as file service and printing). In most cases, a directory that works with a specific NOS is not separable from the NOS itself, and LDAP is typically grafted onto existing products (although the current trend is toward better support for LDAP and other open standards such as XML). For products in this category, integration and ease of management are important, and performance, scalability, and support for multiple platforms are deemphasized.

Intranet Applications

The term *intranet* describes networks inside organizations that are based on open Internet technology. The trend toward use of Internet technology inside organizations started with the adoption of Web servers and browsers inside the corporate firewall and has moved on to encompass messaging, groupware, and directory products. An example of a directory-enabled intranet application is the Netscape Enterprise Server Web server software that uses LDAP for access control and authentication.

Directory software suitable for use within intranets is typically designed to support a wide variety of end-user and server applications, such as corporate phone books and high-volume messaging servers. Ease of management, performance, and scalability are all important within this category.

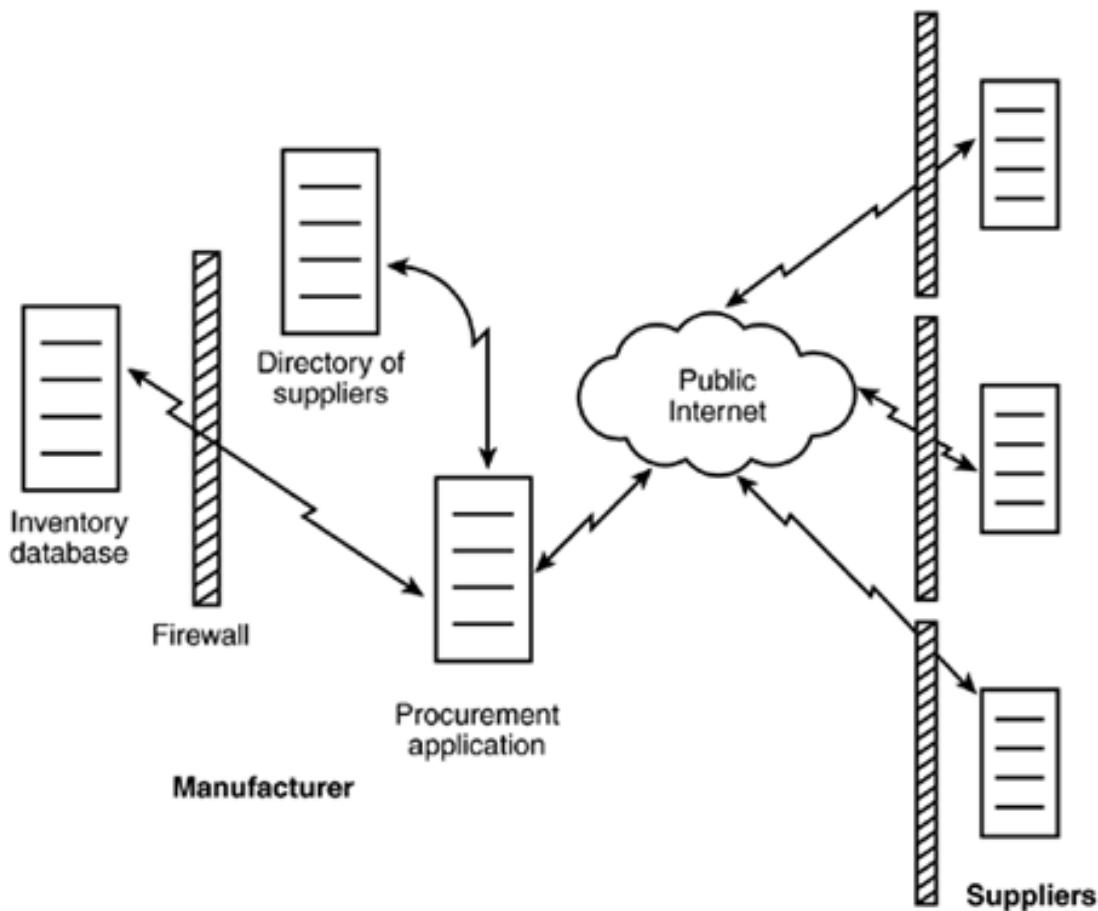
Extranet Applications

The term *extranet* describes business-to-business communication networks that are based on open, Internet technology. Extranets are usually *virtual* networks in the sense that they are typically formed via secure connections over public networks such as the Internet itself rather than via new physical connections. Extranets are gaining momentum because they can be used to deploy completely new applications and replace expensive, proprietary electronic data interchange (EDI) networks.

Extranet directory-enabled applications typically serve large numbers of people. In addition, many of the people served may not be directly employed by the organization that hosts the application. For example, extranet applications often connect manufacturing organizations to their suppliers. Such an extranet application needs to store information about people who work for both the manufacturing

organization (the company hosting the extranet application) and its suppliers. [Figure 13.1](#) illustrates this kind of extranet procurement application.

Figure 13.1. An Extranet Procurement Application



Security and privacy are often the most critical directory features required of extranet applications because the application and its directory service may be accessible through the public Internet. Good performance and scalability of a directory product are also critical for large-scale extranet applications.

Internet-Facing Hosted Applications

The business of an Internet service provider (ISP) is to provide Internet application access for end users and host applications on behalf of other organizations. ISPs typically provide services for many users, and some ISPs provide hosting services for thousands of organizations. As Internet applications such as e-mail increasingly depend on directory services, ISPs find themselves providing directory services too.

Recently, a greater number of large organizations that do not consider themselves ISPs have been finding that the most efficient way to streamline their business processes is to act as service providers for their divisions, departments, external partners, and suppliers. These enterprises that have many of the same requirements as traditional ISPs have been dubbed *enterprise service providers (ESPs)*.

For ISP and ESP directory services to work well, their applications must meet a wide variety of requirements and be inexpensive to manage. Flexibility, scalability, performance, and automatable management are all important in this segment of the market.

Lightweight Database Applications

A relatively new role for LDAP directories is to replace traditional database systems. The typical application being serviced is lightweight and query intensive. Usually the data is widely distributed and shared by many applications. For example, a bank may need to share profile information about employees, customers, and partners with a wide variety of applications at thousands of distinct

locations.

You can achieve high-performance, standards-based access to any kind of data by placing the data in an LDAP directory and replicating it widely. Directory scalability (up to tens of millions of entries!), reliability, and write performance may all be important for these kinds of applications. Access to the directory data using an XML-based representation such as Directory Services Markup Language (DSML) is also becoming important because many applications are being written in an XML-centric manner.

[Table 13.1](#) summarizes these application categories and lists the typical requirements that must be met by directory products.

Table 13.1. Application Categories and Their Requirements

Application Category	Focus	Requirements					Acceptable Cost per User
		Flexibility	Security and Privacy	Performance	Scalability		
NOS	Logon, access control, and management	Low	Moderate	Low	Low	High	
Intranet	End-user and server applications	Moderate	High	High	Moderate	Moderate	
Extranet	Interorganizational applications server up on the public Internet	Moderate	Very high	High	High	Low	
Internet/ Hosted	End-user and server applications for a variety of unrelated individuals and organizations	High	High	Very high	Very high	Low	
Database	Lightweight database applications in need of a standard access protocol	Very high	High	Very high	Very high	Moderate	

Just looking at the category that a specific directory product claims to address is no substitute for using the comprehensive evaluation criteria we help you create in the next section. However, it can help you quickly weed out products that are unlikely to meet your needs.

Evaluation Criteria for Directory Software

One of the first tasks in choosing directory software is to develop evaluation criteria that take into account your current and future needs. Whether you have no idea what software you might use or you already have a specific set of products in mind, evaluation criteria are valuable. A good set of evaluation criteria helps you choose, justify, and feel comfortable with a collection of products.

In this section we provide some advice for developing evaluation criteria to fit your own situation. We divide the criteria into the following areas:

- Core features
- Management features
- Reliability
- Performance and scalability
- Security
- Standards compliance
- Interoperability
- Cost
- Flexibility and extensibility
- Other considerations

Each area is discussed in the following sections. As you read through these criteria, examine your own directory service needs and your design to see which factors are most important to you. Your goal is to create a list of requirements and questions you have about the products. You should assign a priority or weight to each item on the list so that you can make intelligent trade-offs later.

Your list of evaluation criteria will serve as a tool for evaluating products and help you ask the right questions of vendors about what they can deliver. Keep in mind that the areas we cover in the following sections are not exhaustive and that there will be other factors you should consider as well. Use this information only as a starting point to develop your own set of evaluation criteria. And don't neglect the areas in which you have no immediate needs; be sure to take future requirements into account.

Core Features

When you're looking at the core features of LDAP software products, your main task is to make sure that the product can meet the needs of all the directory-enabled applications you plan to deploy now and in the future. This is not a small task, of course, but the work done in [Chapter 6, Defining Your Directory Needs](#), and throughout the rest of [Part II, Designing Your Directory Service](#), should provide all the information you need to create a list of the features that are important to you. Areas to consider include the following:

- Support for the hardware and software platforms you use now or plan to use in the future.
- Support for all the core LDAP operations and extensions needed by your applications.
- Support for alternative data formats and access protocols such as DSML that are needed by your applications.
- Replication features, including support for all the topologies your design uses, the capability to provide uninterrupted service to client applications even if one or more replicas fail, flexible scheduling policies, and so on. [Chapter 11, Replication Design](#), includes detailed product criteria for replication.
- Support for distributed directories, if required by your design. Such support usually

comes from LDAP referrals or server-to-server chaining of requests.

- Data import, maintenance, and backup features.
- The quality, availability, and extra cost of documentation and support. For example, will the vendor provide a dedicated account representative and a dedicated technical contact for you? And what kind of response time will the vendor guarantee?

Management Features

You will need to manage both the directory service itself and its contents. When examining the management features of software, focus on your most critical needs and look for flexibility to meet future requirements. Areas to consider include the following:

- Simplicity and flexibility of installation scripts and procedures.
- Availability of good tools for configuring, monitoring, and maintaining all aspects of the service itself. Look for tools that are easy to use for complicated tasks such as security, access control, and performance tuning. Often both graphical and command-line tools are desirable, depending on your situation and preferences.
- Availability of a variety of LDAP client interfaces that users and administrators can employ to find and manipulate directory content such as person entries, group entries, and any other data you plan to store in your directory. The best client interfaces can be customized to target specific needs.
- Scriptable administration and content manipulation tools that can be used to automate frequent tasks.
- The capability to manage user-specific constraints such as password policies, resource usage quotas, and other items.
- Remote administration capabilities.

Reliability

All directory services are expected to be reliable so as not to inconvenience applications and halt business processes. Specific requirements, of course, vary widely. For example, if your directory will serve primarily as an electronic phone book but everyone has paper copies of the same data available for use if your service is unavailable, reliability is not so critical. However, an extranet application used by customers and business partners to update their contact information requires a very reliable service.

Here are some reliability issues to consider when creating your evaluation criteria:

- Support for 24x7 (continuous) operation. Features such as the capability to back up data and change the server configuration without affecting the running service can be important. Common configuration changes include extending the directory schema, adding indexes to improve performance, and making temporary changes needed during upgrade and maintenance activities (such as a change in the TCP port on which a server listens).
- A robust, transactional server data store that is self-repairing and recovers automatically from temporary failures such as power outages.
- Support for automated failover to minimize downtime in the event of software or hardware failures. With directory server software, such support can be provided as a by-product of support for third-party or built-in high-availability solutions, advanced replication features such as multimaster topologies and hot standbys, and client-side support for transparent failover to a standby server.
- Directory service monitoring tools. As discussed in [Chapter 19](#), Monitoring, monitoring is essential for quickly identifying and correcting problems. Support for Simple Network Management Protocol (SNMP), server error logging facilities, and the availability of proprietary monitoring tools can all be helpful. Otherwise you need to write your own monitoring tools. (If you think you'll need to do this, make sure that

appropriate hooks and SDKs are provided with the directory products you choose.)

Performance and Scalability

High performance and high scalability are important in many directory deployments. The directory design process described in [Part II](#), Designing Your Directory Service, should help you expose all your performance and scalability requirements. Areas to consider include the following:

- Latency associated with directory operations. For example, how fast must the directory service respond to a search or modify request?
- Throughput of directory operations. How many operations can a directory server handle in a given time period? How fast can a client application or SDK send LDAP requests and process the responses? How many search, add, and modify requests can the server process per second? Be sure to take into account the type and mix of operations you expect to have in your environment. For example, server performance for searches is typically much greater than it is for adds and modifies; thus if your needs imply that there will be a lot of add and modify traffic, you should pay special attention to the write performance of the directory server software.
- The capability of the server to handle many simultaneous LDAP connections while providing good performance to all the clients.
- The capability to monitor and tune the performance of the directory server to meet your needs. This is important in determining the characteristics of the hardware you should purchase, as well as in optimally meeting the requirements of your most important directory-enabled applications.
- The capability of server and client software to maintain good performance as scale increases. Dimensions to consider include the number and size of entries, number and size of attribute values, number of access control rules, number of replicas, and number of directory partitions.
- The availability of performance information such as benchmark results and performance-focused documentation. If performance is important to you but a directory software vendor has little to say about it, you may need to look at other products or conduct your own benchmark tests.

Directory Performance Testing

To determine how well a directory product will perform when you deploy it, you can test its performance in a laboratory setting. Creating an accurate performance test that produces meaningful results is often difficult. To closely model your own directory data and directory-enabled applications, you may need to develop your own custom benchmarking tools. This can be a lot of work, but it may be worth tackling up front. Understanding the performance characteristics of your directory software will help you make informed decisions as you finalize your design and deployment plans.

DirectoryMark, jointly developed by Mindcraft and Netscape, is an off-the-shelf tool for benchmarking LDAP directory servers. This tool is highly configurable, so you may be able to use it to simulate the client load you expect to impose on your servers. Connect to Mindcraft's Web site at <http://www.mindcraft.com/benchmarks/dirmark> for more information.

Security

As discussed in [Chapter 12](#), Privacy and Security Design, security requirements vary widely from one directory service deployment to another. Without exception, though, security should be an important part of your product evaluation criteria. Areas to consider include the following:

- Access control capabilities, including adequate flexibility and granularity to meet your privacy, security, and data management requirements.
- The capability to create administrative domains and delegate administration of directory content to independent administrators and end users as needed. Graphical interfaces that support delegation, such as Netscape's Delegated Administrator component, are also required.
- Auditing features to maintain a record of all access and changes to directory data. Ideally, access to the audit information itself should be controlled through access control and separation of administrative domains.
- The authentication methods supported by the client and server software (for example, LDAP basic authentication, various Simple Authentication and Security Layer [SASL] methods, and certificate-based authentication).
- The encryption capabilities for LDAP data streams and directory data stores, including support for Secure Sockets Layer (SSL) and Transport Layer Security (TLS).
- The capability to use SSL or TLS to protect replication updates in transit and to protect operations that cross server boundaries when server-to-server chaining is being used.
- Support for hardware acceleration of cryptographic algorithms. This is important for servers that must perform a lot of encryption or that routinely handle digital certificates.

Standards Compliance

To most people, standards are not interesting for their own sake; the products that adhere to standards are more interesting. Standards-compliant software is important to you because it provides increased flexibility, better interoperability, more customer choice, and a proven, well-understood core feature set.

Standards documents are typically technical and difficult to understand. Although you probably do not need to read and understand all the standards documents, you should be aware of all the emerging and ratified standards so that you can ask software vendors whether they comply with them. Because no available product supports all the standards, you need to determine which standards are most critical to you. You can do that by understanding what each standard specifies and how it aligns with your own directory requirements.

The most important group that creates standards for LDAP is the Internet Engineering Task Force (IETF), which is the standards-setting body for the Internet. IETF standards are published as a series of documents called *Requests for Comments* (RFCs). Note that not all RFCs are destined or even intended to become standards, and that specifications are first published as Internet Drafts. (See [Chapter 1](#), Directory Services Overview and History, for more information on the IETF and how it operates.)

Other important standards are produced by industry consortia, and some de facto standards are developed by the leading directory services vendors. Directory service standards that you should consider for your evaluation criteria list include the following:

- The core LDAP Internet protocol standards, including the complete set of LDAPv3 RFCs. The relevant documents are RFCs 2251 through 2256, 2829, 2830, and 3377. Because these standards are the building blocks on which all LDAP-related software is built, most products claim to support these RFCs. However, be sure to ask your software vendor exactly which aspects of these LDAP standards they do not support.

- Related Internet standards that LDAP has adopted, including SASL (which is defined in RFC 2222), SASL digest authentication (RFC 2831), and the Directory Services Monitoring Management Information Base (MIB; RFC 2605). You can use SASL to integrate with a specific authentication scheme. Support for the SNMP MIB for directory services helps you integrate your servers with off-the-shelf network management systems (NMSs).
- LDAPv3 extensions, such as those for dynamic entries (RFC 2589), the password modify extended operation (RFC 3062), storing vendor information in the root DSE (RFC 3045), server-side sorting of search results, Virtual List View, discovery of LDAP services using the Domain Name System (DNS), connectionless LDAP (over UDP), authentication response control, proxied authorization control, LDAP password policy, schema update procedures, standard access control, and standard replication protocols. At the time of this writing, many of these extensions are available only as Internet Drafts; they have not yet been published as RFCs. In general, you should examine each extension to see whether you need the feature it provides within your own directory deployment.
- The LDAP Data Interchange Format (LDIF), which defines a standard format for representing directory entries and changes to entries. LDIF is defined in RFC 2849.
- LDAP application programming interface (API) standards within SDKs and LDAP clients. Internet standard LDAP APIs for C/C++ and Java are being developed by the IETF. Note that several proprietary APIs also exist, such as Microsoft's Active Directory Services Interface (ADSI) and JavaSoft's Java Naming and Directory Interface (JNDI). However, the time and cost incurred in developing LDAP applications can be reduced through the use of standard APIs. See [Chapters 21](#), Developing New Applications, and [22](#), Directory-Enabling Existing Applications, for more information on available APIs and SDKs.
- Schema standards, such as those being developed by the Directory Enabled Network (DEN) initiative. Good LDAP products can be extended to accommodate a wide variety of schemas.
- Security standards, such as X.509 for certificates and the SSL and TLS protocols. Security is an area in which standards are especially important, not just for basic interoperability, but also for easier evaluation of the security provided by the products. For example, all SSL version 3.0 implementations support a certain level of security, but for proprietary security schemes, you need to trust the software vendor to provide adequate capabilities. Security is an area that is not well understood, but standards reduce the learning curve and allow you to transfer knowledge from one product to another much more easily.
- XML directory standards, such as DSML, which is being defined within the confines of the OASIS Directory Services Technical Committee, and Security Assertion Markup Language (SAML), which is being defined by the OASIS XML-Based Security Services Technical Committee.
- X.500 directory service standards. Because LDAP depends to some extent on the X.500 standards for its naming, information, and operations models, the X.500 standards are relevant to all LDAP directories. X.500 also specifies standards for access control and replication that have been adopted by some LDAP vendors. More information on X.500 can be found in [Chapter 1](#), Directory Services Overview and History.
- Any other standards or certification requirements, such as Y2K compliance, certification by an operating system or hardware vendor, or support for industry-leading third-party products such as high-availability and backup solutions.

Interoperability

Standards compliance is an important part of any product's interoperability story. However, if you require interoperability with specific products, you should take that into account when developing your evaluation criteria. Areas to consider include the following:

- Availability and support for directory-enabled applications that you plan to deploy.

Some vendors, such as Sun, offer a complete line of LDAP-enabled Web and messaging products designed to work with their own directory service products. When working with products from two or more vendors, you probably should verify interoperability yourself.

- Availability of synchronization tools for data sources with which your directory service needs to interoperate. For example, you may have a requirement to mirror the data held in a cc:Mail address book in your LDAP directory; therefore, you would need to buy or build a cc:Mail-to-LDAP synchronization tool. See [Chapter 23](#), Directory Coexistence, for more information on integrating your directory service with other data sources.
- Availability of metadirectory solutions to facilitate the joining and synchronization of disparate directories and data stores. See [Chapter 23](#), Directory Coexistence, for more information on metadirectories.
- Proof of interoperability. This can come in the form of results from vendor-to-vendor tests performed in an ad hoc fashion or at an LDAP interoperability testing forum, such as the DirConnect and LDAP Plugfest events sponsored by the Open Group.

Cost

The most obvious product-related cost is that of the software itself, but you should also consider the total cost of buying, deploying, supporting, and maintaining your directory service and the applications that surround it. [Chapter 15](#), Analyzing and Reducing Costs, covers cost analysis in depth, but some general areas to examine when you're creating your list of evaluation criteria include the following:

- **Software costs.** Be aware of the different cost structures under which the software may be sold—per server installation, per CPU, per user, per entry, yearly lease, unlimited use, and so on. Also don't forget to take into account ongoing upgrade and support costs and the licensing costs associated with operating systems or any other software the directory software depends on. For example, at the time of this writing, Microsoft requires that you upgrade all your Active Directory domain controllers to Windows 2000 Server (or later versions) to take advantage of some Active Directory features.
- **Hardware costs.** Different directory service products require different hardware, some of which may be much more expensive than others.
- **Deployment costs.** The costs of hardware, personnel, software deployment, and so on.
- **Maintenance costs.** The costs of day-to-day tasks such as performing backups, maintaining the content of the directory servers, monitoring it for failures, and any other ongoing activities. Look for products that allow management tasks to be easily automated.
- **Training costs.** These include costs to train administrators of the system, end users, and directory content administrators.
- **Support costs.** Products that are easy to learn, use, and manage cost less to support.

Keep in mind that some of your costs will be difficult to quantify, especially if you have not yet deployed your directory service. Talk to people in other organizations who have already deployed a similar product to gather more concrete information about costs.

Flexibility and Extensibility

It is unlikely that an existing product will be able to meet all your needs out of the box. For this reason, and because you can't anticipate all your future needs, it is important to choose directory products that are flexible and extensible. Areas to consider include the following:

- **Configurability.** Look for the capability to selectively enable and disable features, the capability to adapt the product to work well on your available hardware and operating systems, and the ease with which you can make configuration changes. For example, if you have remote offices in your organization, you may want to run some replicated servers in those offices on inexpensive machines that have moderate amounts of memory, CPU, and disk capacity. For your main campus, you may want to run the same directory server software but tune it to take advantage of large machines that have a lot of memory, CPU, and disk capacity.
- **Flexibility.** To support a variety of directory-enabled applications, you want to be able to extend directory schemas easily, add or remove indexes, and otherwise configure directory servers to provide optimal performance to support an application's queries. For example, if you bring a high-performance, directory-enabled application online in the future, it will probably come with its own schema extensions and may require the tuning of your servers to achieve optimal performance.
- **Extensibility.** You can extend some products by writing scripts or creating software plug-ins that alter the behavior of directory clients and servers. Netscape Directory Server, for example, provides a complete set of well-documented plug-in interfaces. This kind of extensibility can help when you need to support an unusual application or when you want to adapt the directory service to fit your organization's data maintenance policies.

Other Considerations

There are a few additional areas to examine as you construct your evaluation criteria. These issues are related primarily to the future of the product and the vendor that provides it. Consider each of the following topics:

- **The future of the product.** Is it evolving and moving in the direction you expect your organization to move? Is the product important enough to the organization that develops it that you can count on continued availability and support?
- **Completeness of product line.** A vendor that sells a complete line of directory-enabled applications and has made LDAP a key part of its corporate strategy may be able to meet more of your needs and serve as a good partner for all your directory services efforts. On the other hand, it is unlikely that you will buy all your directory products from only one vendor.
- **Industry and developer mindshare.** Look at how much support there is for a product outside the company or organization that provides it. A product that is used and supported by many third parties has a better chance of meeting the needs of most people. On the other hand, if your requirements are specialized, this factor isn't as important.
- **The ability to leverage other products and services the vendor has to offer.** For example, Netscape Directory Server promises easy integration of your intranet or extranet site with various AOL Time Warner online services and content. (AOL Time

Warner is the parent company of Netscape, so this is not a surprise.)

- **General vendor issues.** Make sure that the vendor stands behind the products it sells. Determine whether the vendor has a viable business plan so that you can be assured that it will be around to support you. Examine the vendor's track record: What experience have other people had with the vendor?

An Evaluation Criteria Example

In this section we present a sample evaluation criteria list for directory server software. The sample criteria were developed for a fictitious company called On Time Package Delivery that is deploying a directory service for the first time. The focus of the deployment is support of directory-enabled intranet applications. The first two applications that are expected to come online are an electronic phone book and an e-mail delivery service. On Time currently has only 5,000 employees but expects to grow rapidly in the next few years.

The sample criteria are presented as a series of tables that the On Time directory services planning team created using a spreadsheet program similar to Microsoft Excel. Each row in the spreadsheet lists a specific characteristic used to evaluate each candidate product. A description and a weight are provided for each item. The weight is a number from 1 (not very important) to 10 (extremely important) that captures the importance of the item. Items that are extremely critical (*must-have features*) are marked with an asterisk (*) so that you can spot them more easily when reviewing the results of the evaluation. A score of zero for a must-have feature would immediately eliminate a product from consideration.

The right side of each table provides room to evaluate two products (A and B). Each product is given a rating for each characteristic. A score ranging from 0 (poor) to 100 (best) is calculated by multiplication of each item's weight (W) by a product's rating (R). Adding together all the scores produces an objective number that you can consult when making a final product choice.

[Table 13.2](#) shows On Time's evaluation criteria for core directory server features. As illustration of how the scoring system works, consider the first row in [Table 13.2](#): support for all basic LDAP operations. This feature was given a weight of 7 (out of a possible 10) by On Time because it is fairly important to the company (On Time plans to make full use of its directory service eventually, including allowing employees to update their own contact information).

Product A received a rating of 8 (out of a possible 10) because it supports all the basic LDAP operations but falls short on complete implementation of some of the added features of LDAPv3. The item score for Product A is calculated by multiplication of 7 (the weight) by 8 (the product rating) to arrive at 56. Product B received a rating of 3 because it supports only search operations. Product B's score is 21 (7×3).

Table 13.2. On Time's Evaluation Criteria for Core Directory Server Features

Product A	Product B
-----------	-----------

Feature	Weight (W): 0–10	Rating (R_A): 0–10	Score ($W \times R_A$)	Rating (R_B): 0–10	Score ($W \times R_B$)
Basic LDAP operations	7	8	56	3	21
LDAPv3 Virtual List View extension (for phonebook application)	5	10	50	10	50
*Runs on Windows 2000 Server	10	10	100	10	100
Runs on HP/UX 11.11	5	10	50	0	0
Multimaster replication	4	0	0	9	36
*Single-master replication	10	10	100	10	100
Quality of design and deployment documentation	6	2	12	7	42
Quality of administration and configuration documentation	8	4	32	8	64
Total score (out of 550)			400		413

Tip

For an objective numeric evaluation such as this to work, you need to ensure that the grading is done fairly. In practice, the same group of people must assign the ratings for each criterion, or you must spell out in great detail how the products are to be rated. If you can spare the resources, a good approach is to have two or more people rate each product independently and then compare the results to ensure that all parties basically agree.

Because On Time plans to develop its own phone book application and eventually create

dozens of directory-enabled applications, flexibility and extensibility of the directory software are very important. [Table 13.3](#) shows On Time's evaluation criteria for this area.

The sample evaluation criteria we have presented are, of course, far from complete. When you develop your own evaluation criteria, you should include all the areas we discussed in the previous sections, such as security, interoperability, and cost. The evaluation criteria tables shown here could be improved by the addition of a column to record specific comments or notes about each product feature.

Table 13.3. On Time's Criteria for Flexibility and Extensibility

Criterion	Product A			Product B	
	Weight (W): 1–10	Rating (RA): 0–10	Score (W x RA)	Rating (RB): 0–10	Score (W x RB)
Ease of configuration	4	5	20	7	28
Flexibility of configuration	7	9	63	5	35
Will run on low-end server machines	7	7	49	6	42
*Can be optimized for new applications: tunable indexing	9	8	72	5	45
*Can be optimized for new applications: extensible schemas	10	8	80	4	40
Server-side plug-in APIs	2	9	18	2	4
Total score (out of 390)			302		194

Reaching a Decision

Now that you have a good set of evaluation criteria, you need to decide what software products to buy. If you are new to directory services, you may want to hire a consultant to help with your directory product evaluation. Whether you hire someone or tackle this yourself, it will take some time to make a good decision.

Gathering Basic Product Information

The first step in making your decision is to gather as much information as you can about all the available products that may meet your needs. Use the information you gather about each product to grade it against your evaluation criteria.

You can do this initial fact gathering using data sheets and other information that should be readily available from each vendor's Marketing department. You can usually just visit a vendor's Web site to find most of what you need. You might also draw from independent reviews of directory services software, although magazines and other publications that have performed reviews to date have generally focused on narrow portions of the directory product space, such as NOS directories. Industry analysts and professional consultants are also good sources of information.

Quizzing the Software Vendors

After gathering as much initial information as possible, you should contact people at each prospective vendor and ask to meet with them so that they can tell you about their products and you can ask any remaining questions. Your main goal is to define evaluation criteria and rate each product; however, you should also listen to what the directory vendors have to say because they may tell you something about their product that you have not considered. They may even be able to help you improve your evaluation criteria. Of course, you need to watch out for bias and hidden agendas, and you may need to steer the vendors' conversation back to your own needs if they go too far off topic.

Also ask prospective vendors for a list of reference customers whom you can contact to learn more about how customers are really using the product. If you know of others who have chosen a competitor's product, contact them as well and ask them why they made that choice. When obtaining references from the software vendors, ask to be referred to organizations that are similar to yours and that have undertaken similar directory service deployments.

Challenging the Vendors to Show What Their Products Can Do

If a clear winner has not yet emerged from among the prospective products, you may want to challenge each vendor to show what its software can do and how it will meet your needs. You can do this by asking the vendor to help you with a trial design and a small pilot. This test will help you learn not only how well a vendor's products will work in your own environment, but also what it is like to work closely with the vendor.

You can also invite several vendors to come to your business or a neutral place at the same time and conduct a "bake-off" so that you can perform a head-to-head evaluation. If you do this, make sure that you're prepared with sample data, schema, and benchmarks in hand so that the event is productive. Also don't expect to convince any vendors to participate in this kind of event unless you plan to spend a lot of money on directory services software and support.

Conducting a Directory Services Pilot

Always do your own in-house evaluation and piloting of any software you choose before deploying it. There are always surprises with software, a few of which inevitably are unpleasant and show up only in your environment. The process of creating, running, and learning from a pilot directory service is covered in [Chapter 14](#), Piloting Your Directory Service.

Before you create the pilot service, you may want to do some testing in a controlled environment that is free from distractions. Lab testing is a good way to learn the basics of the software and to see how well it scales and how reliable it is under heavy load. See [Chapter 14](#) for specific ideas on how to conduct useful laboratory tests.

Negotiating the Best Possible Deal

After you have made your final software decision, negotiate the best deal you can. Depending on your requirements, you may want to buy just the software from a vendor, or you may want to negotiate a complete package that includes hardware, software, installation, support, and on-site consulting.

In some cases you also need to take political factors into account. You should fight hard to get the best software available, but if your CEO sits on the board of the same charitable organization as does the CEO of one of the directory software suppliers, your hands may be tied. In that case, use the political relationship to your advantage by asking for a better deal, better support, additional critical features, and so on.

When negotiating with a vendor, be sure to leverage some of the information you gathered as part of your product evaluation. Don't be afraid to tell vendors about the strengths of competitive products and the weaknesses of their own products. Having price information for competitive products in hand is also useful when you're negotiating the best possible deal.

Evaluating Directory Products Checklist

Use the following list as a guide when evaluating directory products:

- Understand the categories of directory software.
- Develop evaluation criteria for the following:
 - Core features
 - Management features
 - Reliability
 - Performance and scalability
 - Security
 - Standards compliance
 - Interoperability
 - Flexibility and extensibility
 - Cost
 - Other considerations
- Conduct your evaluation and reach a decision as follows:
 - Gather basic product information.
 - Quiz the software vendors.
 - Challenge vendors to show what they can do.
 - Conduct a directory service pilot.
 - Negotiate the best possible deal.

Further Reading

Access Control Requirements for LDAP (RFC 2820). E. Stokes, D. Byrne, B. Blakley, and P. Behera, 2000. Available on the World Wide Web at <http://www.ietf.org/rfc/rfc2820.txt>.

Authentication Methods for LDAP (RFC 2829). M. Wahl, H. Alvestrand, J. Hodges, and R. Morgan, 2000. Available on the World Wide Web at <http://www.ietf.org/rfc/rfc2829.txt>.

The C LDAP Application Program Interface (Internet Draft). M. Smith, T. Howes, A. Herron, M. Wahl, and A. Anantha, 2000. Available on the World Wide Web at <http://mozilla.org/directory/ietf-docs/draft-ietf-ldapext-ldap-c-api-05.txt>.

DirConnect and PlugFest LDAP interoperability testing. Sponsored by the Open Group's Directory Interoperability Forum. Information is available on the World Wide Web at <http://www.opengroup.org/directory>.

Directory Enabled Network (DEN) Initiative Web site, http://www.dmtf.org/standards/standard_den.php.

DirectoryMark, The LDAPServer Benchmarking Tool. Information is available on the World Wide Web at <http://www.mindcraft.com/benchmarks/dirmark>.

Directory Server Monitoring MIB (RFC 2605). G. Mansfield and S. Kille, 1999. Available on the World Wide Web at <http://www.ietf.org/rfc/rfc2605.txt>.

Discovering LDAP Services with DNS (Internet Draft). M. Armijo, L. Esibov, P. Leach, and R. Morgan, 2002. Available on the World Wide Web at <http://www.ietf.org/internet-drafts/draft-ietf-ldapext-locate-08.txt>.

IETF LDAP Duplication/Replication/Update Protocols Working Group documents. Available on the World Wide Web at <http://www.ietf.org/ids.by.wg/ldup.html>.

The Java LDAP Application Program Interface (Internet Draft). R. Weltman, C. Tomlinson, and S. Sonntag, 2002. Available on the World Wide Web at <http://www.ietf.org/internet-drafts/draft-ietf-ldapext-ldap-java-api-18.txt>.

LDAP Client Update Protocol (Internet Draft). R. Megginson, M. Smith, O. Natkovich, and J. Parham, 2002. Available on the World Wide Web at <http://www.ietf.org/internet-drafts/draft-ietf-ldup-lcup-04.txt>.

LDAP Control Extension for Server Side Sorting of Search Results (RFC 2891). T. Howes, M. Wahl, and A. Anantha, 2000. Available on the World Wide Web at <http://www.ietf.org/rfc/rfc2891.txt>.

The LDAP Data Interchange Format (LDIF)—Technical Specification (RFC 2849). G. Good, 2000. Available on the World Wide Web at <http://www.ietf.org/rfc/rfc2849.txt>.

LDAP Extensions for Scrolling View Browsing of Search Results (Internet Draft). D. Boreham, J. Sermersheim, and A. Kashi, 2002. Available on the World Wide Web at <http://www.ietf.org>.

[org/internet-drafts/draft-ietf-ldapext-ldapv3-vlv-09.txt](http://www.ietf.org/internet-drafts/draft-ietf-ldapext-ldapv3-vlv-09.txt).

LDAP Password Modify Extended Operation (RFC 3062). K. Zeilenga, 2001. Available on the World Wide Web at <http://www.ietf.org/rfc/rfc3062.txt>.

LDAP Proxied Authorization Control (Internet Draft). R. Weltman, 2002. Available on the World Wide Web at <http://www.ietf.org/internet-drafts/draft-weltman-ldapv3-proxy-11.txt>.

Lightweight Directory Access Protocol (v3) (RFC 2251). M. Wahl, T. Howes, and S. Kille, 1998. Available on the World Wide Web at <http://www.ietf.org/rfc/rfc2251.txt>.

Lightweight Directory Access Protocol (v3): Attribute Syntax Definitions (RFC 2252). M. Wahl, A. Coulbeck, T. Howes, and S. Kille, 1998. Available on the World Wide Web at <http://www.ietf.org/rfc/rfc2252.txt>.

Lightweight Directory Access Protocol (v3): Extension for Transport Layer Security (RFC 2830). J. Hodges, R. L. Morgan, and M. Wahl, 2000. Available on the World Wide Web at <http://www.ietf.org/rfc/rfc2830.txt>.

Lightweight Directory Access Protocol (v3): Extensions for Dynamic Directory Services (RFC 2589). Y. Yaacovi, M. Wahl, and T. Genovese, May 1999. Available on the World Wide Web at <http://www.ietf.org/rfc/rfc2589.txt>.

Lightweight Directory Access Protocol (v3): UTF-8 String Representation of Distinguished Names (RFC 2253). M. Wahl, T. Howes, and S. Kille, 1998. Available on the World Wide Web at <http://www.ietf.org/rfc/rfc2253.txt>.

Lightweight Directory Access Protocol (v3) Replication Requirements (RFC 3384). E. Stokes, R. Weiser, R. Moats, and R. Huber, 2002. Available on the World Wide Web at <http://www.ietf.org/rfc/rfc3384.txt>.

Named Subordinate References in Lightweight Directory Access Protocol (LDAP) Directories (RFC 3296). K. Zeilenga, 2002. Available on the World Wide Web at <http://www.ietf.org/rfc/rfc3296.txt>.

OASIS Directory Services Technical Committee (DSML). Information is available on the World Wide Web at <http://www.oasis-open.org/committees/dsml>.

OASIS Security Services Technical Committee (SAML). Information is available on the World Wide Web at <http://www.oasis-open.org/committees/security>.

Password Policy for LDAP Directories (Internet Draft). P. Behera, V. Chu, L. Poitou, and J. Sermersheim, 2001. Available on the World Wide Web at <http://www.mozilla.org/directory/ietf-docs/draft-behera-ldap-password-policy-05.txt>.

Persistent Search: A Simple LDAP Change Notification Mechanism (Internet Draft). T. Smith, G. Good, T. Howes, and R. Weltman, 2002. Available on the World Wide Web at <http://www.mozilla.org/directory/ietf-docs/draft-smith-psearch-ldap-01.txt>.

Simple Authentication and Security Layer (SASL) (RFC 2222). J. Myers, 1997. Available on the World Wide Web at <http://www.ietf.org/rfc/rfc2222.txt>.

Storing Vendor Information in the LDAP root DSE (RFC 3045). M. Meredith, 2001. Available on the World Wide Web at <http://www.ietf.org/rfc/rfc3045.txt>.

The String Representation of LDAP Search Filters (RFC 2254). T. Howes, 1998. Available on the World Wide Web at <http://www.ietf.org/rfc/rfc2254.txt>.

A Summary of the X.500(96) User Schema for Use with LDAPv3 (RFC 2256). M. Wahl, 1997. Available on the World Wide Web at <http://www.ietf.org/rfc/rfc2256.txt>.

The TLS Protocol Version 1.0 (RFC 2246). T. Dierks and C. Allen, 1999. Available on the World Wide Web at <http://www.ietf.org/rfc/rfc2246.txt>.

Use of Language Codes in LDAP (RFC 2596). M. Wahl and T. Howes, May 1999. Available on the World Wide Web at <http://www.ietf.org/rfc/rfc2596.txt>.

Using Digest Authentication as a SASL Mechanism (RFC 2831). P. Leach and C. Newman, 2000. Available on the World Wide Web at <http://www.ietf.org/rfc/rfc2831.txt>.

Looking Ahead

In [Chapter 14](#), Piloting Your Directory Service, we will test the products you have selected by creating a pilot directory service. Depending on what you learn during the pilot, you may be ready to turn your directory into a production service, or you may need to revisit earlier design decisions and product choices. Either way, a pilot service is a great way to bridge the gap between the design and production phases of your directory deployment.

Chapter 14. Piloting Your Directory Service

- A Piloting Road Map
- Piloting Your Directory Service Checklist
- Looking Ahead

One of the most important milestones in your directory's life cycle is achieved when you pilot the service. Until this point your design has existed only in documents, on whiteboards, and in your head. During the pilot stage, you move from paper to programs, from design to deployment, and from theory to practice. You get your first look at how well your design works in something approaching a real environment. You begin to see how well the software performs, how smoothly your data maintenance procedures work, and how receptive the users of your directory are to the service. Then you modify your design according to feedback from the pilot.

Piloting is important for the following reasons:

- No matter how complete a design you believe you have produced, there will always be things you did not think of that show up only when the service is actually running, and piloting can help find these design flaws.
- Piloting is the best way to provide all users of your service an early look at what you're planning. Getting their feedback early and responding to it is key to producing a successful directory deployment.
- Management often likes to see something tangible before committing additional resources. Piloting can provide an important political tool for you to gain continued funding, extra resources, or other benefits for your project.
- Piloting is the best way to determine how well your directory serves the needs of your directory-enabled applications.

In this chapter we present a road map to guide you through the process of piloting your directory service. We focus on the following:

- Prepilot testing, in which you ensure that the directory software you have chosen will perform well in your environment
- Determining the goals and scope of your pilot
- Determining the piloting environment you need
- How to collect feedback on your pilot and how to scale up your pilot to simulate the load that the service will experience in production
- How to analyze the feedback generated from your pilot experience and the implications the feedback has on your design, and how to make changes to the design

A Piloting Road Map

The steps you will follow in establishing your pilot vary depending on your environment and the design of your service. The steps outlined in the following sections are typical and cover the most common scenarios. Don't worry if you need to deviate from these steps, or if you don't have the time or money to cover everything we suggest. Just make sure you cover all the bases that are important to you in your environment.

Prepilot Testing

Before you go to the trouble of setting up a pilot involving users, you should test some aspects of your service in a lab environment. Testing is different from piloting. *Testing* is done in a closed environment, usually just by you and your staff; *piloting* is more open, users are involved, and the scope is expanded. For example, you should do preliminary scale, performance, and functionality testing to select the software on which to run your pilot. These tests are best performed in a laboratory environment, where mistakes are easily corrected and configurations are easily changed.

In [Chapter 13](#), Evaluating Directory Products, we talked about the steps to follow when selecting software for your directory. However, laboratory testing should be used for more than just selecting software. In the laboratory you can find out whether the system works for you. Such testing is a crucial step before you pilot or deploy any significant change to your service. Although successful testing doesn't necessarily mean the change will work for your users in the environment outside the lab, it gives you some confidence that it will.

Laboratory testing is aimed at answering objective questions about the system being tested. Does it do what it's supposed to do? Does it perform within acceptable limits? Can it scale to the required size? Naturally, to answer these questions effectively you need to have appropriate goals in mind for the features you're testing.

Make a list of objective, measurable goals for your system, similar to the one shown in [Table 14.1](#). In this example the component being tested is a new directory server. The objective questions to be determined are whether the new server scales and gives acceptable performance while holding a certain number of entries, serving a predefined number of client connections, and performing a predefined set of queries. The entries, connections, and queries are chosen to reflect the expected typical load that the server will experience in production.

Laboratory testing cannot answer subjective questions about the system being tested. Questions such as, Will users like the system? can be answered only if we ask users during a pilot. Other questions that seem to be objective at first glance also cannot be answered without piloting in a real environment. For example, the interaction between your service and other services on the network, your network topology, various hard-to-predict failure modes, and real user behavior are difficult to produce in a laboratory environment. Piloting helps produce the appropriate conditions to answer these questions.

You should enter the piloting stage only after you have answered some basic questions by testing in the laboratory. Having done so, you can be confident that your pilot will succeed.

Table 14.1. Examples of Objective Criteria to Measure in a Laboratory Testing Environment

Description	Goal	Comments
Number of directory entries	200,000	150,000 people entries; 30,000 group entries; 10,000 organizational unit entries; 10,000 miscellaneous entries.
Number of simultaneous connections	500	Some of these connections may be idle.
Number of simultaneous active connections	50	These connections are all performing the operations listed below.
Response time	Average of less than one second. No client should experience a delay longer than two seconds.	Response time is for clients performing simple equality searches on a single indexed attribute.

Defining Your Goals

What do you want to achieve with your pilot? You will have different goals depending on your directory service type and your environment. How you define your goals leads you to focus your pilot on different aspects of the service. For example, consider the following goals:

- **To produce a directory service for direct use by many demanding users.** In this case you might focus your pilot on the user experience. This means spending extra time measuring end-user response time, designing user interfaces, involving human factor expertise, piloting with a large and diverse user community, and conducting focus groups. You should measure your success by how much users like your service, how efficiently the service answers their queries, and how completely it serves their needs.
- **To produce a directory service for use by application developers.** In this case your emphasis should be on the interfaces by which application developers access the directory. Measure your success by how easy the system is to use, how quickly new applications can be developed, and how much functionality the system provides. More information on LDAP-enabled applications can be found in [Chapters 21](#), [Developing New Applications](#), and [22](#), [Directory-Enabling Existing Applications](#).
- **To produce a directory service containing sensitive data that serves the authentication and security needs of applications.** If this is your goal, you should focus your pilot on security. This means completing a security analysis to ensure that the security measures protecting your directory are adequate and easy to use. You should measure your success by the security (both perceived and actual) that users of the system are afforded, the ease with which applications can use the security services provided by the directory, and the degree to which the security needs of all applications are covered. [Chapter 12](#), [Privacy and Security Design](#), discussed this topic in detail.

Other potential areas of focus exist, but it's important to realize that you cannot fully pilot every aspect of your service. If you could, you wouldn't have a pilot; you'd have a full-fledged directory service! Try to choose representative aspects of the service that validate your design and pilot those. When you finish your pilot, you should have a high degree of confidence in your overall approach, and you should be confident that any loose ends that you didn't address in your pilot are not going to be showstoppers.

Defining Your Scope and Time Line

The goals you want to achieve by piloting will help you define the scope of your pilot. Pilot scope has several possible dimensions, including the following:

- How much will end users be involved in your pilot? Will your group of users be small and focused or large and diverse?
- What aspects of your service will you pilot? Will you try to pilot a few aspects thoroughly, or will you cover all aspects in less depth?
- Will the pilot have the same number of entries as the planned production service?
- Will the pilot attempt to simulate the client load anticipated for the production service?
- Will you pilot all of the applications planned for the production service, or a subset of them?

You will determine part of your scope by the time and resources you have to devote to your pilot. External constraints may be placed on you, or you may place constraints on yourself. A successful pilot is focused and has clear goals and objectives. Try to avoid endlessly piloting with no way of knowing when you're done. Your pilot may end because of either success or failure, and it's important to be able to recognize both outcomes. In the case of a successful pilot, your next step may be full deployment of the service. In the case of a failed pilot, your next step is probably to redesign, retest, and repilot.

A good practice is to draw a time line showing the major milestones in your pilot. This time line serves two purposes. First, it helps you map out the stages of your pilot, which helps you decide what needs to happen when. Second, it gives you a good reality check on the pilot itself. If your time line leaves only a week for locating, training, and getting feedback from your pilot users, you know that you haven't budgeted enough time.

The sample time line in [Table 14.2](#) includes some time for testing, locating pilot users, rolling out the pilot, gathering feedback, and even applying that feedback to the pilot itself. This time line covers just over 12 weeks—an aggressive schedule. Unless you have very dedicated and motivated pilot users, don't expect to be able to do things this quickly.

Remember that the purpose of restricting the scope of your directory pilot is to ensure that it happens in a reasonable amount of time and with a reasonable amount of resources. Be as explicit as you can about your scope; avoid extending the pilot into areas that are beyond it.

Developing Documentation and Training Materials

Your pilot may involve users who are not familiar with the service being piloted. Documentation, training materials, and other information can help prepare your pilot users to be effective participants. You might be able to revise these materials and give them to your production users, so it's important to pilot these materials along with the directory service itself.

Table 14.2. A Sample Pilot Time Line

Task	Start Date	Duration
Laboratory testing	+0 weeks	2 weeks
Locating pilot users	+0 weeks	2 weeks
Pilot environment setup and rollout	+2 weeks	1 week
Pilot operation	+3 weeks	4 weeks
Data gathering	+4 weeks	3 weeks
Incorporation of pilot feedback	+7 weeks	2 weeks
Revised pilot environment setup and rollout	+9 weeks	1 week
Revised pilot operation	+10 weeks	2 weeks
Data gathering	+10 weeks	2 weeks
Incorporation of pilot feedback into design	+12 weeks	1 week

There are at least three broad categories of users you may need to address:

1. **End users.** If your service exposes end users directly to your directory service, you should provide them with documentation and training materials. End-user documentation is often tutorial. You cannot assume that your users know much about your service or directories in general. You must educate them if you expect them to use the system and be effective pilot participants.

On the other hand, if your directory service is not directly accessible to end users (for example, if the directory is used to provide authentication and personalization services for a Web-based portal), then it's likely that any required end-user documentation will be provided by the portal designers.

2. **Administrators.** Administrative users typically require a different kind of documentation. There are three types of administrators: directory system administrators, directory content administrators, and directory-enabled application administrators. Document the procedures they follow, provide troubleshooting guides for when things go wrong, and train them in the use of the system. Try not to cut corners on documentation for your administrative users; they are responsible in large

part for making the system run smoothly.

3. **Application developers.** Application developers are often the most sophisticated users of your directory. They also require the most extensive training and documentation materials. Application developers usually need to know everything users need to know, but they also have to understand how to access the directory from their application. Furthermore, they need to know about your directory's naming conventions, available schemas, how to access the directory through an API, and more. You can usually count on developers to be willing to tolerate rougher edges than users, but do not underestimate the amount of information they need to do their job.

Selecting Your Users

Selecting your users is important, especially if your service is targeted at end users (as opposed to a small set of applications you control). The users of your directory service are the least predictable variable in your directory equation. Technical problems, such as inadequate capacity, can be solved relatively easily; problems involving user perceptions and expectations can be much harder to solve.

If you do a good job of selecting pilot users, you will have a representative sample of your ultimate directory service user community. Making your pilot users happy translates directly into making your production users happy. No system is perfect, of course, but choosing your pilot users wisely goes a long way toward ensuring a successful directory deployment.

If you do a poor job of selecting pilot users, on the other hand, you will not have a representative sample of your ultimate directory service user community. Making your pilot users happy may then have no relation to the happiness of your production users. From a user perspective, you might as well have not piloted your directory service at all.

How do you select a good set of pilot users? There is no foolproof method, but here are some guidelines:

- **Know who the users of your production service are going to be.** It's important to know the ultimate audience of your directory service. If you don't know this, there is little chance you will select a representative group of pilot users. Be explicit about this. Write down the types of people who you expect to use your directory.
- **Pick your users; don't let your users pick you.** You may be tempted to ask for volunteers to pilot your directory service. Although this is fine in some environments, volunteers are a self-selecting group that tends to be more outgoing, more comfortable with computers, and usually more experienced than the general user population. As such, they often make poor representatives of your user community. On the other hand, if your pilot goals are focused on testing the system components of your directory more than perfecting the user experience, a self-selecting group of users might work just fine. In fact, the extra sophistication and experience these users bring to the pilot may even be an advantage in giving the system a better workout.

If your goal is accurate representation, a better approach may be to recruit users from each group in your organization. This way you can be sure to get appropriate representation from all important constituencies. Also be sure to include users with varying degrees of sophistication. This approach may be difficult. Be prepared to offer some kind of incentive to your pilot participants, such as cash, T-shirts, a free lunch, or another perk. If your pilot offers real advantages to users (for example, better performance), be sure to explain that to your potential pilot users.

Another good approach is to use a combination of volunteers and handpicked users. The volunteers are easy to get, perhaps by advertisement on the Web. Handpicked users ensure that good representation is maintained. You might accomplish a good balance by using a staged approach: Use volunteers first to work out the early bugs, and then use representative users to make sure the system works for your community.

- **Make your expectations clear.** It's important to tell your pilot users what you expect from them and what they can expect from you. Making your expectations clear can help weed out inappropriate pilot users who are not prepared to contribute to the pilot. It also helps users prepare themselves for the pilot and budget their time.

Explaining to your pilot users what they can expect from you also helps avoid mismatched expectations. This applies to any remuneration they will receive for participating in the pilot, the level of support you can provide to them, the quality of service they can expect, and how their feedback will be incorporated.

- **Don't forget administrators.** Piloting is not solely about end users; you also have administrative procedures to test. Don't forget this important aspect of your service or the important administrative users who perform these procedures. Administrative procedures that you might want to pilot include data maintenance, exception handling, data backup and restoration, and recovery from disasters. Be sure to factor these procedures and the corresponding users into your plans.

After you've selected an appropriate number of pilot users, there are some steps you should follow before, during, and after the pilot process. The following list is a minimal set of things to accomplish:

- **Prepare your users.** Make sure the users you select have the tools and training necessary to be effective pilot users. If the goal of your pilot is to see how users with no training cope with your directory, no training is necessary. On the other hand, if your goal is to exercise the system and ensure that a wide range of experienced users are served, make sure you provide any necessary training.
- **Be responsive.** It's important to be responsive to your pilot users. After all, they are going out of their way to help you make your service better. Be as responsive as you can to their needs, by answering questions, responding to feedback, providing support, and so on.
- **Provide feedback.** Your pilot users should feel a sense of ownership, or at least knowledge, of your directory service. One good way to do this is to provide them with constant feedback. If the pilot encounters problems, explain what went wrong. If you make changes to the service, explain what you did. If you gather statistics on the system during the pilot, share them with your users. All these steps will make your users feel more involved in the pilot and more likely to provide good feedback.

One way to respond to users is to create a mailing list or discussion group containing everyone involved in the pilot. You can send regular status reports, notification of exceptional conditions, and other information using this list, and you'll know that they will reach all pilot participants.

All of these steps can help you develop a successful relationship with your pilot users, which is important if you expect to conduct pilot activities in the future.

Setting Up Your Environment

At the same time that you select your user population, you should set up the environment for your pilot. You want things ready to go as soon as your users are identified. Remember, you are not piloting just to see whether users like the service; you are also piloting to test all the procedures you've designed for creating and maintaining the service and its content. In addition, you are piloting to see whether the system works efficiently as a whole.

The multiple purposes of piloting make your choice of pilot environment even more important. Procedures that work well in one environment may not work at all in another. For example, suppose you rely on a local disk for your directory database during the pilot, but the production service needs to run over Sun's Network File System (NFS). The product you select may not work over NFS, and even if it does, performance may not be acceptable. Similar concerns can arise with your networking, hardware, and software.

The kind of environment you end up with for your pilot depends on the kind of environment you will have in production and the resources you have to duplicate it. Ideally, you will set up a pilot environment that exactly duplicates your production environment. In this way you minimize problems resulting from environment changes from pilot to production. Practical considerations, however, often will force you to create a pilot environment that doesn't match what will be used in production. The differences may run the gamut from using bits and pieces of leftover equipment to using less expensive versions of all your production machines. If you find yourself having to scrimp in this kind of situation, your pilot can still be effective, but you will need to be prepared to deal with the uncertainty that this situation will bring. For example, you may need to extrapolate to determine the capacity of your production environment on the basis of performance data collected from a much smaller pilot environment.

Tip

When your pilot is concluded, keep the equipment used during the pilot as a test bed. As you make changes to your service, you can pilot them on your test bed hardware. The test bed provides a convenient staging service for improvements to your directory service.

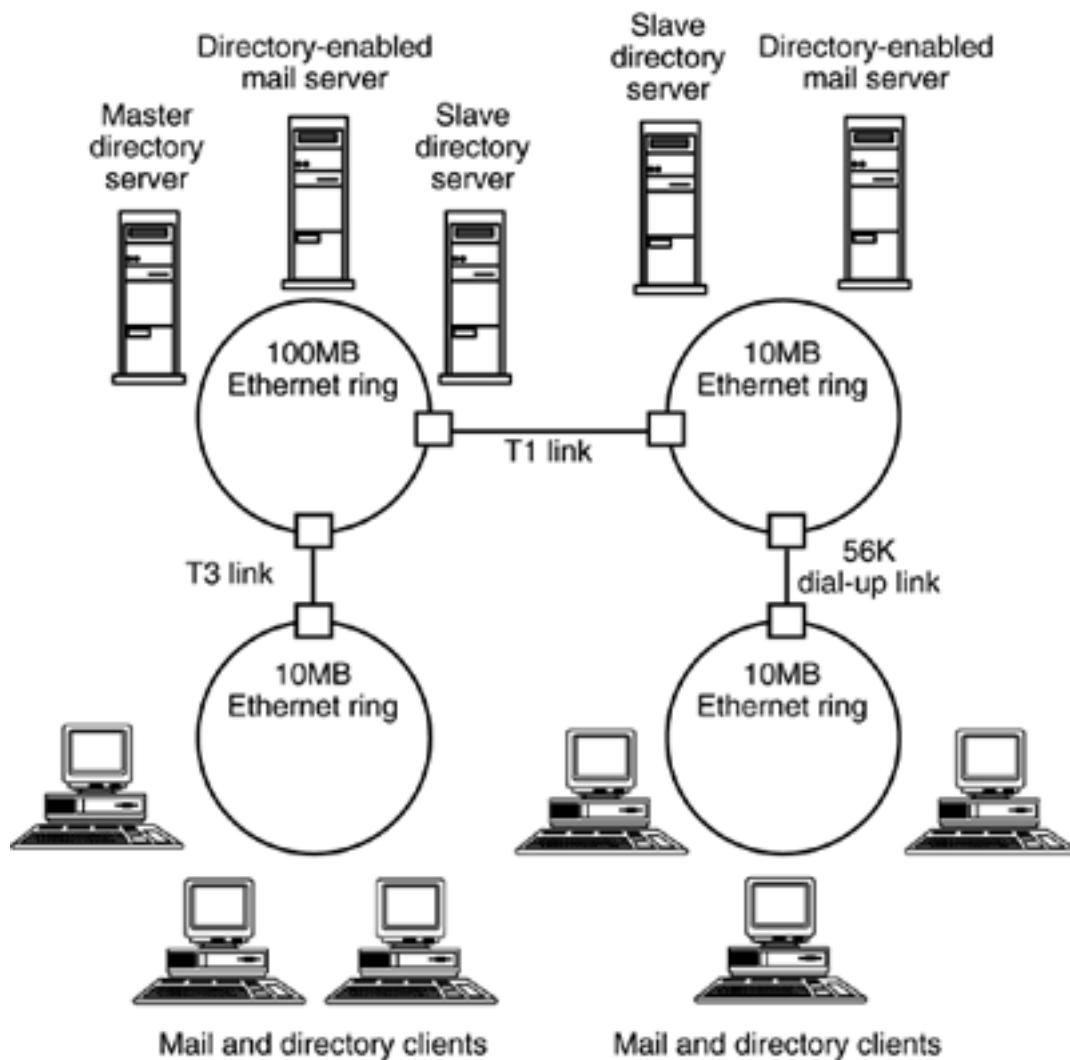
Whatever equipment you have at your disposal, keep the following advice in mind when designing your pilot environment:

- **Software versions.** Use the same operating system versions (including patches or service packs) on your pilot machines that you will use on your production machines. The same guideline applies to backup software, third-party software, and the directory server software itself.
- **Hardware configuration.** Try to use similar hardware configurations in both the pilot and production environments, including such things as the type of processor, number of processors, type of disk drive, type of backup device, network controller, and other hardware. Some things are more likely to cause problems than others are. For example, moving to a multiple-CPU system in production might create problems not encountered on a single-CPU system in your pilot.
- **Network configuration.** Try to ensure that the network configuration of your pilot servers and clients is similar to the production's network configuration, including things such as the available bandwidth, the topology of the network, the amount of other traffic on the network, and the reliability of the network links. Your pilot system may be snappy, but the production system could seem very slow because of too

much traffic on the production network or a different topology creating longer network latency.

It's a good idea to make a map of your pilot system. Identify its major components and the network links between them. Label the map with the hardware, software, and type and speed of network links at each component. Identify links between replicas and the role each replica serves. Compare this to the similar map you have made for your production system. Look for any obvious differences, especially in the trouble areas just mentioned. An example of a pilot environment map is shown in [Figure 14.1](#).

Figure 14.1. A Sample Pilot Environment Map



After you've designed your pilot environment, you need to build it. Do this in plenty of time to fix problems before the pilot's official start date. As with your production system, some problems will undoubtedly crop up only in the implementation phase. Leave yourself enough time to fix any glitches that arise.

Rolling Out the Pilot

There are several steps to rolling out your pilot. The steps you use may vary somewhat depending on how large your pilot is and the type of users involved. The most common steps are outlined here:

1. Bring up the servers.

2. Test the servers.
3. Put system administrator feedback mechanisms in place.
4. Roll out documentation, training, and clients to system administrators.
5. Put end-user feedback mechanisms in place.
6. Begin to distribute software to end users, if required.
7. Begin to distribute documentation and perform training.
8. Get early feedback.
9. Widen the distribution of the pilot.

We recommend rolling out the pilot to system administrators before end users. System administrators, who are relatively few in number—and with whom you probably already have a good working relationship—should go first. If things go smoothly for them, begin rolling out the pilot to a small group of end users. If things go well for them, expand the scope of the pilot with other end users.

Make sure that your feedback mechanisms are in place before rolling out each stage of the pilot. If you don't, you may lose important feedback and, more importantly, your pilot users' confidence. Also be sure to roll out documentation and training as you roll out the pilot to users. Failure to provide this type of assistance can confuse users, giving them a bad perception of the pilot.

Collecting Feedback

With your pilot up and running, you need to begin collecting feedback. Keep in mind that this is the whole point of your pilot, so this step is important. There are several different kinds of feedback you can collect:

- **User feedback.** User feedback from your pilot is your one chance to get an early look at what your users think. Then you can modify your system accordingly.
- **System administrator feedback.** System administrator feedback comes from your pilot users administering the directory and its content. Collecting and acting on this feedback will have a profound positive influence on the maintainability and reliability of your directory.
- **System feedback.** During your pilot, monitor the performance of your servers, your network, and any other relevant system parameters. Collecting and incorporating this feedback into your production system will make it run more smoothly and perform better.
- **Problem reports.** Collect all the failure and problem reports you receive during the pilot. It's a good idea to save these reports and analyze them after the pilot, looking for trends that you might miss if you were analyzing only one problem at a time.

You can use various methods to collect feedback, depending on the type. To collect user

feedback, you might use the following methods:

- **Interviews.** Interviewing users can provide very effective feedback. If conducted by someone with experience, interviews can afford much more effective feedback than practically any other method. The downside of this approach is that it is labor intensive. For an interview to be effective, it needs to be conducted one-on-one. This can be a time-consuming process, but it's an excellent idea.
- **Focus groups.** In this approach, small groups of users are interviewed about the pilot. Focus groups have many of the same advantages as one-on-one interviews, with less cost. A great deal of expertise is still needed to conduct effective interviews, but the total time is reduced by the size of the focus group. However, finding a convenient time to schedule focus group sessions can be difficult.
- **Online comments.** Setting up an online service through which users can respond is another effective feedback mechanism. You need to find a balance between allowing users to comment in a completely free-form manner and restricting feedback with a mechanism such as a multiple-choice comment form. The free-form mechanism allows maximum flexibility, but users are often unclear in their comments if left to their own devices; be prepared to follow up on these unclear comments. The restricted mechanism produces results that are easy to parse and interpret, but you must ask the right questions and provide the correct choices—a difficult thing to get right.

A good approach may be to use a combination of methods. You want to get good feedback but avoid spending too much time developing fancy feedback mechanisms. Most of what you want can usually be achieved through a simple e-mail collection mechanism.

You can use the same techniques for collecting administrator and developer feedback that you do for user feedback. Out of all the techniques, with administrators and developers you should probably do one-on-one interviews; because there are relatively few of them, this technique may be more feasible for these groups.

Tip

After you've solicited feedback from your pilot participants and acted on it, give them a summary of the feedback they provided and the actions you took. Such a summary helps your pilot users know that their participation was worthwhile, and it helps you obtain willing participants for future pilots.

When you're collecting system feedback, the techniques you use are quite different from those used to collect user feedback. Most of the techniques involve collecting data from your automated monitoring sources. [Chapter 19](#), Monitoring, discusses monitoring of your directory in more detail, but some of the more common and useful techniques are listed briefly here:

- **Operating system monitoring.** Use whatever tools your operating system provides to measure the general performance of your servers, including things like disk activity, paging activity, memory usage, system calls, and the ratio between user time and system time on the CPU.

This kind of monitoring is aimed at identifying system bottlenecks. For example, if you notice an inordinate amount of activity on one disk, you might consider switching

some files with high write traffic to a second disk on your system. Doing so distributes the write traffic more evenly, reducing the bottleneck. As another example, if you notice a lot of paging activity, you might either buy more memory for your machine or tune parameters on your software to make it use less memory.

- **Directory software monitoring.** This technique involves using the directory's own monitoring and auditing capabilities to determine how smoothly the system is operating. Such capabilities include creating directory error and access log files, monitoring capabilities through Simple Network Management Protocol (SNMP) or via LDAP (for example, using Netscape Directory Server's `cn=monitor` entry), and providing any other information that the software obtains.
- **Directory performance monitoring.** This technique involves directly monitoring the performance of the directory system by using scripts or other tools. The objective of collecting this kind of data is to understand the performance of the directory system itself—the performance that end users experience. Important performance indicators are low average response times with a small standard deviation. Make sure you measure response time to typical queries so that the data you collect is meaningful.
- **Directory-enabled application monitoring.** This technique involves monitoring the performance of directory-enabled applications that depend on the directory for their own performance. Depending on the focus of your service, such feedback can be very important. For example, if your directory serves the needs of a directory-enabled e-mail delivery service, you'll want to know how well that service is functioning and whether the directory service is a bottleneck.

When collecting data, be honest with yourself about what you know (because you measured it) and what you don't know. Your hunches about what's good and what's bad about the pilot may be valid, but there's no substitute for hard data. If the conclusion of your pilot is that you need to increase your budget, for example, having objective data to back up the conclusion is especially valuable.

Scaling Up the Pilot

By definition, your pilot is conducted on a smaller scale than your production service is. Of course, not every aspect of your pilot is necessarily scaled down from the production version. For example, your pilot may involve only a few users, but the pilot directory servers might contain as much data as the production servers do. Be careful to keep this in mind as you interpret data from your pilot.

You should also try to find ways to scale up selected portions of your pilot to increase your confidence in how the system will scale in production. You can scale up your pilot in several areas:

- Number of entries
- Size of entries
- Number of connections
- Number of queries
- Number of replicas

Some of these dimensions can be tested in the laboratory. For example, you can test the number of entries and connections that a single server can handle. However, it's important to scale up some aspects of the service during the pilot while you have users using the service. Sometimes the interaction among several factors may combine to produce unexpected results. The more you can simulate these real-world interactions, the more realistic your scaling tests will be.

You can use many techniques to conduct realistic laboratory scaling tests. First formulate a model describing the kinds of loads and conditions you want to test, and then develop test clients that simulate these loads. Each test client may make many connections to the directory, simulating many real-world clients. Develop test data that increases the size of your directory. You can increase the number of entries along with their size, the number of values in each attribute, and so on (these don't have to be real entries). Think about your future needs in each area, and focus your testing on areas you think you'll need.

Introduce other factors into the system. For example, load the network links between clients and servers with other traffic. You might do this by writing a special-purpose client or by simply transferring some large files back and forth during your test runs. Most systems have good tools you can use to induce different kinds of loads. For example, spray is a good tool for loading the network, and mkfile is a good tool for loading the file system. Load the directory server machines with other processes to see whether the machine can be shared with other services.

Simulate network and hardware failures during the test by unplugging network cables and power cords. How does the system react? Do clients fail over to directory replicas? Is the directory server able to recover its database? Does replication recover gracefully? These questions are important to answer in any kind of environment, but they become more crucial in a large-scale directory environment. For example, if your directory does not recover gracefully from a power outage, you may have to rebuild the database from scratch. Rebuilding from scratch may be tolerable on a small scale, but for a big directory, it can introduce unacceptable downtime.

Watch out for scaling behavior in which the directory system does not degrade gracefully. This kind of "brick wall," when met in a production environment, invariably comes as an unpleasant surprise. For example, consider a directory that can hold only a fixed number of entries or size of database: When these limits are reached, the directory ceases to function. Look for directory software that degrades gracefully as limits are reached.

Finally, make a note of any performance problems you encounter during your pilot and are unable to explain. For example, if you notice that performance of the directory degrades and then corrects itself, save the server logs and note the date, time, and any exceptional conditions you are aware of. It's a good bet that the problem will recur later (possibly after you've gone into the production phase), and the additional data you collect in the testing phase can be helpful in tracking down the underlying cause.

Applying What You've Learned

Applying what you've learned is the most important aspect of your pilot. After all, the whole point of doing the pilot is to learn how well your design works in practice. Naturally, if you learn of flaws in your design, you should make changes to correct them.

This is especially important during your directory's pilot stage. You will get feedback from your pilot that will change your design. Be prepared to incorporate these changes into the pilot itself, providing a feedback loop to let you know when you get things right.

In many areas you will receive feedback that you should incorporate. Here are some of the more important topics to listen for:

- **User experience.** The directory experience of your users is perhaps the most subjective design criterion your directory service has. Therefore, it is the most important to validate through continuous refinement and user feedback.

- **Operating system configuration.** If you find a problem with your operating system configuration, do your best to correct it during the pilot. Don't assume that upgrading to a newer version of the operating system will fix the problem you experienced; upgrading may work, but it could also introduce new problems. Be aware of the effect of configuration changes on your directory software.
- **Directory software configuration.** You may need to change the configuration of your directory server software. For example, you may need to tune the software's database parameters to provide better performance. There are always trade-offs, however. Adding an attribute index will speed up searches on the indexed attribute, but it will increase the size of the database and reduce the speed of updates.
- **Directory-enabled application configuration.** One or more of the directory-enabled application clients may need to be reconfigured or even recoded. For example, the application may be making inefficient use of the directory by opening a new connection each time it wants to do a query instead of reusing an open connection. Or the application may be using several searches when one would suffice. [Chapter 21](#), Developing New Applications, describes various techniques that application developers can use to make their applications play nice with the directory.
- **Hardware configurations.** You may need to upgrade your hardware because of capacity or other problems. Be sure to pilot with the upgraded hardware; it may introduce other problems, or it may not solve the problem you thought it was solving. Use your laboratory and your pilot to experiment with different hardware configurations and combinations of hardware and software.
- **Network configuration.** The network topology of your directory servers may be inadequate. For example, you may find when you do scale testing that you need to move your servers to a high-speed network.
- **Server topology.** Your server topology may be inadequate. For example, you may decide that you need a replica in each of your branch offices because your WAN links are not reliable enough. Or you may find that the distribution topology you planned leads to poor performance. In this case you may need to redesign your topology so that the data is closer to the clients that need it. Be sure to pilot these changes, and make sure your clients are configured to take advantage of the new topology.

It's important to incorporate as much feedback as possible and then repilot the service with the design changed accordingly, but you also need to be realistic. You will not be able to incorporate all the feedback you receive. Some feedback is so trivial that there is no need to repilot. Some of it will be bad advice. Some of it will not be practical. Some pieces of feedback may conflict with others. For each piece of feedback you receive, decide if it is important enough to try to incorporate into the pilot, or if it can safely be resolved during the transition to production. At the end of your pilot, each issue should have been addressed, or a plan of action for addressing that issue should exist.

Piloting Your Directory Service Checklist

To review, here are some important things to remember when conducting your pilot:

- Select test criteria for scale, performance, and functionality.
- Perform laboratory tests for scale, performance, and functionality.
- Document pilot goals.
- Document a pilot scope.
- Document a pilot time line.
- Develop pilot documentation and training materials.
- Identify pilot end users and administrative users.
- Set up the pilot environment.
- Roll out the pilot to users and administrators.
- Collect feedback.
- Incorporate feedback.
- Repeat the last two steps until you feel confident that you are ready to put your directory service into production.

Looking Ahead

After you've completed your pilot, it's time to roll out your service in production. Before we discuss that topic in [Chapter 16](#), Putting Your Directory Service into Production, we take a slight detour in [Chapter 15](#), Analyzing and Reducing Costs, to look at the costs involved in providing your production directory service.

Chapter 15. Analyzing and Reducing Costs

- The Politics of Costs
- Reducing Costs
- Design, Piloting, and Deployment Costs
- Ongoing Costs of Providing Your Directory Service
- Analyzing and Reducing Costs Checklist
- Further Reading
- Looking Ahead

To design, develop, and deploy a successful directory service, you need to have the proper funding. Getting funding from your organization invariably means understanding the costs of the service you plan to deploy, developing a budget, and communicating that information to those who hold the purse strings. Because budgets are often subject to negotiation, it's also wise to prioritize the items in your directory service budget so that you know which aspects of your budget are negotiable and which are not.

In this chapter we provide an overview of the various costs associated with the phases of a directory service's life cycle. For the purposes of this chapter, we break the life cycle into two major sections: the design, pilot, and deployment phase (everything preceding going to production); and the maintenance and upgrade phase (everything after production service begins).

This chapter isn't meant to be a comprehensive guide to project planning and budgeting; that topic is covered adequately in other texts (see the Further Reading section at the end of this chapter). Instead, we focus on directory-specific costs that you need to consider when planning and budgeting. We also describe ways to reduce costs during your directory's life cycle. Your company may have a well-established planning and budgeting process in place. If that is the case, use the information in this chapter to assist you as you work with the existing process.

This chapter is necessarily incomplete. Each directory deployment is unique and targeted at meeting specific needs. Your directory may not incur some of the costs described in this chapter, and it may incur some costs that we haven't thought of. Use this chapter as a guide to the general types of costs associated with providing a directory service, but be flexible as you think about your particular directory service and its budget.

The Politics of Costs

At some point in your directory service's life cycle, you may be asked to justify its costs, or you may need additional funding to accommodate growth or new applications. To communicate these needs effectively, you must explain costs clearly and make a convincing case for your needs.

When making your case, focus on the benefits that the directory can provide. Translate these benefits into cost savings for the organization, indicating how they represent a return on the company's investment in directory technology. For example, if the deployment of your directory and consolidation of LAN e-mail packages have allowed you to reduce the time spent managing electronic mail accounts from 20 to 2 hours a week, your organization is enjoying a concrete savings. There will no doubt be many of these types of cost savings. Collect them all in one document that clearly shows the total savings. In some cases the directory will pay for itself from the beginning. In other cases, it may be too early in the directory's life cycle for all costs to have been recovered, but you can calculate the date when they will be.

To make your case even stronger, be sure to include revenue that is both directly and indirectly generated because of the directory. For example, if the directory was leveraged to develop a new application that has spawned a new revenue stream, be sure to allocate a portion of that revenue to the directory itself.

Show how your directory service is being used and becoming popular among your users. You might maintain some type of usage data that shows the growth of your service; you can then present the data as evidence that the directory is providing a useful service. If these usage figures are increasing, they make an excellent case for additional funds to cover the cost of upgrading the service to provide additional capacity.

With the information provided in this chapter, you can make a case that you've deployed your directory service in the most cost-effective manner possible.

Reducing Costs

In today's competitive business environment, efficient processes are vital to an organization's bottom line. Making your directory more efficient makes the business processes that depend on it more efficient. Understanding your directory service's costs is the first step toward making it as cost-effective as possible. In this section we'll outline some general cost reduction principles as they apply to directory services. In the next section—Design, Piloting, and Deployment Costs—we'll provide specific suggestions for reducing the costs at each point in the directory service life cycle.

General Principles of Cost Reduction

Keep in mind four general principles when you're striving to reduce the costs of your directory service:

1. You need to understand the level of related expertise that your staff possesses. For example, if several of your staff members have direct experience with directory technology and can train the remaining staff members, your support and training costs will be low. If you do not have the necessary expertise in-house, you will need to either develop it or use an external consulting organization.
2. Hardware and software costs are generally lower than personnel costs. A prudent hardware or software investment can allow your staff to work more efficiently.
3. Automation of manual processes can save time and allows your deployment and maintenance staffs to concentrate on increasing the reliability and availability of the directory. It also allows new applications to be developed more rapidly. For example, moving from a manual, attended backup process to one that occurs automatically frees up a person's time to do something more useful.
4. Your maintenance and deployment staffs are your best resource for identifying opportunities to improve efficiency. They work with the directory each day and routinely see where the system is less efficient than it should be. Involve them in the planning process and solicit their input for ways to reduce the costs of running the directory service.

Design, Piloting, and Deployment Costs

The first half of your directory's life cycle includes all the activity up to the point at which you throw the switch and put your directory into production. The costs associated with this phase can be broken down into three major areas: design costs, piloting costs, and deployment costs.

Design Costs

As discussed in [Part II](#), Designing Your Directory Service, the design phase of your directory's life cycle is the time when you make many important decisions concerning the scope and structure of your directory service. Costs associated with this phase involve primarily staff but include all of the following:

- **Salary and benefits costs.** Costs in this category include salary and benefits for you and the staff helping you during the design phase. If your design phase involves people from other groups, such as your Human Resources department, be sure to include salary and benefits costs for them. To quantify these costs, you will invariably have to apportion staff time among the various phases of the directory life cycle. Also be sure to budget time for the deployers to become familiar with the software so that they fully understand its capabilities and limitations.
- **Fees paid to outside contractors.** If you retain the services of consultants to assist you in the design phase of your directory project, include the fees paid to them in your total design phase costs. As with salaries and benefits, if you employ the services of consultants beyond the design phase, you may need to assign portions of the costs to each phase of the project.
- **Research material costs.** As you design your directory service, you will probably purchase journal subscriptions, special technology reports, and books like this one to assist you in making good design choices.
- **Software costs.** During the design phase you may need to purchase copies of the software you plan to deploy. In some instances, you may be able to avoid these costs by using evaluation copies of the software.
- **Training and conference costs.** You will need to develop in-house directory expertise; therefore, you may incur costs when you send staff members to training seminars and technology conferences.

Reducing Design Costs

In general, be careful about cutting costs in the design phase of your directory's life cycle. Studies have repeatedly shown that mistakes made during the early part of a project are significantly more expensive to correct than mistakes made later in the project. If in doubt, look for other expenses to reduce.

Consider the relative efficiency of attempting your directory design process solely using in-house expertise versus involving outside consultants. Evaluate the directory-specific knowledge of your staff who will design the directory. Are they familiar with directory technology, or will significant training costs be incurred before design can begin? If directory technology is not something your staff is familiar with, it may be better to bring in consultants who can help you with the design process.

Piloting Costs

During the piloting phase of your directory's life cycle, you obtain valuable information about your directory design. You learn whether your design assumptions are valid, and if they're not, you have the opportunity to revisit your design and make improvements. As in the design phase, reducing costs in the piloting phase is risky; if you don't obtain useful feedback, your design may not be optimal. When you're reducing costs in this phase, be sure that you do not compromise your ability to obtain and respond to feedback from your pilot users.

The following are common costs associated with piloting:

- **Equipment and software costs.** Your pilot phase, if done properly, includes an actual small-scale directory deployment. The equipment used to provide this pilot deployment might include CPUs, disks, networking hardware, and software.
- **Staff costs for deployment.** During your pilot phase, you will incur costs associated with installing, configuring, and maintaining the pilot hardware and software.
- **Staff costs for publicity, feedback, and analysis.** During and after the pilot phase, you will incur costs associated with publicizing your pilot, gathering feedback from pilot users, analyzing that feedback, and refining the directory design and deployment plans.
- **Documentation and training costs.** Both vendor and in-house documentation should be available to your users during the pilot phase. You may also find it helpful to your pilot's success to provide training sessions for your users.
- **Incentives for pilot users.** To obtain the best possible feedback from your directory pilot, you might choose to provide incentives to reward pilot users for their participation and feedback.

Reducing Piloting Costs

The piloting phase of your directory life cycle is your opportunity to validate your directory design and anticipate potential problems. If your pilot program doesn't give you the information required for a successful deployment, you need to either do another pilot or proceed without it. Neither outcome is desirable, of course.

To reduce piloting costs, pick your pilot participants carefully, explain what you expect from them, and tell them how they will benefit from their participation. If your pilot participants generate useful feedback that helps you validate your basic assumptions and improves the service before it goes into production, your piloting costs represent money well spent.

You can also reduce piloting costs by reusing the pilot hardware when you bring your production directory service online. If your production service will be rolled out in an incremental fashion, you can bring up the initial production service, then dismantle the pilot system and redeploy it into the production environment.

Deployment Hardware Costs

During the deployment phase of your directory service, you will purchase the equipment required to provide the full directory service you've been planning. Deployment hardware costs include the following:

- **Server costs.** This category includes the price of server hardware used to run the directory. Be sure to include the costs of standby units you may purchase as hot or cold spares.
- **Memory upgrades.** If you purchase your memory preinstalled from the same vendor who supplies your servers, memory costs will already be reflected in the price of your servers. Otherwise, include memory costs in your total deployment costs. Be aware that some server hardware vendors (in particular, those who sell high-end systems) will not honor the factory warranty if third-party options such as memory, disk drives, or network adapters have been installed.
- **Network hardware.** Included in this category are network adapter cards, cabling, hubs, routers, terminal servers for remote maintenance, and other equipment. Some of this equipment may already be deployed in support of other services. You may want to apportion costs among the various services connected to the directory. For more information, see the sidebar Apportioning Hardware and Software Costs later in this chapter.
- **Hardware used for monitoring functions.** A network management system (NMS) may already exist within your organization, in which case you may need to allocate a portion of your budget to maintenance and upgrades of the existing NMS hardware. If no NMS exists and you plan to monitor your directory service, you may need to purchase a system for NMS functions.
- **High-availability solutions.** If your directory must be highly available, you may need to purchase a hardware solution that provides automatic failover across a set of replicated directory servers. Some directory solutions support failover via proprietary technology. However, no current LDAP standard defines how failover is provided. Be aware that high-availability solutions are often provided as part of a storage solution, discussed next.
- **Mass-storage devices.** Storage devices for your directory data might include disk drives, redundant array of inexpensive disks (RAID) storage units, or more advanced options, including network-attached storage (NAS) and storage area network (SAN) solutions. High-end NAS and SAN systems are often shared among several services.
- **Backup solutions.** Directory data must be backed up to protect against disk failures and disasters such as fires and floods. Backup solutions might include digital linear tape (DLT) or linear tape-open (LTO) units and management software. You may also choose to back up to a mirrored disk drive or use replication to keep a backup server up-to-date with your directory data. If you use a high-end NAS or SAN solution, it will frequently provide an integrated backup management solution, either as a standard feature or as an added-cost option.
- **Physical plant.** You need a place to locate the equipment that provides your directory service. You will almost certainly locate it with other computing infrastructure services; therefore, you may need to allocate a portion of your budget to the rental costs and upkeep of this facility.

Reducing Deployment Hardware Costs

There are several ways to reduce the cost of hardware used to provide your directory

service. First deploy your directory service on reliable hardware that scales well. This means using server and network hardware that has sufficient capacity for your present needs, as well as enough extra capacity to accommodate six months to a year of anticipated growth. Also be sure that the hardware can be upgraded after that time. When evaluating hardware vendors, consider the following questions:

- What is the maximum RAM capacity of the server? Is it sufficient for your current and future needs?
- Does the hardware permit the required level of reliability and redundancy? For example, are any necessary hot-swappable disks and power supplies available?
- What mass storage options are available? Are they sufficient for the size of directory you plan to deploy? If you use NAS or SAN solutions, can you share the costs of these among several projects?
- Do the hardware and operating system (OS) support multiple CPUs? Can additional CPUs be added easily? Does the directory server software take advantage of multiple CPUs?
- Does the operating system for this hardware limit you in an undesirable way? For example, can the operating system support large files (greater than 2GB)? Can it address a large amount of RAM? Can it handle enough network connections to carry the load?
- Are the costs of maintaining the operating system reasonable? Do you have sufficient in-house expertise on the OS platform, and can it be managed efficiently?
- Does the network adapter have sufficient throughput to handle the I/O loads planned? Does the network have sufficient capacity to handle the traffic generated by directory clients and servers?

Purchasing a server without sufficient scaling capacity means that you'll need to replace that hardware at some point in the future. This may be unavoidable considering how quickly CPU speeds are improving and prices are dropping. Ideally you should choose to replace hardware when newer, cheaper, and faster hardware is available, not because your current hardware is overburdened and cannot be upgraded.

Another way to reduce hardware costs is to choose directory server software that runs on multiple hardware and OS platforms. Choosing this type of software allows you the flexibility to select the best hardware for the task instead of limiting you to a single hardware and OS platform. This approach also makes it more likely that you'll be able to choose a hardware vendor with which you already have a relationship, in which case you may be able to leverage existing software and hardware maintenance contracts, and possibly improve your quantity discounts.

Being creative when negotiating pricing can also help you reduce hardware costs. For example, you may purchase hardware in larger quantities. This approach, of course, may be at odds with deploying your directory in a stepwise fashion, in which you might plan to add additional capacity only as the need arises. However, if you have some idea of how your directory needs will expand, you may be able to negotiate a better price from a vendor by committing to purchase the additional hardware over a specified period. If you can pool your purchases with other projects, you may be able to obtain even more favorable pricing.

Finally, to reduce hardware purchase costs, choose directory server software that is efficient and scales well in terms of the following:

- The total number of directory entries stored
- The total size of the database
- The average and maximum sizes of a directory entry
- The maximum number of concurrent client connections supported
- The maximum number of LDAP operations that can be performed in a given time period

Although all of the major directory software vendors provide at least reasonable scaling performance for all of these factors, some are better than others in each area. Depending on your directory needs, some performance factors may be more important than others.

Ask the software vendor to suggest reasonable hardware configurations for the directory you plan to deploy. Be sure to provide information both on the number of entries you plan to store in your directory and the type of directory client load you plan to place on the directory. You may find that the minimum hardware requirements vary widely depending on the software vendor. Obviously a directory that can be deployed efficiently on a single server is less expensive to deploy than a directory that requires data to be partitioned among many servers. Does the server software take advantage of multiple CPUs? Alternatively, can additional replicas be deployed easily to handle additional client load?

The answers to all these questions will help you understand how hardware costs will be affected by your choice of directory server software. These answers will also affect your hardware choices. For example, if a directory server does not make effective use of more than four CPUs, there is little benefit in purchasing systems with eight CPUs.

You may be able to reduce your costs further by sharing some of the more expensive components with other projects. For example, most SAN and NAS solutions are designed to be shared among many servers. Similarly, load balancers can usually be logically partitioned to balance load to more than one logical set of servers.

Finally, effort spent tuning your directory service can reduce your costs significantly. A well-tuned server makes more efficient use of the hardware it runs on. For example, Netscape Directory Server 6 operates best when it has cache settings tuned for maximum performance, and when the proper indexes are configured.

Deployment Software Costs

To deploy your directory service, you need several different types of software. You may also need to develop custom software for your directory service. Software costs might include the following:

- **Operating system software.** Your servers will require an operating system. Depending on the hardware platform, your OS costs may be one-time costs (possibly with upgrade costs in the future), or they may involve an up-front cost and an annual license or maintenance fee.
- **Operating system enhancements.** To enhance performance and reliability, it may be beneficial to purchase additional software that enhances the OS. For example, you may purchase high-performance file system software, disk mirroring software, high-availability software, and clustering software.
- **Directory server software.** Directory server software may be priced per server, per seat (per user), or by another method. Understand the long-term costs of your vendor's pricing model.
- **Utility software.** You may find it beneficial to purchase utility software to assist you in data maintenance tasks.
- **NMS software.** As described in [Chapter 19](#), Monitoring, you can use a commercially available NMS package such as Aprisma Spectrum Enterprise Manager or HP OpenView, or you can develop custom monitoring software. You may be able to

leverage an existing deployed NMS, in which case your costs will be lower.

- **Synchronization software.** If you need to synchronize dissimilar directories, you may want to purchase a metadirectory package or develop custom synchronization scripts in-house. If the number of directories that need to be synchronized is small and the synchronization process is straightforward, custom code can be a cost-effective solution. When there are many directories to be synchronized, or the data flow is complicated, a metadirectory may be required.
- **Directory client software for end users.** This category may include commercially available client software or custom-developed software such as HTML-based interfaces that allow your users to interact with the directory.
- **Development tools for application developers.** If you plan to develop your own directory-enabled applications in-house, you may need to purchase development tools. Generally, LDAP client software development kits are available free of charge, but you may need to purchase compilers, graphical user interface (GUI) libraries, and other utility software necessary to develop your applications.
- **Directory-enabled applications.** Along with the deployment of your directory, you may be able to deploy either new directory-enabled applications or upgrades to existing applications to make them directory-enabled. More information on directory-enabled applications can be found in [Chapters 21](#), Developing New Applications, and [22](#), Directory-Enabling Existing Applications.
- **Backup and restore software.** In some cases, backup devices are bundled with backup and restore utility software. Even if this is the case, you may decide to purchase a more comprehensive package—for example, if you need to back up remote network nodes automatically to manage a tape carousel or jukebox.

Reducing Deployment Software Costs

As with hardware costs, software costs are often negotiable. Ask the vendor about various pricing options. Is the pricing per CPU? per server? per user? per entry? If multiple pricing options are available, one might be the best choice for your situation now, but another might benefit you down the road. Be sure to consider future growth when deciding on a pricing model.

Often you can obtain better pricing by purchasing additional software from the vendor at the same time. Take the time to find out whether your organization is in (or is planning) negotiations with the vendor over other software. A vendor will often provide more favorable pricing when multiple products are being purchased.

Ongoing Costs of Providing Your Directory Service

As soon as the directory service is in production, there are costs associated with maintaining the directory server hardware and software and the directory data. Costs also are associated with supporting end users and scaling the service to meet the needs of a growing company that is developing and deploying new directory-enabled applications.

Software Upgrade Costs

When your directory service is in production, you may need to invest in software upgrades from time to time. Upgrades may be needed for the following software packages:

- **Operating system software.** As new versions of OS software become available, you may want to upgrade to obtain benefits such as increased performance or the ability to address larger amounts of memory or disk space. You may also need to upgrade your operating system when you upgrade your directory server software to take advantage of additional functionality.
- **Directory server software.** Upgrades of directory server software that add additional features, provide better performance, or fix bugs will become available.
- **Directory-enabled application and client software.** Directory-enabled applications (such as e-mail server software and groupware applications) and end-user client applications may need to be upgraded from time to time. New versions of these applications may require that you also upgrade your directory server software to enable new features.

Apportioning Hardware and Software Costs

When you consider the various types of hardware that your directory service comprises, some items are completely specific to the directory service itself, whereas others are shared among all parts of your computing infrastructure. For example, if you have a dedicated machine on which you run an LDAP server, the CPU, memory, disks, monitor, and so on are used solely to provide the directory service. On the other hand, the router that connects the machine room Ethernet to the rest of your network is shared among multiple services.

Similarly, some software may be shared among multiple services. For example, you might have a site license that allows you to install operating system software on multiple systems throughout your organization. A portion of the site license cost should be attributed to the directory service.

When analyzing costs, you might perform a simple calculation to apportion the costs of these shared resources among the various services. If you have a machine room that houses 50 server computers, and 5 of those servers are dedicated LDAP servers, you might allocate 10 percent (5 divided by 50) of the total machine room costs to the directory service. These costs cover networking (routers, hubs, cabling, monitoring software), power (AC power, uninterruptible power supplies), air conditioning, and staffing costs, if any.

Of course, this calculation can be more complicated. For example, if the

attachment of your directory servers requires an upgrade to the router hardware (to handle the additional load), it might be argued that the directory service should assume the entire cost of the upgrade. On the other hand, it's likely that another service will eventually be deployed or expanded, necessitating the upgrade. As with most types of budgeting, there are always opinions on both sides.

- **Other software.** Other software used in support of your directory service may need to be upgraded during your directory's life cycle. Such software might include NMS software and metadirectory software.

Reducing Software Upgrade Costs

To reduce software upgrade costs, you can use two techniques: negotiating the best price for the upgrade, and deploying the upgrade efficiently.

When negotiating a price for the software upgrades, the same principles apply as when you make your initial software purchase. Volume discounts may be offered, so it is wise to purchase all your upgrades at once or negotiate an arrangement in which you commit to a certain number of upgrades over a set period of time. In some cases, your company may already have site licenses or support contracts in place that either cover the software you need or can be modified to include it at a nominal cost.

The second way to reduce costs is to deploy your upgrade in the most efficient manner. For example, if you need to deploy a new directory-enabled client application to all your users, certain approaches allow you to minimize the amount of staff time required. You might produce a self-extracting archive that users can install themselves (many software vendors distribute their applications in this manner anyway), or you can use one of the enterprise management packages (Tivoli, for example) to automate the distribution and installation of software packages on end-user computers.

Of course, for certain types of software, such as server software or Web-based applications, the number of places you need to install the upgrades is small, perhaps only a few servers. In that case, streamlining the installation process does not improve efficiency by much. In the case of server software, however, you can reduce overall costs to the organization by performing the upgrade in a way that provides the least disruption to end users and business processes. For example, you might schedule the upgrade during off-hours, when users will not be inconvenienced by unavailability of the directory. Be aware of any automated data maintenance processes that might also be scheduled during off-hours, and make sure that these processes either can tolerate a temporary directory outage or can be rescheduled to occur after the upgrade is complete.

Hardware Upgrade and Replacement Costs

As the demands on your directory increase, you may find it necessary to add additional capacity to existing servers or replace them entirely. The types of upgrade or replacement costs you may incur include the following:

- **Additional memory for servers.** As the number of concurrent client connections to your directory server increases, you may need to add more memory to your servers to maintain performance. When upgrading servers, keep in mind the number of available sockets for memory expansion. For example, suppose that your server has four memory module sockets available. Completely populating all four slots with lower-capacity memory modules means that you will need to remove some of them

later if you need to add more memory.

- **Additional disk space for servers.** If the amount of directory data you need to store increases, you may need to purchase and install additional disk capacity for your servers. This need might arise when the number of directory entries increases or if you need to maintain additional attribute indexes. Certain types of storage technology, including NAS and SAN solutions, allow you to add storage incrementally without shutting down your directory service. In addition, if you add more memory to your system, you may need to increase the amount of disk space used for virtual memory paging.
- **Other server upgrade costs.** You may choose to upgrade your servers by adding additional CPUs or switching to faster processors. You may also add more I/O capacity as the demands on your directory increase.
- **Server replacement costs.** As newer, faster hardware becomes available, you may decide to replace rather than upgrade your server hardware. In some cases you may be able to reuse peripherals such as network adapters on the new servers.
- **Costs for additional network capacity.** As the usage of your directory increases, you will probably find that you need to upgrade your network hardware at some point. For example, you may need to move to gigabit Ethernet technology, or you may need to provide an additional network segment and router port to handle the increased network traffic created by your service.
- **Upgrade and replacement planning costs.** When you upgrade or replace server equipment, proper planning is required. You'll need to spend time thinking about short- to medium-term growth (6 to 12 months), and develop an upgrade plan that will meet your needs during that time. Be sure to take the costs of this planning into account.

Reducing Hardware Upgrade and Replacement Costs

Just like negotiating a good initial price for your server hardware, you should try to negotiate the best price for upgrades and replacements, purchasing in larger quantities where possible.

If you've purchased server hardware that can be upgraded, it may be significantly cheaper to add CPUs to an existing server than to add and manage a second server. Of course, this approach makes sense only if your directory service has become CPU-bound. If your server is I/O-bound, adding another CPU won't be much help.

Costs for hardware can be amortized over the lifetime of the equipment so that the entire cost of new hardware or upgrades need not be paid in a single fiscal year. The same principle can also be applied to software purchases and upgrades.

Finally, you may be able to recover some of the costs of your old equipment by selling it.

Monitoring Costs

Proper monitoring of your directory service requires upkeep of your monitoring tools as well as ongoing staff costs for responding to directory problems. The following are some of the monitoring costs you may incur:

- **Pager and cell phone fees.** If you have on-call staff, you probably equip them with pagers or cellular telephones so that they can be contacted in the event of a problem. Initial purchase costs and monthly charges must be taken into account.
- **On-call pay.** If your maintenance staff is on call and must be available to address problems, you will incur additional salary costs.
- **Costs associated with refinement and maintenance of monitoring software.** As your directory service evolves and new servers are deployed, you will need to incorporate monitoring of those servers into your existing monitoring system.

Reducing Monitoring Costs

To reduce monitoring costs, leverage any existing network management infrastructure that might be present in your organization. For example, instead of designing your own directory monitoring system that runs in parallel with an existing monitoring system, integrate your monitoring with the existing NMS. Doing this not only allows you to avoid reinventing the wheel, but it also provides a central focus point where your users and maintenance staff can go to learn about all system failures.

Data Maintenance Costs

Maintaining the data in your directory, which is discussed in [Chapter 18](#), Maintaining Data, is one of the most important ongoing activities you will perform. Good maintenance of your directory ensures the quality of the data, which in turn improves the usefulness of your service. Here are some of the data maintenance costs you may incur:

- **Personnel costs.** Your data update process may involve manual tasks. For example, you may have a process that requires a computer operator to retrieve a tape containing a dump of personnel data from an administrative mainframe computer, run the data through some sort of transformation, and load the data into the directory. Even if most routine tasks are automated, periodic review of the log files produced by these tasks is still necessary to ensure that they are functioning properly.
- **Fees levied by data owners.** If your organization is arranged into separate cost centers, the other parts of the organization may charge you for obtaining their data. For example, a Human Resources division with its own IT staff may charge a fee to extract the data you need to update your directory.
- **Costs of ongoing development data maintenance tools and procedures.** When external data sources change in some way, perhaps moving from mainframe-based databases to a relational database running on a Unix server, you may need to change your data import tools and procedures at the same time.
- **Metadirectory maintenance.** If you use a metadirectory to synchronize external directories, you will need to spend time maintaining it and its relationships with the foreign directories. Metadirectories are discussed further in [Chapter 23](#), Directory Coexistence.

Reducing Data Maintenance Costs

To the greatest extent possible, automate your data management systems. The way you accomplish this automation will vary depending on how your data is managed. This section

presents a few ideas.

If your directory is routinely updated from a central source, such as a human resources database, automate the update process as much as you can. By using automatically scheduled processes to perform the updates, you free up staff to concentrate on tasks that are more useful. Make sure, however, that the update process provides useful diagnostics when problems are encountered and that someone periodically reviews the diagnostic output. For more information on updating directory data from external sources, see [Chapter 7](#), Data Design.

It's likely that your external data sources are themselves undergoing development and modernization; you should be prepared to deal with such changes. Whenever possible, make data transformation tools flexible. For example, suppose that you need to develop a tool that transforms a table of ASCII data into an LDAP Data Interchange Format (LDIF) file for import into the directory. If you can make the mapping from column locations to attribute types table-driven, you can accommodate changes in the width of the columns by simply updating a table instead of changing the program code.

Whenever possible and desirable, delegate responsibilities for updating data to departmental administrators or even end users. Of course, such delegation is possible and desirable for only certain attributes, but it may make sense to make those groups responsible for some data updates. Delegation distributes responsibility for data management across the organization and reduces administrative burden. It also improves the quality of the information in the directory by improving its timeliness.

If you do allow departmental administrators or end users to modify certain attributes, make sure that you deploy easy-to-use tools for this task. An application should validate the values that the users enter and reject invalid data with a helpful message that describes the appropriate format for the data.

Backup and Restore Costs

Keeping your data backed up, as discussed in [Chapter 17](#), Backups and Disaster Recovery, is important. Safeguarding your mission-critical data involves the following costs:

- **Personnel costs.** Some staff costs will be associated with performing regular backups. Loading and unloading backup media and performing backup and restore operations are some of the tasks involved.
- **Backup media costs.** You need to have a sufficient supply of blank backup media to accommodate your backup strategy. Also make sure that you plan for the retirement of older backup media to prevent failing media from endangering the integrity of your backups.
- **Transportation and off-site storage costs.** A robust backup strategy involves transporting the backup media to an off-site location to prevent its destruction in the event of a disaster. You may incur expenses for shipping the backup media to the remote location and for leasing the storage space.

Reducing Backup and Restore Costs

One obvious way to reduce backup and restore costs is to use an existing backup system if one is already in place. If all your servers are backed up across the network to a central tape drive, backing up the directory servers to this system will be cheaper than deploying a

new backup system.

However, you might decide to purchase and deploy a backup system instead of using an existing system. In that case, when comparing expenses for various backup solutions, consider the cost of the hardware and the cost per byte of the media. In some cases a backup device that costs more may actually be less expensive to operate over the long term because it is able to store more data on each individual piece of backup media. Tape libraries, although more expensive, can manage multiple tape cartridges automatically and reduce the amount of operator intervention required.

Backups protect you against catastrophic data loss. They can also be used to allow end users to recover from incorrect changes made to their own directory entries. However, performing restore operations for end users can be time-consuming and expensive; most backup software does not even provide access to individual entries in a backed-up directory. Usually the entire directory must be restored to a different location and the required entries extracted. In general, avoid setting expectations that backups are to be used for anything other than recovery from catastrophic data loss.

Whatever backup approach you decide on, consider keeping several days worth of daily backups on disk so that if it's necessary to restore data, you don't need to retrieve a backup tape.

Disaster Recovery Plan Costs

A well-designed and well-tested disaster recovery plan can protect your critical business processes from certain types of disasters, including floods, earthquakes, and fires. Development and execution of a disaster recovery plan are expensive; but compared to the potential catastrophic business losses that could accompany a disaster, the development costs seem much more reasonable. For more information on disaster recovery, see [Chapter 17, Backups and Disaster Recovery](#).

Here are some of the costs you may incur for your disaster recovery plan:

- **Periodic disaster recovery service fees.** If you use the services of a disaster recovery (DR) vendor to provide a hot or cold backup site, you will pay that vendor on a regular basis.
- **In-house disaster recovery costs.** If disaster recovery is provided in-house, you will incur costs associated with providing the directory service portion of the entire DR solution, including space rental and backup hardware and software.
- **Disaster recovery testing.** After your DR plan has been deployed, it should be tested periodically. This testing may be elaborate because it requires simulating an actual disaster. A significant amount of planning is involved in the development of the DR test plan, and a significant amount of staff time can be spent performing the tests.
- **Disaster recovery review and update.** Periodically, your DR plan should be reviewed and updated to accommodate new applications or changes in the underlying directory service. If new applications have been developed or your directory service has changed in a fundamental way, it may be necessary to redesign your disaster recovery solution.

Reducing Disaster Recovery Plan Costs

It's certainly possible to design a comprehensive disaster recovery plan that will protect you against most types of disasters and have your business processes back in service quickly. However, the costs of such a plan may be prohibitive. A cost-effective DR plan takes into account the likelihood of each type of disaster.

For example, you might be deciding where to locate a cold standby location for your data center. If you are situated in an area prone to earthquakes, it probably makes sense to locate the backup site far enough away that it's unlikely that both the primary and the backup sites would be affected by a single earthquake. On the other hand, if your location is not subject to earthquakes (or other large-scale disasters such as hurricanes), you can save money by locating your backup site relatively close to your primary site. In the event that a disaster does render the primary site unusable, it will be less costly to transport your staff to the backup site if it is nearby.

Also consider the acceptable time for putting a backup site into service. Maintaining a hot site certainly offers the quickest turnaround, but a hot site is also significantly more costly than a cold site to maintain and test. If it's acceptable to incur 48 hours of downtime with a cold site versus 4 hours with a hot site, you can save money by using the cold backup site.

Beyond the disaster plan itself, another way to save money is to weigh the relative costs of contracting with a DR solutions provider versus providing the backup sites yourself. If your organization has only a single location capable of supporting a backup site, it may make more sense to contract with a DR solutions provider. On the other hand, if your organization has several locations with high-speed network connectivity, sufficient power and backup generator capacity, and enough physical space, you may be able to design and deploy your own backup site at a lower cost. For more information on DR, see [Chapter 17](#), Backups and Disaster Recovery.

Support and Training Costs

Support and training enable your end users and directory administrators to use and maintain the directory more effectively. Here is a wide range of options for providing training and support, with associated costs:

- **End-user support costs.** You can provide support for your end users by maintaining an in-house help desk or contracting with an external support provider. Tasks include password resetting, software troubleshooting, and other general end-user support services.
- **Training for end users.** Your end users can be trained in various ways, depending on the application. For commercially available applications, you may be able to use vendor-supplied training materials or the services of a third-party training organization. You may find that online tutorials are available or that you can send your end users to training classes and seminars. For applications developed in-house, you need to develop your own training materials, which can take the form of online help and tutorials, seminars, brown-bag sessions, or printed training materials.
- **Training for support staff.** Your support staff will also require training on the various directory technologies used within your organization. The types and sources of training materials are generally the same as for end-user training. Unlike end users, however, your support staff will require deeper knowledge and easy access to reference materials such as operations manuals and troubleshooting guides. For applications developed in-house, your application developers may be available to develop a training course for the administrators and Help Desk personnel who will support the application.

- **Training for developers.** If you plan to develop directory-enabled applications in-house, you may need to provide training for your developers, especially if LDAP and directory technologies are new to your organization. Developer training resources might include seminars, conferences, and reference documentation.

Reducing Support and Training Costs

Whether your organization provides an in-house help desk or contracts with an external provider of support servers, providing your end users with better information can significantly reduce the number of support calls. Here are some suggestions for doing this:

- Make sure your directory-enabled applications provide clear and helpful error messages when a directory problem is encountered. These messages should state succinctly what the problem is and what the user should do about it.
- Use the Internet to deliver help documents to end users and advertise their location. If your users know that there is a high-quality collection of help documents online that often answer their questions, you can head off many support calls.
- Provide status information about your service to your end users. For example, if your directory service is temporarily unavailable, providing this information on a central status page and via a recorded telephone message might reduce the number of calls received by your support desk.
- Provide convenient training for your users to help reduce the number of support calls. Seminars and online tutorials are excellent ways to improve end-user knowledge.
- Provide good reference material (such as this book) to your directory developers and deployers.

Support and Maintenance Contract Costs

Keeping your directory system running smoothly is much easier if all your hardware is functioning correctly and your software is up-to-date. The types of support and maintenance contracts you might purchase include the following:

- **Software support contracts.** Some of the software packages you use to provide your directory service may offer (or require) an annual maintenance fee, which entitles you to bug fixes and support. For example, some OS vendors offer an online database of patches available only to customers who purchase a support contract.
- **Hardware support/maintenance contracts.** You might choose to purchase a maintenance contract for your server hardware that covers the cost of repairing any failed hardware. These maintenance contracts provide widely varying turnaround times and costs. The advantage of maintenance agreements is that their costs are fixed and can be included more easily in a budget.
- **Self-servicing costs.** An alternative to purchasing a maintenance agreement is to provide your own in-house service. This approach requires that you have adequate knowledge in-house to diagnose hardware failures, as well as a supply of spare parts for all your servers. It also requires that you have an arrangement with a supplier who will exchange failed modules for new or refurbished modules (for a fee, of course). If you have a heterogeneous computing environment, consider a vendor who can stock parts for a wide range of computing systems.

Reducing Support and Maintenance Contract Costs

Hardware maintenance contracts vary widely in terms of turnaround time. If you have

sufficient extra capacity in your directory service (perhaps you've deployed some replicas and have spare capacity), you may be able to tolerate a longer turnaround time for repair or even use a depot service arrangement.

If your organization is large, it may make sense to provide your own in-house service as just described. If you have sufficient knowledge in-house to consider this option, you may find that it is significantly less expensive than purchasing a service agreement.

Finally, as with the other major purchases you make to deploy your directory service, purchasing support and maintenance agreements in larger quantities may offer some benefits when you're negotiating prices.

Costs of Adding New Directory-Enabled Applications

Planning to accommodate new directory-enabled applications and additional load is vital to your directory's continued success. Understanding the costs of these enhancements, which are described in the following list, is therefore also necessary:

- **Costs of additional systemwide capacity for new directory-enabled applications.** Each new directory-enabled application deployed may make additional demands on your directory. For example, if you deploy an extranet application to your distributors, you might use the directory to control access to the application. This means that your directory must be prepared to accommodate directory entries for all the users at your distributors who will use the application. You might choose to deploy a dedicated replica of your directory data for the exclusive use of that application.
- **Development costs for new applications.** Developing a new directory-enabled application has a set of costs associated with it, including costs for development hardware and software, training for developers, and developer salaries and benefits.
- **Deployment costs for new applications.** Putting new applications into production involves piloting the application, developing and distributing documentation, training your support staff, and rolling out the application for general use.
- **Costs associated with planning for expansion.** In addition to the actual costs of capacity expansion, planning for the expansion has its own associated costs. Typically these staff costs involve the process of understanding and planning for the additional capacity needed for new applications.

More information on directory-enabled applications is available in [Chapters 21](#), Developing New Applications, and [22](#), Directory-Enabling Existing Applications.

Reducing Costs of Adding New Directory-Enabled Applications

When adding capacity to your directory service, use the same techniques you used to obtain the best price for your original hardware. Also be sure to purchase appropriate hardware for continued growth. For example, if you know that in the future you will need 100Mbps networking to your servers to handle your directory traffic, don't make a large investment in 10Mbps technology that will have to be replaced.

You can reduce ongoing costs of application development by using sound software engineering techniques. Try to develop reusable components that can be shared by your developers. For example, you might be able to avoid reinventing the wheel by developing or

purchasing a utility library that contains code common to all directory-enabled applications. This library can be distributed in binary form and linked with new applications. If bugs are found in the library, they can be fixed by the library maintainer, and a new library can be distributed to the application developers.

Team LiB

◀ PREVIOUS

NEXT ▶

Analyzing and Reducing Costs Checklist

To review, here are some important things to consider as you plan for reducing your directory costs:

- Understand how your directory service affects the overall business. How does it save money? How does it cost money? Knowing this information can help you maneuver through the politics of your organization more effectively.
- Understand the various costs associated with each portion of your directory life cycle —design, piloting, deployment, and maintenance.
- Hardware and software costs are lower than personnel costs. Reduce bottlenecks that prevent your staff from being efficient.
- Automate repetitive processes.
- Seek input from your maintenance and deployment staffs on how efficiency can be improved.

Further Reading

Best Practices in Information Technology: How Corporations Get the Most Value from Exploiting Their Digital Investments. J. W. Cortada, Prentice Hall, 1997.

The Effective Measurement and Management of IT Costs and Benefits, 2nd Edition. D. Remenyi, A. Money, and M. Sherwood-Smith, Butterworth-Heinemann, 2000.

IT Investment: Making a Business Case (Computer Weekly Professional Series). D. Remenyi, Digital Equipment Corporation, 1999.

Looking Ahead

In this chapter we have provided a sample listing of the various costs your directory service might incur during its lifetime, along with some suggestions for reducing these costs and making your directory service as efficient as possible. If you understand the costs, you can effectively communicate this information and negotiate the funding you need to handle additional capacity and growth. In [Chapter 16](#), Putting Your Directory Service into Production, we will focus on developing a plan for throwing the switch and putting your directory into full production.

Chapter 16. Putting Your Directory Service into Production

- Creating a Plan for Putting Your Directory Service into Production
- Advice for Putting Your Directory Service into Production
- Executing Your Plan
- Putting Your Directory Service into Production Checklist
- Looking Ahead

Putting your directory service into production is a big step. It is what you have been working toward as you have developed an understanding of directory services and your own project's needs, created a detailed design, and performed a directory service pilot. Because you get only one chance to make a good first impression, it is important to execute well at this stage.

In conducting your pilot, you learned a lot about what is involved in rolling out and running a directory service. Here are some of the ways a production directory service is different from a pilot directory service:

- **Expectations are dramatically higher.** People expect a higher level of service, including good performance, minimal downtime, and absolutely no loss of data. To meet these expectations, you need to monitor your service and have staff on call to fix problems as soon as they are detected.
- **The audience for your service is significantly larger.** To the early adopters who made use of your pilot, you will add a large set of people who view computing services simply as tools to help them get their jobs done. These users demand an easy-to-use, well-documented, and well-supported service.
- **The scope and scale of your service may expand greatly.** In a typical pilot deployment, not all the servers and applications that you plan to use in your production service are online. In addition, production servers are often deployed on larger machines than pilot servers are, and the larger machines may require more care and feeding for backups and other maintenance tasks.
- **Monitoring and troubleshooting responsibilities are increased.** The people deploying the directory service have a greater responsibility to keep an eye on the service and to be available to fix problems. This is really a consequence of the increased expectations and larger audience for the service.

Throughout this chapter we assume that your directory service deployment is fairly large and complex. Therefore, we place a lot of emphasis on written plans and communication among the people participating in the production rollout. If your deployment is small, you can safely use less formal methods of communication. Whether it is large or small, you can win many supporters simply by providing a useful, reliable service and telling people about it right from the start.

The best way to achieve a successful production rollout is to start with a comprehensive plan. The first section of this chapter helps you create such a plan. After that we present some advice to help you succeed with your production deployment. We finish with a discussion on how to execute your plan successfully.

Creating a Plan for Putting Your Directory Service into Production

A good plan for putting your directory service into production covers a lot of ground. Unless the scope and audience for your directory service are extremely small, you should develop a detailed, written plan before putting your directory service into production. Everyone who will be involved in or directly affected by the rollout should participate in creating and reviewing the plan.

A good plan for putting your directory service into production includes five major parts:

1. A list of resources needed for the rollout
2. A set of prerequisite tasks that must be completed before the rollout
3. A detailed rollout plan that indicates what needs to be done, the order in which you will do it, and who will be responsible for completing each task
4. Criteria used to measure the success of the service
5. A publicity and marketing plan

These five parts of the overall plan are discussed in detail in the sections that follow.

List the Resources Needed for the Rollout

One key to a successful rollout is to make sure that all the necessary pieces are in place before you bring up any services. Some of the resources you need to line up include

- Directory service software, including appropriate licenses, support contracts, and any necessary documentation. Refer to [Chapter 13](#), Evaluating Directory Products, for more information on selecting directory software.
- Hardware, operating system software, and third-party software that will be used to deploy and manage your servers.
- Other infrastructure elements, such as network connections, physical space for servers, uninterruptible power supplies, and systems to enforce physical security of your most important servers.
- Directory management tools to meet the needs of those who manage the directory service. A combination of off-the-shelf products and custom tools developed in-house is usually needed.
- Administrative staff who are responsible for the day-to-day care and feeding of the service. Their duties include supporting people who use the service, monitoring the service, performing backups, and being on call to respond to and fix problems.

Depending on the specifics of your own environment and the goals for your deployment, you may need additional resources. Create a checklist of all the required resources, and assign a responsible person to ensure that each resource is obtained before you expect the rollout to begin.

Create a List of Prerequisite Tasks

You should do several things before you start installing servers and rolling out your service.

These prerequisite tasks include the following:

- Create a directory service test bed that you can use to simulate server load, try out new versions of directory software, experiment with directory or application design changes, and so on. Ideally, this test bed should be completely separate from your production service (that is, located on a different part of the network, or using different server and client hardware). Sometimes hardware used in the piloting stage is used to create such a test bed.
- Obtain the data you will use to populate your directory. This task may be difficult and time-consuming if the data comes from a source outside your immediate control. If you need to merge data from more than one source, obtaining the data and preparing it for use in your directory service will be even more time-consuming. Hopefully you worked out the procedures for this process during your directory service pilot; if not, now is the time to do so.
- Obtain permission to publish the data in your service. For example, you may need to ask your Legal department to sign off on your plan for putting end-user or customer information in the directory service.
- Coordinate the deployment of the directory applications that will use your directory service. Presumably these applications will be put into production shortly after your directory service is. Alternatively, they may already be in production, but the plan is to change them to use your service instead of a proprietary directory service or database. Either way, you will need to coordinate with the people who run the directory-enabled applications. Your own group may be responsible for some of the applications, of course.
- Develop a complete set of well-documented, well-understood procedures for monitoring the directory service (see [Chapter 19](#), Monitoring, for more information).
- Develop a plan for performing backups, as well as a disaster recovery plan (see [Chapter 17](#), Backups and Disaster Recovery, for more information).
- Educate important stakeholders who are involved or interested in the directory services project and prepare them for the rollout. Important stakeholders might include your Human Resources staff, Telecom staff, Legal department, and so on. By informing these other groups in advance of your rollout, you avoid surprising them and can tell them whom they should contact if any problems or concerns arise.
- Conduct appropriate training for support staff who will help directory users.
- Conduct appropriate training for system administrators who will troubleshoot and resolve directory-related problems (see [Chapter 20](#), Troubleshooting, for more information).
- Create all necessary end-user documentation and online help.
- Publicize the service.

Again, depending on your environment and your design, you may need to do additional things before beginning your production rollout. Create a checklist of all the prerequisite tasks and assign a person to be responsible for each task. Consider dependencies among the different tasks and make sure that you prioritize your efforts accordingly.

Create a Detailed Rollout Plan

Create a detailed rollout plan that indicates what needs to be done, the order in which everything must be done, and who is responsible for completing each task. This plan should include the following information:

- A list of the people directly involved in the production directory service rollout, along with their primary tasks.
- A list of the people indirectly involved in the rollout. This list should include people responsible for directory-enabled applications, Help Desk staff, and so on.
- A series of tasks for bringing up directory servers. This part of the plan should include the order in which you will bring up the servers, as well as the person

responsible for installing each one. For example, you might bring up one master server first, and then a second master, followed by three local replicas, followed later by seven remote replicas.

- A series of tasks associated with bringing each directory-enabled application online. Do not plan to bring up all applications at once. A phased approach is safer and more likely to succeed.
- A plan for ramping up the use of the service. If possible, gradually increase the service load by enabling more users and more directory-enabled applications. By doing so, you can more readily observe the system behavior and make appropriate adjustments before the service becomes overloaded.
- A plan for distributing documentation to administrators, and then to end users.
- A plan for training end users and administrators.
- Strategies for dealing with problems that might arise, such as unanticipated software or hardware bugs. Typically such strategies are documented in the form of an escalation plan that lists whom should be notified about which kinds of problems, and what procedures will be used if critical problems are not resolved quickly. Be sure to gather the support contract information for your software and hardware vendors.

Depending on the scope of your service, there may be anywhere from a few to dozens of people involved in the production rollout. In either case, make sure that each person involved helps create the detailed rollout plan and that each person understands her role.

Develop Criteria for Success

While you're lining up all the resources you need, tackling prerequisite tasks, and developing your detailed rollout plan, you should also create a document that lists criteria you can use to measure the success of your directory service rollout.

If you already have goals for your directory service (as discussed in [Chapter 6](#), Defining Your Directory Needs), use them to formulate criteria by which you can objectively evaluate the success of your production service. Don't be overly ambitious; it is probably sufficient just to jot down a few things you hope to accomplish. Here are some simple examples of production goals:

- To support your most important directory-enabled application without causing any unplanned service outages
- To respond to all directory-related questions from users within 24 hours of receiving them
- To handle 100,000 searches per hour on your busiest server with no errors and an average response time of one second or less
- To teach every administrative assistant in your organization how to use a new phone book application powered by your directory service
- To get 50 percent of the people you trained on the phone book application to use it to change an employee's entry within the first month of your production service

It is best to keep your success criteria simple. Share the criteria with the other members of your deployment team so that everyone is aware of the big picture. Make sure that all the criteria are measurable; it is important to know whether you're meeting your targets.

Create a Publicity and Marketing Plan

Before you begin your production rollout, develop a publicity and marketing plan for your new directory service. Your two major goals in this area should be to make people aware of the new service and to encourage them to use it.

Publicity can take many forms. For a directory service aimed at people inside your

organization, a few well-crafted articles for the company electronic newsletter or postings to appropriate discussion groups may be all you need. If your organization holds brown-bag meetings or any other type of training sessions on computing topics, sign up to lead a session or two to promote your new service and educate people about it. For a directory service aimed at people outside your organization, publicity and marketing will likely be handled directly through a Web site or by paper or electronic newsletters sent to partner companies.

Some topics you may want to cover when spreading the word about your directory service include

- The basics of directory services and LDAP
- Quick-start information on how to use the service
- How your directory service is being used now
- How you expect usage to grow in the future
- What a great new online service your team has created and how people benefit from it
- The extensive design and piloting work that was done by your team to prepare for a smooth production rollout
- All the help your directory services team had from other groups inside and outside your organization (don't forget to say thanks!)

It is important to tailor your publicity and marketing efforts to meet the needs of the intended audience. In practice, this means that you may need to develop separate materials for end users, directory application developers, system administrators, executives, and so on.

If possible, enlist the marketing people who work for your organization to help draft your publicity plan and create appropriate materials. Also consider whether you're willing to serve as a reference customer for any of your directory software suppliers. By doing so, you may be able to generate some external publicity about your organization and foster a better relationship with the supplier.

In summary, the deployment of a production-quality LDAP directory service is a major milestone. Make sure that people know about it. There is nothing wrong with congratulating yourself and your team on a job well done while simultaneously raising awareness about your new service!

Advice for Putting Your Directory Service into Production

You should consider a few more things before you execute your production plan. In this section we present some advice to help you avoid mistakes others have made.

Don't Jump the Gun

The most important advice we can give is this: *Don't try to put your directory service into production until you're ready*. The biggest mistake people make is to skip one or more essential design steps. Usually the result is a deficient, hard-to-maintain service. If you do skip a design step, you might need to go back and perform the step you left out and then redeploy your entire service. In the long run, the time you save up front by skipping some of the directory design work will be greatly outweighed by the aggravation you cause yourself and the users of your service.

The other major preproduction task that many are tempted to omit is the directory service pilot (described in [Chapter 14](#), Piloting Your Directory Service). A pilot is the best way to prove your directory design and learn what it's like to run a production service. By creating a pilot service and heeding the lessons learned, you greatly increase the chances of success for the production service.

Maintain Focus

Another common mistake is to lose focus and forget what you're trying to accomplish. Always keep your production success criteria in mind as you proceed with your deployment. Remind yourself who your most important users are and think about what you need to do to meet their needs. Work diligently to ensure that your most important directory-enabled applications succeed. The success of your service is linked closely to the success of those applications.

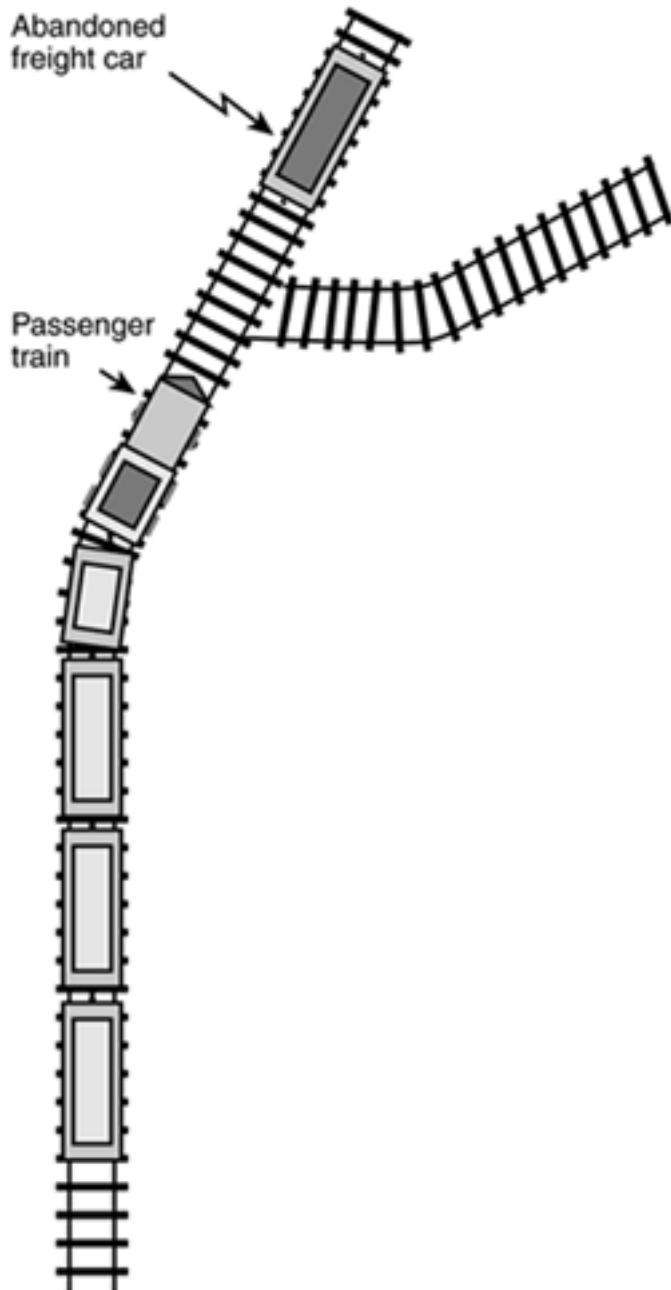
Adopt an Incremental Approach

One of the running themes throughout this book is that an incremental approach is more likely to succeed than an all-or-nothing approach. This is especially true when you're rolling out a production directory service. Don't try to add too many dependent applications the first day your service is up and running. Don't tell all of your users at the same time to start using the service or the directory-enabled applications. Don't create unrealistic expectations by overhyping your service. Don't try to roll out all the replicas your design includes in the first few days of service. Adopt a phased approach and make sure that each new part of the service is working before increasing the complexity of the system.

Prepare Yourself Well

There is an old story about a passenger train engineer who was faced with a crisis. While traveling with a full complement of passengers at high speed, his train came upon a single empty freight car that someone had mistakenly left on the main track instead of on a nearby siding (see [Figure 16.1](#)). Without missing a beat, the engineer immediately increased his engine speed to full throttle. His passenger train struck the empty freight car at nearly full speed, knocking it off the track and out of the way and avoiding derailing his own train.

Figure 16.1. A Dangerous Situation



After the incident, when the engineer was asked how he knew to increase his speed rather than try to stop, he had a ready answer: "I knew how far I was from the empty car because many times in the past, as I passed by that same stretch of track, I imagined that same exact situation and thought about what I would do. I knew that if I was too close to stop in time, I should increase my speed because the impulse created by striking the empty car would most likely drive it off the track and safely out of the path of my own train. I didn't have to think about any of this at the time of the crisis because I already knew exactly what to do."

The moral of the story is, of course, be prepared for whatever might come your way. This is good advice, and you should apply it to the rollout of your production directory service. Think ahead and prepare yourself as much as possible for anything that might go wrong. Play "what if" games and consider how you will react to any situation that arises, and change your strategy if necessary. If you take time to prepare yourself, you will be able to act as quickly and decisively as the passenger train engineer did.

Potential problem areas to consider include the following:

- **Bugs in hardware or software.** If these occur, you may need to adjust your design to work around the problem, or you may just need to wait for a bug fix from a

vendor.

- **Flaws in your design, in software, or in hardware that lead to inadequate capacity for your service.** If you can afford it, pad your resource estimates so that you have extra capacity. If necessary, be prepared to slow your rollout to reduce the load on the system until you can install additional servers.
- **Confusion of end users or directory service administrators.** Be prepared to develop additional documentation and provide additional training to alleviate any confusion concerning how to use or maintain the service.
- **Unexpected hardware failures.** Unexpected hardware failures lead to inadequate capacity or complete disruption of your service. If you can arrange it, have extra hardware lined up that you can borrow in the case of an unexpected failure.
- **Changes in the requirements for your service.** Any major change in requirements will demand that you take a step back and consider whether you need to adjust your plan. For example, if you find out that your service must support a new directory-enabled application, you may need to reconfigure servers to add new schemas, configure additional indexes, add additional replicas, or make other changes to accommodate it.
- **Staff shortages.** Shortages in staff might occur because of illness, family emergency, or people being called away to address problems in other production systems for which they are also responsible. You can reduce the pain caused by a staffing shortage by ensuring that each person involved in your directory service rollout has trained another person who can handle his tasks in an emergency.

Be realistic about the risks involved in your production plan, and think about how you will detect problems and what you will do if they occur. Make sure that you inform your management and the owners of directory-dependent applications of the risks so that they are not surprised by the problems or your actions.

For example, suppose that your design calls for you to roll out five replica servers to provide service to a busy directory-enabled mail delivery service. Consider what you would do during your production rollout if you found a critical bug in your directory service software that prevented you from creating more than three replicas. Clearly, such a bug would reduce the capacity of your service. You must decide whether three replicas would be able to handle the application load long enough to get a patch from your directory server software vendor. If not, you may need to scale back the mail service—or possibly delay your directory service rollout entirely. Depending on the situation, any of these options might be the best choice. Thinking about the options in advance enables you to act quickly and decisively if problems occur during your rollout.

Executing Your Plan

The last step, of course, is to put your directory service into production (finally!). Doing so means executing each of the five pieces of your production plan:

1. Gather needed resources.
2. Perform prerequisite tasks.
3. Roll out the service.
4. Check your progress against your success criteria.
5. Publicize your new service.

Before you start to execute your plan, make sure that you know where to turn for outside help. Create a list of contact information for hardware vendors, directory software vendors, suppliers of other software you're using, and anyone else whom you may need to ask for help. During the most critical phases of your rollout (for example, when your first important directory-enabled application comes online and begins using your directory), we suggest that you put on alert the people you may need to turn to for help.

Hopefully your rollout will go smoothly, but it may not. For example, you may encounter a critical bug that did not show up during your pilot. Or you may find that part of your design just won't work with the software you have or in the environment in which the service needs to run. If this happens, do not despair. Regroup and consider whether you need to revisit your design or conduct another pilot before proceeding with your production rollout. It is important to know what kinds of problems would lead you to halt the rollout and which ones could be dealt with while the rollout proceeded.

Finally, remember to keep everyone informed about your progress. Communicate openly with your management, colleagues, developers, and end users about how the production rollout is proceeding. Try not to promise more than you can deliver. After you have met all or most of your success criteria, let everyone know that you consider the directory service to be in production. And don't forget to thank everyone for their hard work and help in reaching this major milestone!

Putting Your Directory Service into Production Checklist

You should perform these tasks when putting your directory service into production:

- Make sure that your directory design is complete.
- Perform a directory service pilot.
- Create a plan for putting your directory service into production:
 - List the resources needed for your rollout.
 - Create a list of prerequisite tasks.
 - Create a detailed service rollout plan.
 - Develop criteria for success.
 - Create a publicity and marketing plan.
- Prepare for possible problems.
- Execute your production plan:
 - Gather needed resources.
 - Perform prerequisite tasks.
 - Roll out the service.
 - Check your progress against your success criteria.
 - Publicize your new service.

Looking Ahead

With this chapter we wrap up [Part III](#), Deploying Your Directory Service. By now, you should be well on your way to deploying a production-quality directory service. In [Part IV](#), Maintaining Your Directory Service, we will shift our focus to maintenance of the directory service. The topics covered in [Chapters 17](#) through [20](#) include backups and disaster recovery, managing your data, monitoring your service, and troubleshooting directory service problems.

Part IV: Maintaining Your Directory Service

[17. Backups and Disaster Recovery](#)

[18. Maintaining Data](#)

[19. Monitoring](#)

[20. Troubleshooting](#)

Chapter 17. Backups and Disaster Recovery

- Backup and Restore Procedures
- Disaster Planning and Recovery
- Directory-Specific Issues in Disaster Recovery
- Backups and Disaster Recovery Checklist
- Further Reading
- Looking Ahead

The data contained in your directory is (or soon will be) critical to the day-to-day operation of your organization. If that data becomes unavailable as a result of equipment failure or data corruption, your business's bottom line could suffer. Sound backup and restoration procedures safeguard your important data from damage. In addition to a backup strategy, you should have a disaster recovery plan that details the procedure for your organization to put its directory systems back in service after a disaster such as a fire or flood.

Of course, comprehensive backup and disaster recovery plans involve all the various information technology (IT) services provided, including file systems, databases, and applications. We won't attempt to provide a comprehensive guide to backup and disaster recovery in this chapter. Instead we'll focus on the specifics of backing up and restoring directories, and how these procedures differ from other backup/restore procedures that might be familiar. We'll also discuss disaster recovery procedures as they pertain specifically to your directory. With this knowledge you should be able to incorporate your directory into your established backup/restore and disaster recovery plans. For a more general guide to disaster recovery, see the Further Reading section at the end of this chapter.

Backup and Restore Procedures

Like file systems and databases, a directory's data is stored on disk drives. This data can become damaged for various reasons, including the following:

- **Disk drive media or controller failure.** The magnetic media on which the directory data is stored can fail, resulting in corruption or total loss.
- **Software bugs.** The directory software or operating system can have a bug that corrupts the data in the database.
- **Erroneous applications.** Directory applications can go haywire, placing incorrect data in the directory or deleting directory entries that should not be deleted.
- **Operator error.** Files or directory entries and attributes can be inadvertently erased.
- **Theft or vandalism.** Your directory server equipment and the data stored on it can be stolen or intentionally damaged.
- **Disasters.** Fires, floods, and earthquakes can all destroy your equipment.

You can back up your directory data in two ways: by using traditional backup techniques such as tape backups and disk mirroring, or by using directory replication. Each approach has advantages, and a comprehensive backup plan benefits from using both approaches simultaneously. We'll discuss the advantages of each approach throughout this chapter.

Backing Up and Restoring Directory Data Using Traditional Techniques

Just like user files stored on a file server, your directory data should be backed up periodically to some sort of medium such as magnetic tape. You can also back up to an alternate disk drive locally or over a network. The backups can be used to restore the data in the event of data damage or loss.

However, backing up a directory is different from backing up a file system in several important ways:

- File servers are usually much larger than directories; therefore, more data needs to be backed up. This means that whereas it's usually feasible to back up your directory by copying it to an alternate disk, file system backups generally require the use of higher-capacity backup media such as tape. However, very large directories, which may be many gigabytes in size, cannot be copied to an alternate disk.
- Unlike file systems, directories are frequently replicated, so it's important to understand the implications of restoring a replica from a backup tape. In most cases it's better to rebuild a damaged replica from its peer replicas than to restore it from a backup tape; the data in the peer replicas should be more up-to-date.
- The tools provided to back up directories generally do not support incremental backups, in which only the data that has changed since the last backup is copied to tape. This may change in the future, but for now, directories are generally backed up as a unit.
- The directory service may be distributed across multiple individual servers, with each server holding a portion of the directory information tree (DIT). To back up the entire service, you must either back up all the servers individually or back up all the directory data from a central location.

Your actual backup procedure depends on the particular directory server software you use. To back up Netscape Directory Server 6, you use the `db2bak` script, which is found in the server instance directory `install-root/slapd-instance-name`. This script, which can be run while the server is running, copies the database to a backup directory.

Backing Up an Online Directory Server

It is possible to back up a Netscape Directory Server 6 database while the server is running, without worrying about the fact that updates are being processed while the backup is in progress. The way the underlying database works is what makes this possible. All updates to the directory are written to a *transaction log* before a response code is returned to the client. The server marks the beginning and end of the update in the transaction log, thereby ensuring that all the changes written to the log are treated as a unit.

If the transaction is not completely written for some reason (perhaps the server's power is turned off), the client will not receive a success code, and the transaction will be incomplete. An incomplete transaction will be detected and discarded when the server is restarted because it will be missing the end marker. This setup ensures that if a client receives a success code in response to an update operation, the server has written the update to its disk. As long as the disk is not damaged, the data will not be lost.

When making a backup, Netscape Directory Server 6 first copies all database files to the backup directory. Then it copies any transaction logs that exist. This procedure ensures that all database changes made while the backup was in progress are contained in the backed-up data. When a backup is restored, any transactions found in the backed-up transaction logs are automatically applied to the database as the last step in the restore process. The database will then be in the same state it was at the time the backup was completed.

You can then copy the database files to backup media such as magnetic tape, or you can copy the backup files to a remote file system. You can also initiate the backup remotely using the `db2bak.pl` script. Examples that show how to use the `db2bak` script are included in [Chapter 4](#), Overview of Netscape Directory Server. For more information, see the *Netscape Directory Server 6 Administrator's Guide*.

By default, the `db2bak` script creates a new directory under the `bak` directory each time it is run. You can also specify a destination directory by passing it as an argument to the `db2bak` script. It is not possible to back up a Netscape Directory Server 6 database directly to tape. You need to take a snapshot using the `db2bak` script and then archive the database to tape. Therefore, to perform a backup you need enough free disk space to hold a copy of the database. After the backup has been archived to tape and verified, the snapshot can be deleted.

Note

If you have extra free disk space, it is beneficial to keep several days of snapshots online, even though they have been archived to tape. In the event that erroneous data is placed in your directory, the snapshots can be used to restore the directory, speeding the recovery process.

Restoring Directory Data from a Snapshot

To restore a Netscape Directory Server 6 database from a snapshot, you must shut down the server and use the `bak2db` script, or you can use the `bak2db.pl` script to initiate a restore operation remotely (the database being restored will be taken offline if you use the `bak2db.pl` script). Both scripts can be found in the server instance directory `install-root/slapd-instance-name`. You give the full pathname of the backup directory as the argument to the script (restore the snapshot from tape to disk first, if required). The `bak2db` script will copy the snapshot files into the database directory, including any transaction logs required. After the `bak2db` script completes, you can start the directory server. Examples that show how to use the `bak2db` script are included in [Chapter 4](#), Overview of Netscape Directory Server.

Backing Up to LDIF Files

What if your directory server software does not support online backup? Another way to back up your directory data is to read all the entries using the Lightweight Directory Access Protocol (LDAP) and write them to disk or tape in LDAP Directory Interchange Format (LDIF). For example, you can use the `ldapsearch` command-line utility to read all directory entries underneath `dc=example,dc=com` in the following command:

```
ldapsearch -h directory.example.com -p 389 -D "cn=Directory Manager" -w secret -s sub -b  
→ "dc=example, dc=com" "(objectclass=*)"
```

Note

The `ldapsearch` command should all be typed on a single line; it doesn't appear on one line in this book because of page size constraints.

This sample command connects to the LDAP server running on port 389 on host `directory.example.com`, authenticates as `cn=Directory Manager` with the password `secret`, and performs a subtree search rooted at `dc=example,dc=com`. (These parameters should be tailored for your environment.) The filter, `(objectclass=*)`, matches any entry. The server will respond by sending all entries within the subtree `dc=example,dc=com`, and the client will generate LDIF output that can be redirected to a file or tape.

There are some caveats with this approach. First, the `ldapsearch` command as given would not return any operational attributes such as `modifiersName` and `modifyTimeStamp`. These attributes must be mentioned explicitly on the command line, as illustrated here:

```
ldapsearch -h directory.example.com -p 389 -D "cn=Directory Manager" -w secret -s sub -b  
→ "dc=example, dc=com" "(objectclass=*)" "*" modifiersName modifyTimeStamp  
creatorsName  
→ createTimeStamp
```

The additional arguments at the end of the command line specify that all user attributes are to be returned (as called for by the asterisk), along with four operational attributes (`modifiersName`, `modifyTimeStamp`, `creatorsName`, `createTimeStamp`).

The second caveat is that the command-line utility may return entries that do not reside on the specified server. This can happen if your directory is distributed across multiple servers using chaining or referrals (as discussed in [Chapter 10](#), Topology Design). Receiving these entries may be what you want, of course, if your aim is to copy all your directory entries across your whole distributed directory. If this is not what you want, you can use the `-R` command-line flag to `ldapsearch` to instruct it not to follow referrals. If your directory uses chaining, however, there is no way to ask for entries local to only one server.

A third caveat is that if the directory is modified while the backup is under way, certain inconsistencies may result. For example, if a group references members, those members may not appear in the

backup snapshot if they are added while the directory is being backed up.

The final caveat is that data backed up in this manner is likely to be incomplete. Quite a bit of additional information may be stored in the directory to support its operation, such as superior and subordinate knowledge. The **ManageDSAIT** control, available on LDAPv3-compliant servers, can be used to access this type of data (this control is discussed in [Chapter 2](#), Introduction to LDAP).

The bottom line is that using a client to read the data over LDAP may not produce a backup that can be used to recover from complete data loss. If your directory server software supports direct backup of its database files, that method is preferred.

Restoring Data from LDIF Files

To restore data in an LDIF file, you simply shut down the directory server and import the LDIF file. For example, to restore a Netscape Directory Server 6 database from an LDIF file, stop the server using the **stop-slapd** script, import the LDIF file using the **ldif2db** script, and start the server using the **start-slapd** script. All of these scripts are documented in the *Netscape Directory Server 6 Administrator's Guide*.

Other Things to Back Up

Although backing up your critical directory data is important, additional information should probably also be backed up. For example, you should back up the configuration files for your directory server when you back up your directory data; re-creating them from scratch would be time-consuming. In addition, your directory schema configuration and access control information may reside in separate files or databases; make sure that these are backed up as well. Consult your directory server documentation to learn about other configuration files and data that you should back up.

Tip

When you're maintaining a complex set of configuration files like those in most directory server software, it's beneficial to keep a history of changes made to them. Having such a record allows you to revert to an older version of the configuration if an error is made. One way of creating this history is to use a revision control system such as the Source Code Control System (SCCS), Revision Control System (RCS), or Concurrent Version Control System (CVS), all of which are available on Unix platforms. Revision control systems allow you to retrieve any previous version of a file's contents and determine who made a particular change. Such tools can also help protect against erroneous configuration changes.

Backing Up Netscape Directory Server 6 Configuration Data

Netscape Directory Server 6 stores all its configuration data beneath a directory named **config** underneath the server instance directory. For example, if the directory server is installed on a Unix host in a server root directory named **/usr/netscape**, and the server instance name is **directory**, configuration files for the server are stored beneath the directory **/usr/netscape/slapd-directory/config**. You can back up this configuration data by copying all the files in that directory to a backup location such as a remote disk drive or a tape. You can use the Unix **tar** (tape archive) utility to place all the configuration files into a single archive. For example, if you wanted to back up all the configuration files and place the **tar** archive in the directory **/n/backups**, you could use the command **tar cf /n/backups/slapd.config.backup.tar /usr/netscape/slapd-directory/config**.

Using Replication for Backup and Restore

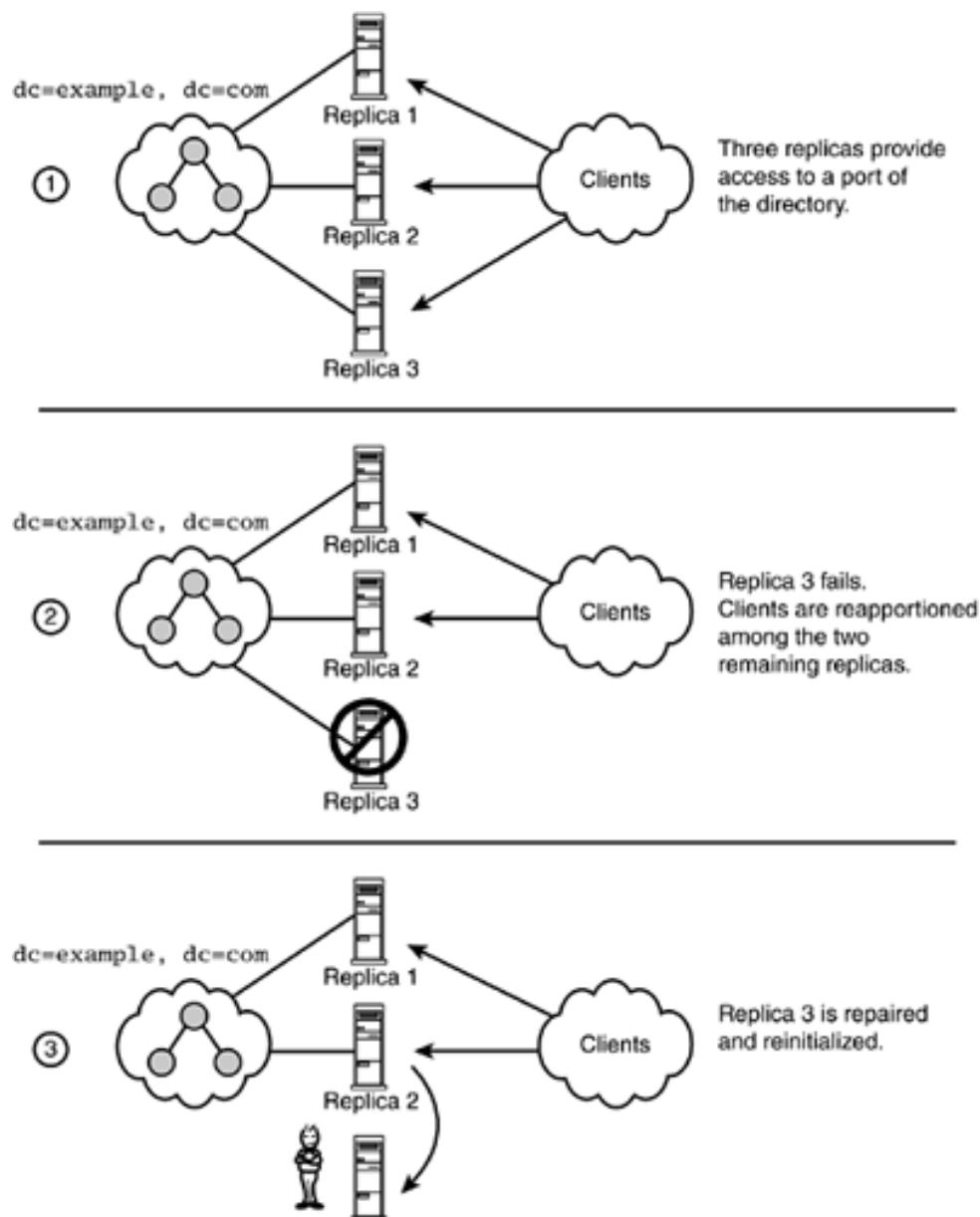
Although traditional backup techniques can protect you against many types of problems, they have one major drawback: Restoration of data is time-consuming. Copying data from the backup media to the server may take many minutes or even hours for a large directory. However, if you use replication as your primary means of providing redundancy and fault tolerance, you can avoid the costly downtime that would be required for the restoration phase.

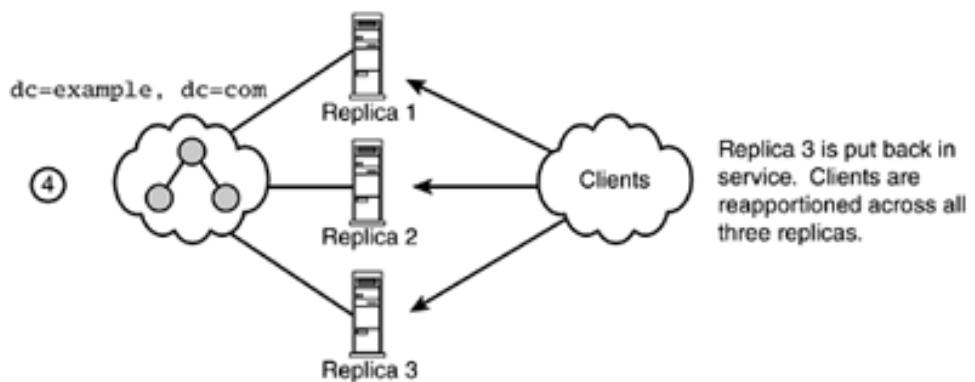
Because replicas are online copies of your critical directory data, no delay is incurred while data is being restored from backup media. If a server fails, you can simply remove it from the set of replicas and repair the failure. After the server is repaired, you can bring it back online and re-establish it as a replica. Users will generally be unaware of the problem, as long as there is sufficient capacity across the remaining replicas to handle the client load.

Using replication for backup and restoration has another advantage: Directory data is usually more up-to-date on replicas than on backup tapes. Of course, most directories support loose replica consistency; that is, it's possible for changes to be held on a replica for some time before being propagated to other replicas. Thus there is no absolute guarantee that all replicas are completely in sync at any given time.

Directory server software that supports multimaster replication makes this backup procedure simple. Because any server can accept updates in such a configuration, there is no loss of functionality when a server is missing from the replica set. [Figure 17.1](#) shows the process of repairing a failed server in a multimaster environment. In the situation depicted, Replica 3 experiences a failure of its disk drive. The server is removed from the set of replicas, and the disk drive is replaced. The replica is returned to the set of replicas as soon as its directory data is reinitialized from one of the other replicas.

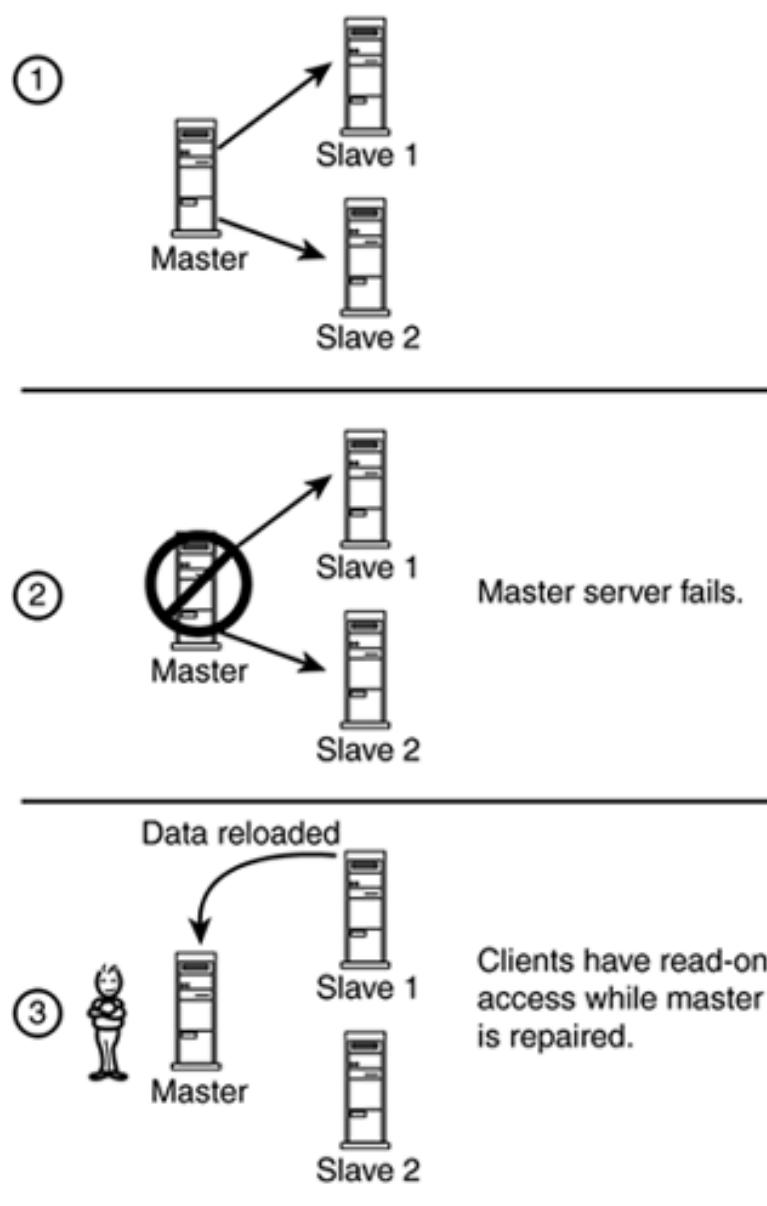
Figure 17.1. Multimaster Replication Provides Protection against Server Failure





If your directory server software supports only single-master replication, the backup process can become more complicated. In single-master replication, only the master server can be updated; all other servers are read-only copies. If the master server fails, no updates can be processed by the directory until the master is repaired and brought back online (see [Figure 17.2](#)), or until a different server is designated as the master (see [Figure 17.3](#)).

Figure 17.2. Single-Master Replication: A Master Server Fails, Is Repaired, and Is Brought Back Online



Slave 2

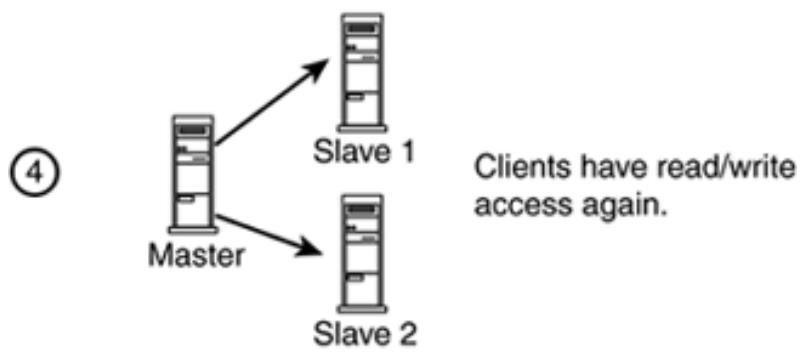
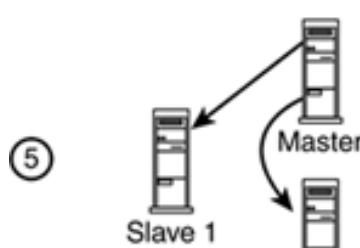
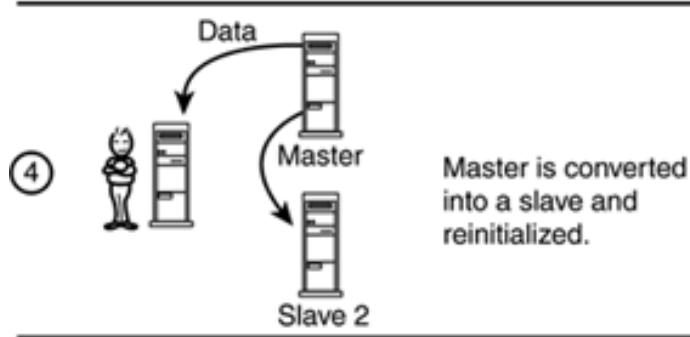
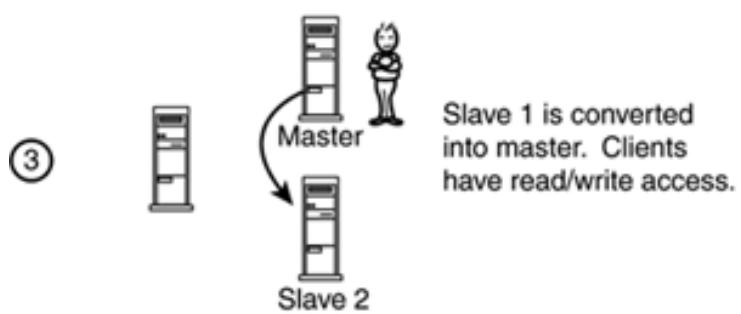
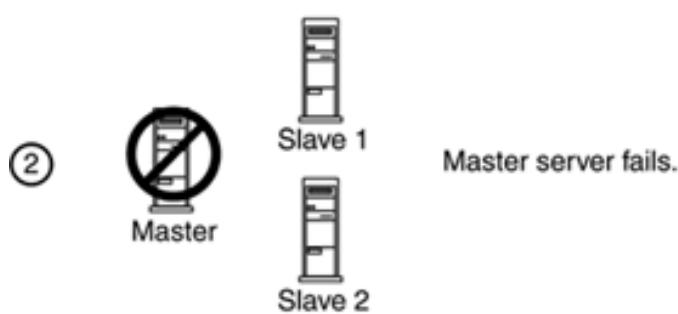
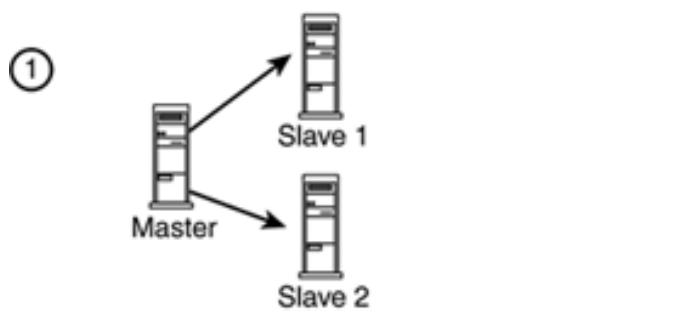
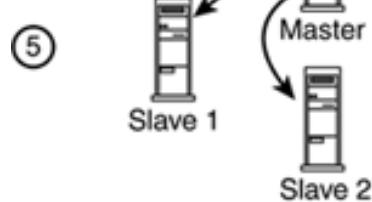


Figure 17.3. Single-Master Replication: A Master Server Fails, and a Slave Server Is Converted into a Master





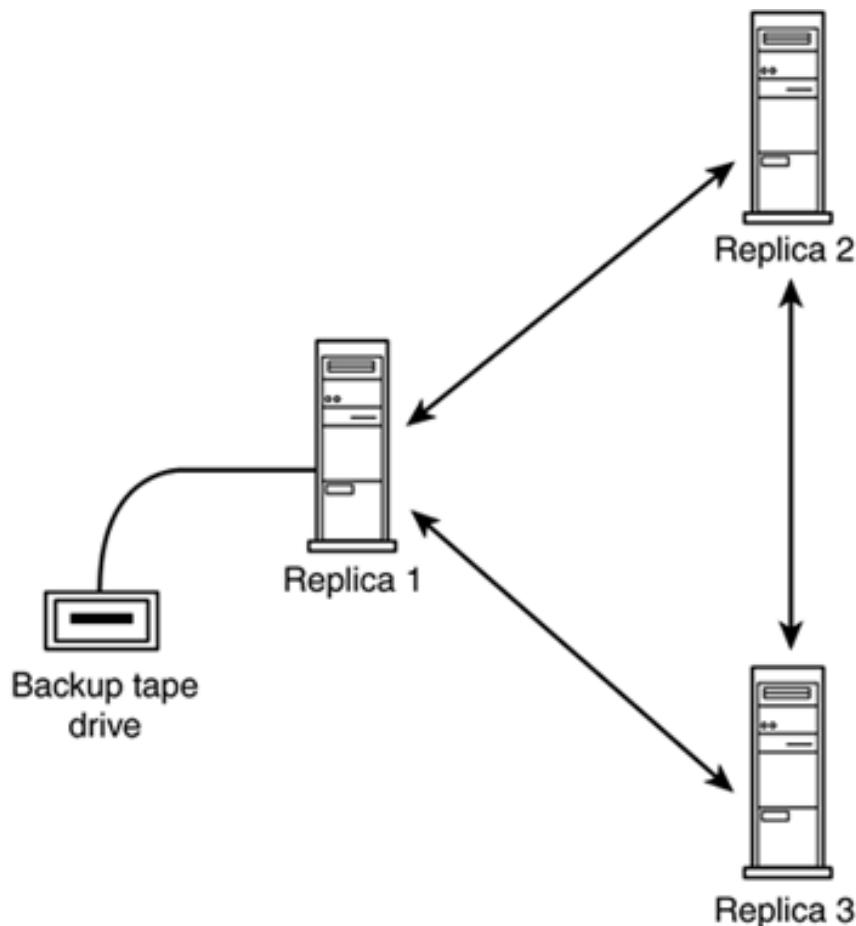
Converting a slave server to a master can be complicated. At the very least, the new master must be configured to send replication updates to all the consumer servers covered by the failed master (excluding the new master). In addition, it may be necessary to update state information stored in each replica, such as the last update number received from the master, to reflect the update numbers on the new master. Because this can be complicated, make sure you understand the process thoroughly. For more information, consult your directory server software manual.

Using Replication and Traditional Backup Techniques Together

Although replication provides redundancy and high availability in the event of a single server failure, it cannot protect you from incorrect data being placed in your directory. For example, if an automatic data update process begins erroneously deleting entries from your directory, replication unfortunately ensures that these incorrect updates end up on all your replicas! To protect yourself from this type of problem, periodically back up directory data to media such as magnetic tape.

[Figure 17.4](#) shows a hybrid approach that uses both backup methods. One of the replicas is equipped with a tape backup unit, which is used to back up the directory contents periodically.

Figure 17.4. A Hybrid Approach to Backup, Incorporating Both Replication and Traditional Backup Techniques



Safeguarding Your Backups

It's important to keep your backup media in a safe place. This means protecting them from damaging environments such as extreme temperatures and magnetic fields. You should also keep copies of your backups off-site. In the event of a disaster such as a fire, the off-site backups can be used to restore

data to replacement servers. You can outsource the transportation and storage of off-site backups to reduce costs. We'll discuss disasters and disaster recovery in detail shortly.

Another important aspect of safeguarding your backups is the physical security of the backup media storage facility. No matter how good your privacy and security design, it is all for naught if someone can just stroll up to the cabinet holding your backup tapes and walk off with a copy of your sensitive directory data. Backups should be well secured; this might mean placing them in a locked, fireproof vault in an off-site facility and using a bonded courier service to transport them.

Finally, consider storing your off-site backups in a location unlikely to be affected by a natural disaster that damages or destroys your main location. For example, an off-site backup storage facility ten miles away may be adequate in the midwestern United States, where the primary natural disaster is severe weather. In California, however, where the primary concern is earthquakes, it may be prudent to store off-site backups at a more distant location, in a region that is less seismically active.

Verifying Your Backups

Backing up your critical directory data is important. However, the act of backing up is futile if the backups can't be restored. You need to take steps to check the integrity of the backups you produce. At the very least, check that your backup media are readable and free of media errors.

How you verify your backups depends on the server software you use. One option is to restore the backup onto a test server, start the server, and then verify that the content of the server is correct. This approach may not always be practical, however. For example, Novell eDirectory copies data from the entire set of distributed servers onto a single backup. Restoring the directory to a test server may be impractical because of the number of partitions in the directory; it may be difficult to fit them all onto a temporary server used for verification. If you run Novell eDirectory, consult your backup software documentation for instructions on verifying backups.

Verify backups immediately after they complete. If the verification fails, the cause of the failure should be determined and fixed, and another backup should be performed immediately. You may also want to incorporate your backup process into your monitoring system so that you'll know if something goes wrong.

Be especially careful to verify your backups fully when you initially develop and deploy your backup procedure or change it in any significant way. Like any other software, backup software can contain bugs. It's always better to discover this, and other flaws in your backup procedures, before you need to restore critical directory data from your backup.

Disaster Planning and Recovery

Businesses are becoming increasingly dependent on directories, to the point where they will simply be unable to function if the directory becomes unavailable. What happens if the data-processing center of your organization falls victim to a fire, a flood, or sabotage? How will you restore critical directory services?

Developing a comprehensive recovery plan is the best way to anticipate and prepare for business continuity in the face of a disaster. Numerous books cover this subject, and there are some disaster recovery vendors (also known as *business continuity services*), including SunGard Availability Services and IBM Business Continuity and Recovery Services. These vendors and others like them can help you plan and implement disaster recovery procedures.

In this section we provide a brief overview of disasters and disaster recovery planning, and we discuss how planning for directory disasters differs from other types of disaster planning. Then we discuss specific issues that can help you design and implement your directory disaster recovery plan.

Types of Disasters

A disaster is any occurrence that destroys your computing infrastructure or makes it inaccessible for an extended period. Examples include the following:

- Fires
- Severe weather, such as hurricanes, tornadoes, and severe storms
- Earthquakes
- Extended electrical power outages
- Hardware or software errors that cannot be fixed within a reasonable time period
- Floods
- Burst pipes
- Explosions
- Chemical spills
- Airplane crashes
- Riots
- Sabotage
- Security breaches
- Terrorist attacks

Developing a Directory Disaster Recovery Plan

When you develop a plan for directory disaster recovery, follow a methodical process similar to this:

- Step 1.** Perform a risk assessment, ranking risks from most likely to least likely.
- Step 2.** Understand the business implications of each type of risk.
- Step 3.** Design and implement the recovery solution.
- Step 4.** Periodically review and update the plan.

Each of these steps is explained in detail in the following sections.

Step 1: Perform a Risk Assessment, and Rank the Risks from Most Likely to Least Likely

When planning for disaster recovery, the first questions to ask yourself are, What risks does the computing infrastructure face? and How likely is each risk? For example, if your data center is located on a hill, flooding is probably unlikely. On the other hand, if the data center is in the San Francisco Bay area, earthquakes are a risk that must be taken into account.

Ranking your risks allows you to make rational decisions about whether you should attempt to protect against a particular risk. Although it would be nice to provide protection against every conceivable risk, it's probably not economically possible. Understanding which risks are more important to address allows you to allocate your disaster preparedness resources wisely.

For some risks you may decide that preemptive measures are appropriate. For example, if your location is subject to frequent electrical storms and power failures, you may decide to invest in a generator that can provide an alternate power source during extended power failures.

Step 2: Understand the Business Implications of Each Type of Risk

For each type of disaster, think through its implications and how it will affect your organization's business processes. For example, assume that your directory service is destroyed by a fire and that it takes three business days to obtain replacement hardware and restore the directory data from backup tapes. What are the business implications of this three-day delay? What business processes are halted by the unavailability of the directory? Such impeded processes might include

- Delivery of e-mail. (Inbound mail from the Internet will typically be returned to the sender if your electronic mail servers are continuously unavailable for three or more days.)
- Any processes that depend on the timely delivery of e-mail.
- Login to your intranet and extranet applications.
- Any intranet or extranet applications that use the directory for authentication.

Next you need to understand the implications of directory failure—and what they mean for your bottom line. In other words, if the directory is unavailable, how much money will the business lose as a direct consequence of the failure? Will customers switch to an alternate vendor because you cannot provide the goods or services they require? Are there contractual obligations that you must meet even in the face of a disaster? With this information you can determine the recovery times you need to target.

For example, you might determine that the maximum acceptable directory downtime is 24 hours before the business begins to suffer significant losses. When you know the potential costs of *not* having a disaster recovery solution, you can begin to weigh them against the costs of a recovery solution.

Step 3: Design and Implement the Recovery Solution

The next step is to design the actual recovery solution and understand its costs. You can design and implement the recovery plan yourself, or you can use the services of a disaster recovery vendor to design and/or implement the plan.

Disaster recovery vendors typically offer both "hot" and "cold" recovery solutions. A *hot site* is kept up-to-date with your latest data and application software, and it can be put into service quickly. A *cold site* contains sufficient equipment to meet your computing needs, but

it is not kept up-to-date; your computing environment must instead be re-created at the cold site after the disaster recovery plan goes into effect. Disaster recovery vendors also offer mobile recovery solutions, in which a portable data center can be driven to your site in the event of a disaster.

Hot sites and cold sites each have advantages and disadvantages. Whereas a hot site can be put into service relatively quickly because all the data is up-to-date and ready to go, with a cold site data must be transported to the remote site, the needed software must be installed, and the data must be restored. As you might expect, it is much more expensive to maintain a hot site, especially if you contract with a disaster recovery provider. Assuming that not too many customers experience simultaneous disasters, a disaster recovery provider can use a single cold site to support multiple businesses, thereby lowering the cost for the customers. A hot site, on the other hand, must be dedicated to a single customer, which makes it much more expensive.

Consider also what happens if your directory data is damaged, perhaps by a malicious hacker. If the bad data has replicated everywhere, your only recourse may be to restore from the most recent backup. In this case it doesn't matter whether you have a hot site standing by. You need to close the security hole that allowed the hacker access, find and remove any backdoor access methods that may have been installed, and restore your data from the most recent backup. More information on handling security breaches can be found in [Chapter 20](#), Troubleshooting.

Step 4: Periodically Review and Update the Plan

Finally, after the recovery plan is implemented, it must be periodically reviewed and updated as your business requirements change and as new applications are developed and old ones retired. The plan should be reviewed at least annually, and more often if your organization deploys new applications frequently.

The disaster recovery procedures should also be tested and repaired if they are found not to work anymore. Some organizations even go so far as to simulate a disaster to exercise the recovery procedures. Your disaster recovery tests should be in line with your disaster recovery needs; very stringent needs dictate more rigorous testing.

Directory-Specific Issues in Disaster Recovery

A cold-site disaster recovery plan for a directory is much like that for the other critical data repositories you might use. The recovery procedures are, for the most part, like recovering a database. If you use a cold site, directory server software must be installed on the replacement machines, and backup tapes containing the directory data must be restored.

With hot-site disaster recovery, however, directory replication offers unique opportunities to improve recovery time significantly. By using replication to maintain the hot standby servers, you can make your directory available much more quickly. For example, with hot standby replicas no recovery at all is necessary; because the replicas are always updated, they are ready to take over at a moment's notice.

This does *not* mean, however, that it's impossible to lose transactions when replication is being used. A server at the primary site certainly could be destroyed before having a chance to transmit all its replication updates to its peers. Directories almost always use loosely consistent replication, as discussed in [Chapter 11](#), Replication Design. To minimize data loss due to replication latency, configure your server to replicate changes immediately.

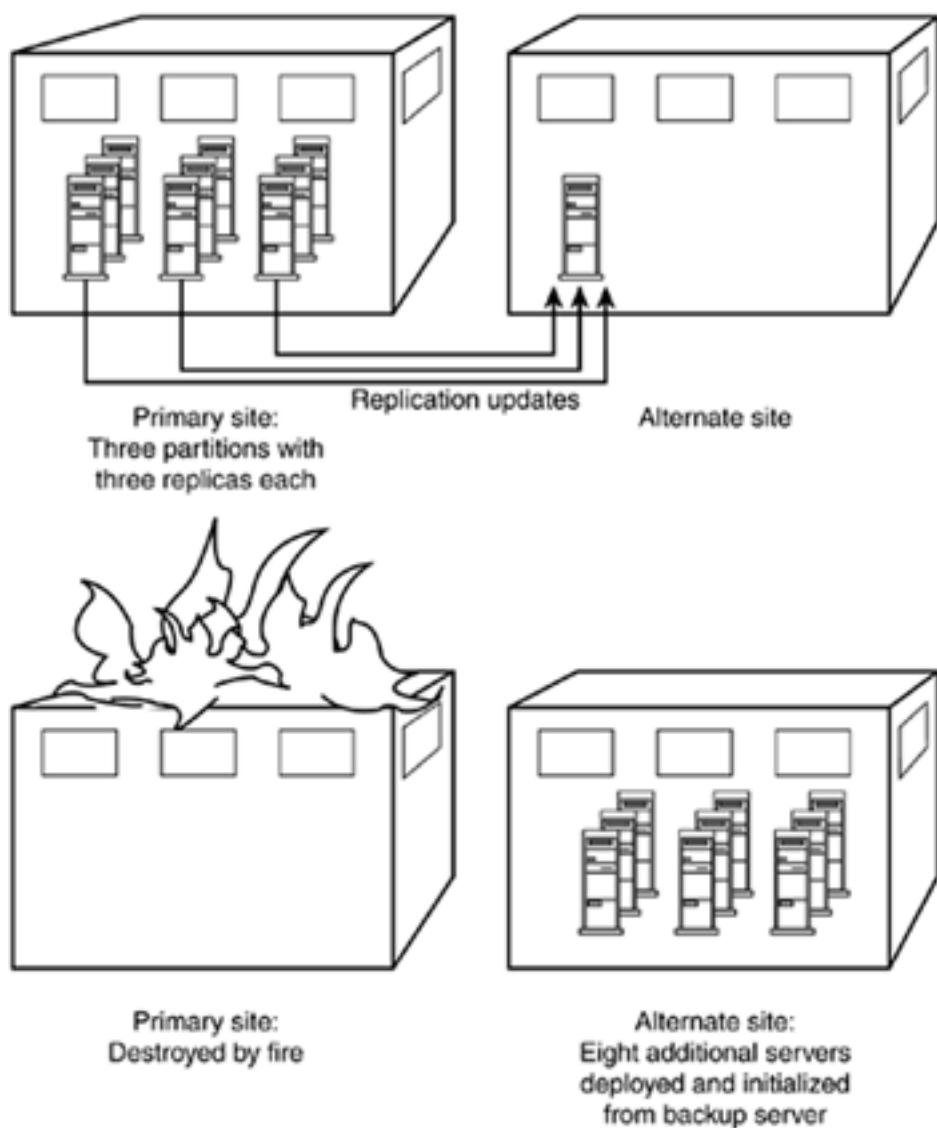
One issue that arises with replication is the reassignment of master servers. Directory server software that supports only single-master replication requires additional steps to recover from the destruction of the master server. In such a case, another server must be configured to be the updatable master and to supply updates to the backup servers at the alternate site. The recovery plan must be modified to include time for these steps if necessary.

Another issue to consider is the distributed nature of directories. Recall that a single directory tree can be spread across multiple servers, each holding a portion of the tree. To provide a backup set of servers, at either a cold or a hot site, you need to either provide equivalent computing power (number of servers, speed of CPUs, and so on) or make a conscious decision that the standby site will be limited in some fashion.

For example, at your primary site your directory may be divided into three partitions, each residing on a separate server. The backup site, however, might have all three partitions on the same server. Although the backup site would have less capacity than the primary site, you might choose this approach to limit costs. Reduced capacity may be perfectly acceptable anyway, especially if you can reduce the load on the directory by shutting down less important directory-enabled applications while the backup site is in operation. However, the backup site must meet at least some minimum performance requirements for it to be useful at all.

You can, of course, install additional replicas at the backup site after disaster recovery commences. Doing so would reduce the amount of replication update traffic that must flow from your primary site to your backup site under normal conditions. For example, if your primary site consists of three directory partitions replicated three times each (a total of nine servers), a complete hot standby site would also have nine servers. Maintaining the backup replicas might significantly increase the workload of the servers at the primary site, however. If instead you maintained a single server at the hot site that held all three partitions, replication update traffic to the remote site would be reduced significantly. Additional replicas could be rebuilt from the single server at the hot site in the event of a disaster at the primary site. This might be considered a hybrid hot/cold site approach: The online replica at the remote site is hot, and the other servers are cold, put into service only if a disaster occurs (see [Figure 17.5](#)).

Figure 17.5. A Hybrid Hot/Cold Strategy



Backups and Disaster Recovery Checklist

To recap, here are the important things to consider when planning your backup and disaster recovery strategies:

- Select a backup strategy appropriate to the criticality of the data in your directory and your directory server software.
- Consider using replication as your primary means of protecting against catastrophic equipment failures.
- Even if replication is used to provide backup, traditional procedures such as backing up directory data to magnetic tape are still needed to protect against erroneous data update procedures that destroy directory data.
- It is critical to safeguard your backups from damage and to ensure the security of the data they contain.
- Verify backups to ensure that the data they contain is valid and can be successfully restored.
- Develop a disaster plan appropriate to your business needs and the types of disasters you may encounter.
- Disaster recovery services can be provided in-house or purchased from a disaster recovery vendor (sometimes also called a business continuity service). In either case, costs of a disaster recovery plan are directly related to the speed with which services can be restored.

Further Reading

Disaster Recovery Journal. Available on the World Wide Web at <http://www.drj.com>.

Disaster Recovery Planning: Strategies for Protecting Critical Information Assets. J. W. Toigo and M. R. Toigo, Prentice Hall PTR, 1999.

IBM Business Continuity and Recovery Services. Information available on the World Wide Web at <http://www-1.ibm.com/services/continuity/recover1.nsf/documents/home>.

Netscape Directory Server 6 Administrator's Guide. Available on the World Wide Web at <http://enterprise.netscape.com/docs/directory>.

SunGard Availability Services. Information available on the World Wide Web at <http://www.recovery.sungard.com>.

Windows NT Backup & Recovery. J. McMains and W. Rinaldi, Osborne/McGraw-Hill, 1998.

Looking Ahead

In this chapter we have described how you can protect the data in your directory from damage resulting from equipment failure, software failure, and disaster. In [Chapter 18](#), Maintaining Data, we will turn our attention to the topics of managing data, ensuring its timeliness and accuracy, and incorporating data from new sources.

Chapter 18. Maintaining Data

- The Importance of Data Maintenance
- The Data Maintenance Policy
- Handling New Data Sources
- Handling Exceptions
- Checking Data Quality
- Maintaining Data Checklist
- Further Reading
- Looking Ahead

Data maintenance refers to the policies and procedures used to keep the data in your directory up-to-date and accurate. Data maintenance is one of the most important tasks of maintaining your directory service. Your directory service is all about providing access to data; if that data is inaccurate or of poor quality, the quality of your entire service is reduced, users are disappointed, and directory-enabled applications may function incorrectly or not at all.

You will also find that without proper planning, data maintenance can become one of the most expensive tasks involved in maintaining your directory service. Failure to implement automated data maintenance procedures can cost hundreds of thousands of dollars in staff time spent updating data and resolving problems. The loss of productivity and user satisfaction caused by inaccurate data in the directory is often expensive.

This chapter describes the process of maintaining your data. We begin with a section explaining in more detail why data maintenance is so important. Then we describe how to formulate your data maintenance policy. Afterward, we discuss how to manage data maintained both centrally and by your users. Next we explain how to check the quality of your data. Finally, we explore what happens when you need to extend your directory with data from a new source.

The Importance of Data Maintenance

A directory without good data is virtually worthless. Worse yet, bad data can cause real harm. Imagine a directory providing a white pages service that users consult to find contact information of friends and colleagues. If the information they retrieve is not accurate, they will soon stop using the service. Or imagine a directory providing authentication, authorization, and personalization services for a suite of e-commerce applications. The directory contains information specifying which users are allowed to access the applications and what their preferences are. If the information is not accurate, there is not much use in consulting the directory in the first place.

More serious consequences can result from inaccurate data as greater numbers of mission-critical applications use the directory. For example, consider an internal corporate directory used to control access to vital corporate resources. An employee's termination is supposed to be reflected in the directory and his access to resources instantly revoked. Failure to provide accuracy in this case can open your organization to sabotage by disgruntled ex-employees, corporate espionage, and the like.

Poor data maintenance can also lead to user distrust and dissatisfaction with the directory service. This dissatisfaction is especially likely if your directory contains information about users. Whereas users may be disappointed to find incorrect information about others in the directory, they can become downright angry if they find incorrect information about themselves.

The potential for dire consequences that can result from inaccurate data in the directory can be used to your advantage. The very fact that the consequences are bad should be sufficient motivation to keep the data up-to-date. Later in this chapter we show you how to use this motivation to improve the quality of your directory data by distributing the authority to update it. Ultimately, you may want to allow users to update their own entries for some kinds of information, or you may want to empower a small group of directory content administrators to maintain directory data.

The Data Maintenance Policy

In [Chapter 7](#), Data Design, you learned how to identify, locate, and obtain the data needed to populate your directory. The result of that phase of your design process was a table listing all your data sources, the data you need from each source, and the procedures you will use to obtain the data. This information plays an important role in formulating your data maintenance policy.

Your data maintenance policy determines who is responsible for maintaining which attributes in the directory. If more than one entity is allowed to maintain an attribute, your data maintenance policy determines how conflicts are resolved. For example, suppose that users and the human resources databases are both allowed to update the `telephoneNumber` attribute. What do you do if both sources update the attribute with different values at the same time? Your data maintenance policy should outline procedures to determine the answer to this question. It should also determine the frequency of updates for pieces of information in the directory and the security those updates require.

Another important procedure determined by your data maintenance policy is how exceptions are handled. Every policy has exceptions. Do not fool yourself into thinking your data maintenance policy is the exception to this rule.

There are potentially as many reasons to make exceptions to your data maintenance policy as there are users of your directory. For example, consider an operation that obtains home address information from the official payroll database. Although this might be just fine for the vast majority of your employees, consider an employee who has his checks mailed to a location different from his home. Including this mailing address in the employee's home contact information would be just plain wrong. You may want to make an exception for this user.

How you handle data maintenance exceptions will have a great impact on the cost of maintaining your directory service. Consider the number of exceptions in relation to the projected size of your directory. Is the policy you choose applicable to 90 percent of your users? 99 percent? all of them? Make some educated guesses here, and realize that the larger your directory becomes, the more important it is that the policy be nearly universal.

For example, consider a small directory containing entries for 100 people. If your policy correctly covers 99 out of these 100 users, it is a relatively small burden to do something special for the remaining person. On the other hand, if your directory supports an e-commerce application and contains 1 million entries, 99 percent coverage is pretty close to a disaster: You would have to make special exceptions for 10,000 people, a task that can become arduous and expensive.

Consider automating the exception process itself. This may sound counterintuitive, but it is possible in some situations. For example, you might allow users to modify their own entries in a way that exempts them from the standard data policy. One way to do this would be to have the user set a flag in his or her entry. The flag would then be read by the automated data maintenance procedure, prompting special handling of the user's entry.

For the purposes of this chapter, we separate the attributes in your directory into six categories:

1. **Attributes maintained by directory administrators.** These kinds of attributes might include access control attributes, password policy information, or other attributes used in operating the directory.

2. **Attributes maintained by directory content administrators.** These attributes might include account information maintained by system administrators of other systems or services. They might be maintained by your Help Desk, departmental administrative assistants, or other agents acting as proxies for other users.
3. **Attributes maintained by official data sources.** By "official," we mean corporate data sources maintained by your organization's human resources, finance, or other departments. These attributes might include official name, work telephone number, employee identification number, title, salary, manager, and other attributes your organization maintains in other databases.
4. **Attributes maintained by directory end users.** These attributes might include home address and telephone number, description, picture, and other attributes containing personal information about the user. For an e-commerce application with self-serve account creation, users may maintain all their attributes.
5. **Attributes maintained by directory-enabled applications.** These attributes might include application-specific preference information, application or user state information, or attributes shared across multiple instances of the application.
6. **Attributes maintained by the directory service itself.** This category includes attributes such as `modifiersName`, `modifyTimeStamp`, and others that the directory maintains either as a convenience or to ensure its proper operation.

You probably don't need to worry about attributes that the directory itself maintains. Application-maintained data is not up to you to maintain, but it's important that you monitor and approve how applications use the directory. In the following discussion we also consider centrally maintained data—that is, data maintained by administrators and other data sources—and user-maintained data.

Application-Maintained Data

Data maintained by directory-enabled applications may end up being the majority of the data in your directory. The health of this kind of data and the applications that maintain it can have a tremendous effect on the quality and performance of your directory service. Make sure that the directory-enabled applications that you or others develop use the directory in an efficient and sensible way. Be proactive when you do this; don't wait for application developers to come to you.

There are many aspects of efficient and high-performance directory-enabled application development. A tutorial on directory-enabled application development is beyond the scope of this book, but the following list highlights some of the more important things to remember. Following these tips will go a long way toward making any directory-enabled application perform better:

- **Minimize connections.** Both your application and your directory service will benefit if you open a connection only once and reuse it for many operations. A search request, with a single network round-trip to the directory server, can often be processed in only a few milliseconds. Opening a connection to perform the same operation can take much longer and consume far more network and server resources.
- **Perform only efficient searches.** The capabilities of your directory software and how you configure it determine what kinds of searches your directory can handle efficiently. The difference in response time between a search your directory can

respond to efficiently and one it can't is often measured in minutes. Efficient searches are important both for reducing the response time perceived by the application making the query and for increasing the overall throughput provided by your directory server. Clearly, a balance must be reached between application developers and directory administrators. Sometimes application developers can be shown a better way to do things; at other times, directory administrators may be able to reconfigure the server to better serve an application's requests.

- **Minimize the number of searches.** Application developers often don't think in terms of consolidating operations for efficiency. As a result, multiple searches might be performed when only one would do. For example, if neighboring parts of the code call for an application to recover the `mail` and `title` attributes, an application developer might make two searches—one for each attribute. A more efficient approach would be to do one search asking for both attributes at the same time.
- **Retrieve only required attributes.** Another area often ripe for improvement is the number of unnecessary attributes retrieved by an application. Sometimes developers are tempted to retrieve attributes they don't really need, perhaps even every attribute in the entry. In some directory installations, this can be a lot of data! For example, you might maintain a JPEG photo or audio greeting attribute that is tens of thousands of bytes long. Transmitting such attributes needlessly wastes bandwidth and computing power and can have severe performance implications because of access control and other processing overhead. It's better to ask only for those attributes the application will use.
- **Minimize updates.** As discussed in [Chapter 1](#), Directory Services Overview and History, a defining characteristic of a directory is that it is better equipped to handle read operations than update operations. Typical directory implementations can handle several orders of magnitude more reads and searches than writes. The less writing an application does to the directory, the better performance everyone accessing the directory will experience. It's also important to make updates as efficient as possible. Encourage application developers to change only the minimum information necessary. A common mistake is to perform a modification by deleting the old entry and adding the modified entry even if only one attribute value in the entry has changed. This kind of behavior can make updates substantially less efficient.

There are many techniques for communicating these guidelines to application developers. Among the more effective methods we've found are the following:

- **Documentation.** If you simply document and publicize good practices, developers will often follow them. You might do this in the form of printed or online documentation.
- **Training and seminars.** Most inefficiencies in the way directory-enabled applications use the directory are simply due to a lack of education. Documentation can help, but sometimes developing a formal training course is called for.
- **Sample code.** Whatever vehicle you use to distribute guidelines to developers, be sure to include plenty of sample code. Not only does providing sample code put things in a language developers can understand, but it also makes incorporation into an application easier.
- **Consulting.** In some situations it may be worthwhile to provide one-on-one consulting to application developers. You might do this during the application's design phase, during development when you can act as a support resource, and

during application testing when you can help fix any last-minute problems.

- **Laboratory testing.** One good way to root out and correct potential problems is to host application developers and their applications in your testing laboratory. This is good for developers because it gives them a chance to test their application in a controlled, easily monitored environment. It's good for you and your directory service because it gives you a chance to correct bad behavior before it is unleashed on your production service.

You may not have the resources to implement all these techniques. Think about the kind of community you're dealing with and which techniques will give you the biggest bang for your buck. Make sure you document the policy to which applications must conform. This documentation will give you an objective tool to use in judging and correcting applications.

Centrally Maintained Data

The first thing to decide about centrally maintained data is who is the central authority responsible for the maintenance. Options include the directory administrator or a third party. In either case you'll need to think about how the data is maintained, the frequency of updates, the effect updates will have on the operation of the directory, and other issues. These choices are summarized in the following list:

- **Online or offline update.** Do the updates come in over LDAP, or do they come in via a file or other format imported into the directory service by other means? Typically, such imports go directly into the underlying directory database. If possible, have the updates come in over LDAP. That way, you can use the normal LDAP security mechanisms to protect the directory and avoid taking the directory down.
- **Update security.** If the information being updated is sensitive and has to travel over insecure connections on its way to the directory, you will need to consider how to protect it. You might use Secure Sockets Layer (SSL) or Transport Layer Security (TLS) to protect updates that come in over LDAP. Other solutions, such as Secure Shell (SSH), are available to protect non-LDAP data transfers.
- **Update process.** If the update process is regular, you should almost certainly automate it as much as you can. If you can't, you need to consider alternatives. Who performs the updates? What kind of training do they need? What kind of support do you need to provide for them?
- **Exception handling.** Any process, automated or not, has exceptions. Depending on the size of your database and the number of updates you process, these exceptions can be significant. Think about making exception handling as automated and low-cost as possible.
- **Update frequency.** The frequency at which data is updated depends on many factors, including the volatility of the data, the timeliness required, and the consequences of out-of-date data. Updates should be scheduled with care to avoid affecting the service and inconveniencing users.
- **Data validation.** You may be surprised at the low quality of data that creeps into some of the data sources around your organization. Use the directory update process as an opportunity to improve the quality of this data. Higher quality of the data will improve the quality of your directory service and serve as an incentive to the data source to increase the quality of its data.

Make sure that you fully understand the implications of each of your choices. For example, opting for an offline process may involve shutting down the directory during the update. Depending on the update frequency, shutting down may or may not be acceptable. Opting for an online updating process, however, may degrade the performance of the directory. An online updating process may not complete as quickly as an offline process.

Update security may have implications as well. Can you arrange to give the updating entity access to only the fields it is allowed to update? Or do you need to give it more access than it really should have? If the update is accomplished offline, consider the implications for the security of your system. For example, you may need to provide physical access to the machine. If this is not acceptable, you may need to act as the update agent.

You'll probably want to develop as automated a process as possible to save staff costs. How much will the development of this system cost you? If you opt to protect the security of online updates via technologies such as SSL or TLS, consider the effect that this process will have on your service. How much will service be degraded? Will you need hardware acceleration to get acceptable performance? These considerations are all implications of the update process.

Update frequency is another choice that can have many implications. There is a conflict between wanting the data in your directory to be as up-to-date as possible and wanting the service to always be up and performing at peak capacity. For example, an offline update process may force you to take the system down while you import data. This is typically the fastest way to get data into your directory, but it means that the server being updated is totally inaccessible during the update. This lack of access during updates limits how often you can perform such an update. Extracting update data from the data source might also be an expensive process that cannot be performed often.

For an online update process, different concerns must be addressed. Depending on the number and complexity of updates being applied, online updates can significantly degrade your directory's performance. For example, consider updates from a database with 100,000 entries, each of which changes once an hour. Keeping absolutely up-to-date with these changes requires your directory to process 100,000 updates per hour. Multiply that by the number of replicas in your system; depending on the capabilities of your directory software and the load of other queries on the system, this may be too much. A more prudent approach may be to update the directory only once a day or even once a week. Make sure that you understand how up-to-date the information needs to be to be useful.

In general, we have found that the following principles help make centralized update processes safe and efficient:

- **Automate as much as possible.** Automation reduces staff costs for updating, which can be one of the most expensive aspects of running your directory service.
- **Be prepared to handle exceptions and errors.** No process, no matter how foolproof it seems, will be able to handle everything. Make sure that you train a group of people to field exception reports and take appropriate action.
- **Use an online update process whenever possible.** Updating online maximizes directory availability and allows you to use the directory's built-in security features to protect the updates.
- **Keep logs so that you can figure out what went wrong.** Logs are invaluable when you are first developing and testing the system. Keeping detailed logs of the update process will continue to be valuable as your service moves into production.

Here's one final word of advice: You can save yourself a lot of work and potential for trouble if you avoid centralized update processes altogether. The closer you can push update responsibility to the owners of the data being updated—be they human resource administrators or end users—the better off you will be.

User-Maintained Data

Some data contained in your directory is best maintained by the users to which it pertains. This kind of data includes things such as home address and phone number, vehicle license number, user-owned mailing lists, and other information that the user has the most interest in seeing up-to-date. Having users update this information themselves can be a benefit to both administrators and users for the following reasons:

- **Accuracy of the data.** For many kinds of user information, the most accurate source is the users themselves. This is true of information such as home addresses and telephone numbers, which users know better than you do. It's also true of other information—such as title, salary, manager, and so on—that you might think would be better known by corporate sources. Rarely do you want to allow users to change this information, but you should make it easy for them to request that changes be made.
- **Less work for you.** There is no question that maintaining data can be a labor-intensive process; even automated systems do not relieve you of all this burden. However, these systems must be developed, maintained, and monitored, so the more you can distribute this task across your entire user population, the better.
- **Empowerment of your users.** The more empowered users feel, the happier they are likely to be. This is especially true today because users are inundated with information. Personal information is maintained in literally dozens, if not hundreds, of databases across the globe, usually with little or no opportunity for it to be corrected when it's wrong. Although your directory service cannot address the larger problem, it can provide users with the opportunity and the means to correct mistakes they notice. Make sure that you ask users to update only the data for which they are the best source.

Despite these advantages, there are still problems to be overcome before you enable your users to update their own information. Although user updating of data has many long-term benefits to you, your directory service, and your users, you will have to make an investment up front to enable this process. In most cases, however, this investment is well worth the cost.

User-maintained data is not a panacea; it comes with its own set of problems. For example, users often do not have the motivation or expertise required to perform updates. They might not realize the consequences of out-of-date information, or they might not have the time to take care of it. You will probably want to take control of information that you deem critically important to providing a good directory service. Another good approach is to train a small group of responsible users, such as administrative assistants or system administrators, to perform the updates.

Update-Capable Clients

The first thing a user requires to update his own entry is a client capable of performing the update. Depending on the type and capabilities of the service you have deployed, such clients may already be available. If not, consider three alternatives:

1. **General-purpose client.** A general-purpose client can be one of the many generic directory clients on the market. The advantage is that such clients come ready to go, requiring of you only configuration, distribution, and training. The costs involved can still be substantial, however. Make sure that the client you select has the functionality you require but is not so general-purpose as to be overwhelming to users. Users may find it frustrating if the client has capabilities they cannot use because of access control or other restrictions. Sometimes clients allow configuration changes to remove such extra options.
2. **Special-purpose client.** This might be an application you distribute to each user's desktop or a centralized Web application accessible from any Web browser. The latter option is attractive in terms of development and distribution costs.
3. **Online request system.** With this option, users don't directly make changes to their entries in real time. Instead they access the online request system to submit their changes. The request is then handled by administrative staff or an automated offline process. Such changes may be applied to the directory itself or to the source databases (an attractive feature). Another advantage is the human verification of update requests that can be performed if this method is being used. These advantages need to be weighed against the lack of instant update capability, which may confuse users.

See [Chapter 21](#), Developing New Applications, for tips and techniques you can use to develop some of these applications yourself.

Authentication and Security

If your users have an update-capable client in their hands, they'll need to authenticate to the directory before it allows them to make changes. This is an important step toward protecting the security and integrity of information in the directory.

[Chapter 12](#), Privacy and Security Design, discussed options for authentication in detail, so by this point you should already have an authentication and security plan. Part of your data maintenance plan must include provisions for distributing and maintaining authentication credentials. These provisions might involve passwords, certificates, or another form of credentials.

As soon as users have authenticated, you need to be concerned about the security implications of the updates they make. Again, this topic was covered in detail in [Chapter 12](#). We will not repeat that discussion here, except to recall that different kinds of information require different kinds of protection. For example, public information needs its integrity protected, but privacy is not generally a concern. Sensitive information needs both privacy and integrity protection.

You should also apply access controls to your directory, again as described in [Chapter 12](#). These controls ensure that users are able to update the fields you want them to update. Even more importantly, access controls ensure that users are unable to update data that you don't want them to update.

It's often important to maintain an audit trail when allowing users to update their own entries. Problems are inevitable, and you need to be able to distinguish among user errors, directory system errors, and security breaches. Maintaining an adequate audit trail showing who updated what and when they did it can go a long way toward resolving these problems. Most directory server software supports some kind of audit capability.

Training and Support Costs

One of the few downsides of allowing users to update their own entries is the extra cost of training and support that may be required. You need to balance the extra cost against the savings you get by not having to spend administrative staff time performing updates.

Training costs depend on the complexity of the update process you devise and the sophistication of your user community. If your update process is embodied in a self-explanatory Web application, your training costs will be minimal. If your update process is embodied in a more complicated standalone application, more extensive training may be required.

Support costs also can vary depending on the update process you devise. In addition to Help Desk calls received from users asking how to use the update applications, you can expect to receive calls from people wanting to update fields that you don't want to allow them to update. Be prepared to explain your data maintenance policy; if possible, publish the policy as widely as possible.

One good way to lower support costs is to be proactive about giving users as much information as possible. For example, if a user's manager is responsible for updating certain fields for the Human Resources department before the changes are reflected in the directory, explain this in your online documentation. A user reading this documentation may then go directly to her manager rather than bothering the Help Desk, which would not have the relevant information to help her.

System Effects

Just as with centrally maintained data, you need to consider the effects of your user-maintained data policies on the directory service. The effects concern your directory's performance, its replication system, the quality of data, and other factors.

Directory performance can be affected by user updates just as it can by other updates. This type of traffic tends to slow down the directory, and many updates coming in unchecked could even result in a denial of service to other users of the directory. This might happen because of a malicious user or simply because of an error in the way someone is using the directory. For example, consider a user who tries to write a little program to update his directory entry several times a day. If this program becomes caught in some kind of loop, the load on the directory can be substantial. Use your audit trail and normal directory performance monitoring methods to guard against problems like this.

Replication performance is also affected by the number of updates your directory processes. Too many updates can cause replication to become bottlenecked. The result may be long delays during which different copies of your directory contain different information. Although loose consistency like this is a fundamental characteristic of most directories, taken to extremes it can lead to user confusion and more calls to the Help Desk.

User-maintained data does not always cause more updates to the directory. It may well be that the same or even fewer updates result. The question is one of control. With user-maintained data, you have no control over how many updates are performed, when they are performed, and by whom. You should guard against an out-of-control user update problem, even if it is unlikely to occur. Also guard against users not updating their information in a timely manner, which leads to stale data in the directory.

Data Validation

Quality of data is another concern when you're allowing user updates. We stated that improving data quality is one of the main reasons for introducing user updates in the first place, so this may come as a bit of a surprise. Keep in mind, however, that although users may be able to provide up-to-date and accurate information, they are, after all, only human. Unintentional mistakes are often made, and you should do what you can to guard against such mistakes. Also keep in mind that user motivation to update data may be highly variable.

One good method of guarding against data quality problems is to screen the data entered by users. With data screening, a process inspects the data to be added or changed in the directory. If the data is found to be faulty, the update is not made. You can often catch syntactic errors this way, although rarely semantic ones. Examples of syntactic errors include entering an e-mail address that contains spaces, a telephone number with nonprintable characters, a JPEG photograph that does not conform to the JPEG standard, and so on. Examples of semantic errors include entering a valid but incorrect e-mail address, somebody else's telephone number, and more. Some semantic checks can be performed. For example, you can perform minimal verification of an e-mail address by looking up mail exchange (MX) or address (A) records in the Domain Name System (DNS). If a user enters the telephone number of an existing user, a simple directory search can be used to detect this duplication and reject the change.

You can screen users' data in two places. First, you can modify the clients used to update users' entries to do the screening, if you have access to the source code and have the in-house expertise. Some off-the-shelf clients already provide support for this kind of data validation, although most do not. This method has the disadvantage that no checking is done if a user finds a different way to update her entry, perhaps by using a different client.

Another option is to enable data validation in the directory servers themselves. This approach eliminates the possibility of a user bypassing the validation because all updates come to the directory servers. As you learned in [Chapter 2](#), Introduction to LDAP, attributes have a syntax associated with them that determines the kind of information their values can contain. Many directory servers provide validation at this basic syntax level. This is enough to keep nonprintable characters out of phone numbers and e-mail addresses, but it's probably not enough to check the syntax of an e-mail address, and it is certainly not enough for checking something like a special employee number.

To perform more elaborate data validation checks such as those shown in the last few examples, you need the ability to change the directory's behavior. There are different ways of doing this, although few directory software products provide this capability. Netscape Directory Server 6 provides a set of plug-in interfaces that allow you to write a bit of code to perform a validation. (The Value Constraint plugin example from [Chapter 4](#), Overview of Netscape Directory Server, is an example of code that uses this set of interfaces.) This code then gets plugged into the directory server and called before an update operation is executed. Your plug-in has the option of refusing the operation (if the data is really messed up) or changing the data to an acceptable format (to remove dashes from an employee number, for example). Make sure that your directory software supports the kind of validation you need.

Handling New Data Sources

Unless your directory is very special-purpose, eventually you will need to add more data to it. Adding data involves integrating a new data source—perhaps another database within your organization, a new application that needs to store things in the directory, or your users themselves. In any case, you'll need to plan to handle the new data, as outlined in the following steps:

Step 1. Design a schema to hold the new data.

Step 2. Design a security policy for the new data.

Step 3. Integrate the new data source into your data maintenance policy.

Step 4. Design a new data update procedure for the new source (or adopt an existing one).

Step 5. Assess the impact of integrating the new data source on your directory service as a whole.

Step 6. Implement the new data update procedure.

Step 7. Make sure that the update procedure works correctly.

Whatever your new data source is, you'll need to design a schema to hold the information in the directory. Make sure that the policy governing maintenance of the new data is integrated into and compatible with your existing data maintenance policies. Follow the policies outlined in the section titled The Data Maintenance Policy earlier in this chapter, and the policies suggested in [Chapter 7](#), Data Design, to design the procedure by which the new information is updated. Finally, you'll need to implement the new procedure that embodies your data maintenance policy.

Following these steps should allow you to integrate new data sources with minimum disruption to the service. Make sure that you have an explicit plan for handling new data sources that covers all these steps in detail. Our experience has shown that integrating new data sources is a common task. If your directory service is popular and successful, many new applications will want to use it. Each application may well have specialized data that it needs to store or have available in the directory. The future success and growth of your service depend in large part on how well you respond to these needs. As with all substantial changes to your directory service, be sure to pilot the change before deploying a new data source in production.

[Chapter 23](#), Directory Coexistence, talks about this topic in great detail. [Chapters 21](#), Developing New Applications, and [22](#), Directory-Enabling Existing Applications, talk in great detail about creating applications that use the directory.

Handling Exceptions

As we've said repeatedly throughout this chapter, don't count on your data maintenance policies to cover 100 percent of the situations you encounter. Exceptions are a fact of life. You would do well to formulate policies and procedures that minimize the need for exception handling, but you should also prepare to handle the inevitable exceptions.

Exceptions take many forms. One form of exception involves simple errors. For example, you might count on data from one of your sources to be formatted in a particular way. If it is not, your data update procedure should handle this situation gracefully. Logging errors is always a good practice, as is correcting errors when possible. Aborting the entire update process because of a single error in the data is usually a bad idea, although there are exceptions even to this advice, of course. If the error is in mission-critical data that cannot tolerate any errors, aborting the update process may indeed be the best approach. Evaluate your situation to determine which course of action is most appropriate.

Another form of exception involves the policy itself. These kinds of exceptions are usually caused by user needs. For example, your policy might state that telephone and office address information for all employees is to be published to the outside world. Some of your employees might not want this information published for reasons of privacy and security (perhaps they've been the victims of stalking, for example). You could handle this exception by placing access control on the user's entry or by simply not updating the user's phone and address information from the data source. Be prepared to respond to legitimate exceptions like this.

Checking Data Quality

The purpose of data maintenance is to ensure that the data in your directory service has the highest possible quality. Quality of data has several aspects, but we will focus primarily on the accuracy and timeliness of data. Naturally you will want to check the quality of your data both to monitor how well your data maintenance procedures are working and to get an idea of the kind of service you're providing to the users of your directory.

Bad data can creep into your directory service from various directions, including the following:

- **Bad source data.** If there is bad data in the source from which you're populating your directory, the data in your directory will also be bad. If you detect such a situation, use the opportunity to improve the quality of the source data. If that approach doesn't work, you might consider filtering the data as it comes in to remove things such as nonprintable characters. If the data in the source is just plain wrong, try to find out why and correct the problem.
- **User or administrator error.** People make mistakes. Whenever users or administrators are responsible for entering data, you run the risk of human error. Increased education and training, as well as directory data validation filters, can help correct this kind of problem.
- **Systematic error.** Systematic error can be introduced by a flaw in the automated procedure used to populate the directory or by a program that has a bug in it. Fixing these kinds of problems can dramatically increase your directory's quality.

Methods of Checking Quality

There are several methods for checking the quality of data in your directory. The following are three common methods:

1. **Source of truth.** If you have a source of truth for the data you want to check (typically one or more of your source databases), you can simply compare its data with the data in your directory. This may be easier said than done, of course. You might dump the directory data and source data to files and write a script or program to compare the two files and then report any differences. Or you might write a program that reads information directly from the directory and the source database and then does the comparison online. This is likely to be expensive, however you do it. A lower-cost approach may be to incorporate this check into the regular data synchronization procedure you have developed for the source.
2. **Spot checks.** A second method is to perform spot checks of the directory and rely on statistical inferences to tell you about the overall quality of your directory data. You can write a program to select entries from the directory at random and compare them to the corresponding entries in the source-of-truth database. This method is much less expensive than doing a complete comparison. You'll need to decide for yourself how many entries to check to have confidence that you're getting an accurate and representative sample of data.
3. **User survey.** A third method is to survey users to ask them about data quality or monitor user complaints about incorrect data. This method works well only for data that users care about and can judge the accuracy of. This is also a statistical method, so you'll need to do some educated guessing to derive the overall quality of your

data.

Checking a source of truth and spot-checking are techniques that can be used to check the syntactic validity of information even when no source-of-truth database exists or is accessible. For example, you could read all (or a sampling) of the e-mail address attributes in the directory and determine whether they are syntactically valid.

Implications of Checking Quality

It's important to consider the implications of the methods you use to check the quality of your data for the operation of your directory service. Be sure to choose a method that does not significantly reduce directory performance. Depending on the method you select, you may have to make a trade-off between how often you check for quality and the accuracy of your checking methods. The main concerns in this area are methods that cause an excessive load on the directory or cause the directory to be unavailable.

For example, consider a method that requires reading over LDAP all the entries in your directory. Your directory might have the capacity to respond to this kind of request without degrading performance for other users, but then again it might not. If you use a method like this, you can run the check at night or at another off-peak time when the directory has plenty of extra capacity to respond to the data-checking requests. However, such an arrangement may be difficult if your directory operates in a global environment in which there is no off-peak time. Another approach then is to create a dedicated directory replica that does nothing but process these data verification tasks.

Consider also a method that requires you to dump your directory's data to a file. Some directory server software allows you to perform this operation without taking the service down, but some does not. If you are planning to use this method, make sure that the software you choose supports online dumps or that your service can tolerate the downtime. Remember that you have replication to help with the availability problem, so consider taking down a replica to produce the extract instead of taking down the master server. Also consider producing your own extract over LDAP, but be careful you don't degrade performance as discussed earlier.

Correcting Bad Data

Whatever method you use to check the quality of your data, be sure to investigate the cause whenever you encounter an error. Identifying the cause will help you correct problems with the system that produced the bad data. Although this kind of investigation can be time-consuming and expensive, it's usually well worth it. You'll often find that many errors are caused by the same underlying problem. Fixing that problem can dramatically increase the quality of your data.

Bad data may be caused by many underlying problems, some of which were already discussed briefly. Systematic errors in programs or procedures should be treated as bugs and corrected. Bad data introduced through human error might be the result of inadequate training or documentation for either users or administrators; increasing the quality and coverage of this training and documentation can cause corresponding improvements in the quality of your data. Human error can also be the result of poor software design. Spend time with users and administrators responsible for updating the directory, and observe the steps they take when maintaining data. Observing others will help you spot flaws in the software and procedures they use.

Finally, even if you can't eliminate poor data coming into your directory, you can mitigate the damage by installing data validation filters. As mentioned earlier, these filters can be installed in directory clients that users and administrators use to update the directory, or

they can be installed in the directory service itself.

Team LiB

◀ PREVIOUS

NEXT ▶

Maintaining Data Checklist

To review, perform the following steps when planning how to maintain the data in your directory:

- Identify who is allowed to update which attributes in which parts of your directory tree.
- Determine which attributes and which parts of your directory tree are centrally maintained and which are user-maintained.
- Determine the frequency of data updates from each source.
- Create a data maintenance policy.
- Design your data update procedures.
- Create a policy and procedures for adding new data sources.
- Analyze the effects of these procedures on the performance and availability of your directory service.
- Develop client software needed for performing updates.
- Develop training and documentation for users and administrators who will update the directory.
- Conduct training, and distribute documentation and software.
- Monitor data quality.
- Take action when you find quality problems. This is a repetitive process that often requires you to revisit earlier steps.

Further Reading

LDAP: Programming Directory-Enabled Applications with Lightweight Directory Access Protocol. T. Howes and M. Smith, Macmillan Technical Publishing, 1997.

Looking Ahead

Now that you know how to ensure the accuracy of data in your directory, it's time to learn how to monitor other aspects of the service's operation in [Chapter 19](#), Monitoring. After that, we will close the directory maintenance part of the book with [Chapter 20](#), Troubleshooting.

Chapter 19. Monitoring

- Introduction to Monitoring
- Selecting and Developing Monitoring Tools
- Notification Techniques
- Taking Action
- A Sample Directory Monitoring Utility
- Performance Analysis
- Monitoring Checklist
- Further Reading
- Looking Ahead

To deploy a reliable directory service that meets your users' expectations, you need to develop and implement a monitoring strategy. Your monitoring system should help you attain three main goals:

1. To quickly detect and be able to remedy failures of your directory service
2. To anticipate and resolve potential problems before they result in failure
3. To understand the usage patterns of your directory to aid you in capacity planning

In this chapter we first discuss general monitoring principles and the various types of monitoring strategies you might use. Then we cover the various types of monitoring systems available, including off-the-shelf network monitoring systems and custom monitoring tools. Next we cover the topic of notification, in which you inform the appropriate people about any problems detected. Then we discuss how to proceed when a failure or degradation of service is detected. We provide a simple monitoring tool, written in Perl, that you can use as a starting point for developing your own custom monitoring system, and we conclude with a discussion of performance monitoring and analysis.

Introduction to Monitoring

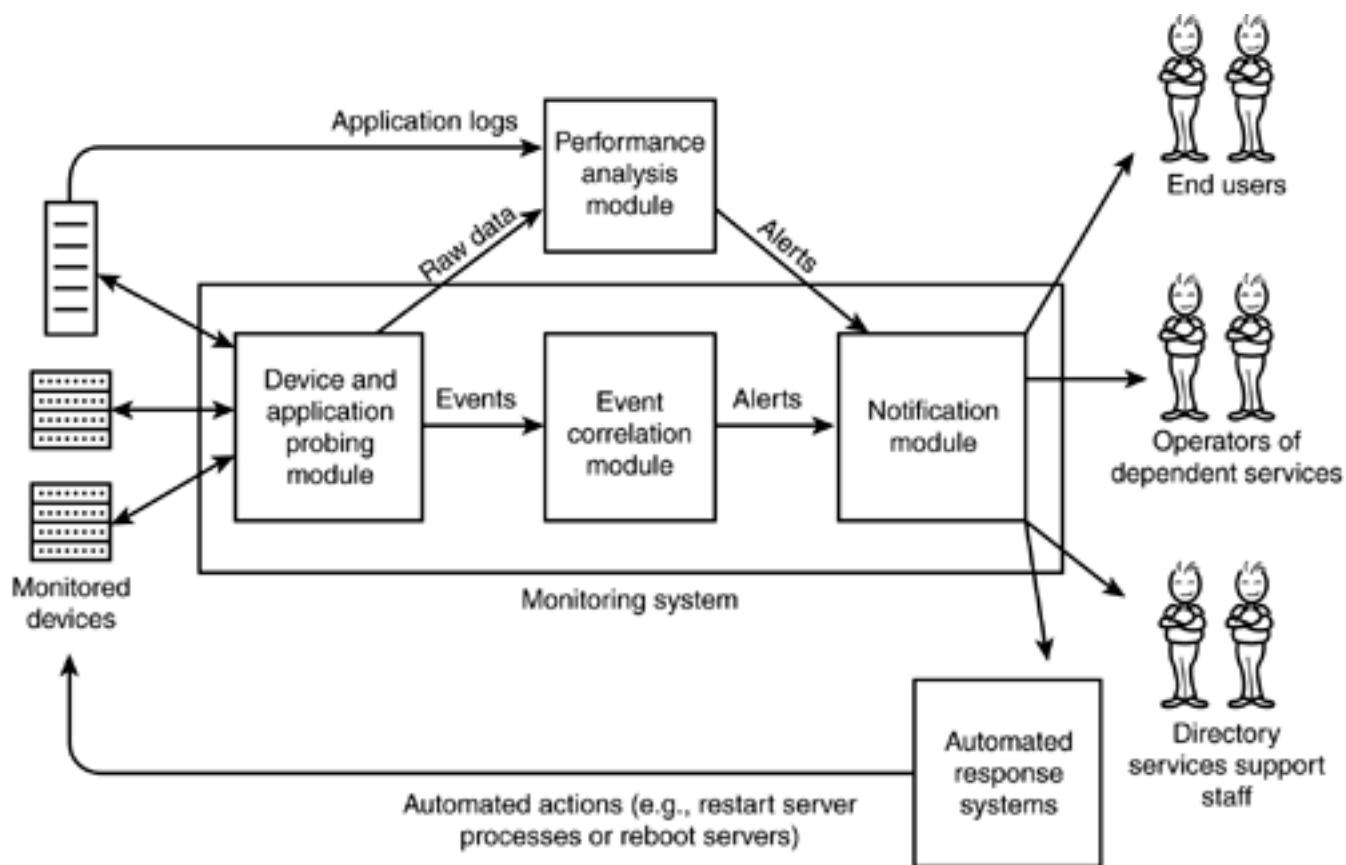
It's likely that your directory service is (or will be) a vital part of your computing infrastructure. Users may depend on it for things such as login, personalization, and address books. Applications may depend on it for such things as access control and e-mail delivery. Failure or unavailability of the directory can result in downtime for users and applications, translating into lost time and money. By monitoring your directory service, you can learn of outages as soon as they occur.

With more sophisticated, proactive monitoring strategies, you can also anticipate problems before they result in an outage or degraded service. Information you gather from this type of monitoring can be used to fine-tune your directory server software. Proactive monitoring alerts you to the need to change directory configuration parameters to optimize performance for common queries. It can also provide data that can help you optimize your management procedures. For example, an increase in the number of updates handled by your directory may signal the need for more frequent backups.

A monitoring system consists of four conceptual modules (see [Figure 19.1](#)):

1. **The device and application probing module.** This module is responsible for periodically checking the status of the monitored devices, hosts, and applications. When a device fails a test, an event is generated that describes the device and the test that was performed. This module may also log the response time for each probe, for use in performance analysis.
2. **The event correlation module.** This module is fed events, analyzes them to determine the root cause, and suppresses any events that might have occurred because of other events. For example, a network router might fail, temporarily making all devices, hosts, and applications beyond it inaccessible. Any alerts for those events would be suppressed because they are probably false alarms. After suppressing any inappropriate events, the module constructs one or more alerts and sends them to the notification module.
3. **The notification module.** This module receives alerts and notifies the appropriate people who can remedy the problem. Alternatively, this module might arrange for an automated system to take remedial action, such as restarting a server process or rebooting a failed server.
4. **The performance analysis module.** This module receives the raw data from the device and application probing module. The raw data generally includes probe response time and usage data collected from devices. The performance analysis module may also read and interpret application logs collected from servers.

Figure 19.1. A Conceptual Overview of Monitoring



The monitoring system shown in [Figure 19.1](#) is a conceptual model that we use to frame our discussion of directory monitoring. Any of the modules' functions could be performed by humans or a software program; however, if you use a commercially available network management system (NMS), you'll probably find that it implements some or all of these functions for you.

The most basic type of monitoring detects when the directory (or a part of it) is unavailable, perhaps because a server machine has crashed or has become unreachable because of a network failure. These directory failures are *hard failures*—that is, a part of the directory has failed completely.

Other types of directory problems can result in degraded performance. For example, looping electronic mail can cause the load on the directory to increase dramatically as the messaging servers attempt to deliver the looping mail. A more advanced monitoring tool could conceivably detect the increased load on the directory and alert a system administrator, who could take corrective action. A complete monitoring system should be able to detect hard failures and also detect when performance drops below an acceptable level.

In addition to detecting hard failures and unacceptable performance degradation, a well-designed monitoring solution provides you with valuable information on performance trends. Such proactive monitoring can help you anticipate problems before they become serious enough for your users to notice.

Methods of Monitoring

There are several ways to monitor a directory service. Following are the various types of monitoring you should consider:

- **Monitoring with Simple Network Management Protocol (SNMP).** SNMP is a network protocol that allows a management application to monitor the state of

managed devices on your network. Although SNMP has found its widest application in the management of networking hardware such as switches, hubs, and routers, it is also possible to use SNMP to monitor and manage application processes running on server computers. SNMP allows a management application to monitor the status of an entity on the network. It's also possible for a management application to be asynchronously notified via the SNMP trap mechanism when some sort of problem occurs (if a server process terminates unexpectedly, for example). We'll discuss SNMP in more detail later in this chapter.

- **Probing the directory via Lightweight Directory Access Protocol (LDAP).** One of the most straightforward and useful ways to monitor your directory service is to probe it by connecting to it as a client and issuing LDAP requests. For example, a simple probing tool might connect to a directory server and issue a search request for a given entry. If the entry is returned within a reasonable span of time, the directory is considered functional. If not, the probing tool can report a failure.
- **Monitoring operating system performance data.** Most modern operating systems (OSs) provide tools to query their operating parameters. This type of information can help you identify when your directory server performance is suffering because of an OS problem.
- **Indirect monitoring.** Monitoring the applications that use the directory provides more of an end-user view of the reliability and responsiveness of your system.
- **Analyzing log files.** You can automatically scan the directory service's log files for messages that indicate an error condition, and you can watch for conditions that signal a performance problem. Log file analysis is also a good way to perform proactive monitoring, in which you identify undesirable performance trends and telltale signs of impending problems before they are noticed by your users. Finally, log file analysis is the best way to understand how your directory's usage patterns are changing over time. This information is invaluable during capacity planning.

Later in this chapter, we discuss each of these five approaches in detail and provide specific examples.

General Monitoring Principles

Before we discuss specific monitoring methods, let's take a moment to introduce some general principles that apply to all methods.

Monitor Unobtrusively

You should always understand the implications of your monitoring strategy. A poorly designed monitoring system may adversely affect performance if it places a heavy load on the directory service. In general, you should strive to make the monitoring as unobtrusive as possible while still providing the information you need.

How do you make your monitoring unobtrusive? Use the most lightweight method available that gives you the needed information. For example, if you probe the directory, retrieving a single entry is probably sufficient; it's unnecessary to retrieve many entries. You should also perform the probe no more often than necessary to implement your desired responsiveness. For example, you can discover problems sooner if you probe the directory every five seconds, but that may be overkill; it's probably reasonable to probe every minute, or even every five minutes, depending on the level of service you are expected to provide to your users.

One Failure Can Cause Other Failures

Another potential problem if a failure occurs is that it may trigger other alerts in your monitoring system. For example, if one of a set of replicated servers becomes unavailable, the load on the remaining servers may increase as clients reapportion themselves among the remaining replicas.

Keep a Problem History

You should strive to design your monitoring system so that it provides a reliable history of problems and summarized usage data. For example, if you use a commercial NMS that logs alerts in a standard format, you might periodically extract the directory-related alerts and archive them in a central location. These extracted logs can help you identify trends that you can use to plan for expansion—and demonstrate your ever improving reliability figures to management (or hide the figures if they don't show improvement!). You can also summarize each day's directory logs and save the summary in an archive. Over time, the archived data becomes increasingly useful for capacity planning.

Have a Plan

Finally, for every type of failure you can anticipate, you should create a written action plan to share with all operators and support personnel who might be the first to learn of a failure. It's also a good idea to have a default action plan to be followed in the event of an unanticipated error. Action plans are covered in more detail in the Taking Action section of this chapter.

Selecting and Developing Monitoring Tools

As you set out to design your directory monitoring system, you have two main alternatives. You can choose an NMS package, such as IBM Tivoli NetView, Computer Associates' Unicenter, Hewlett-Packard's HP OpenView, or Aprisma's SPECTRUM; or you can develop your own set of tools to monitor the directory. Which approach should you use?

NMS packages have historically been used to monitor SNMP-enabled network devices such as routers, hubs, and switches. These packages typically have excellent data archiving and reporting capabilities; allow the definition of customized alerts; and offer event correlation capabilities, which permit the NMS to suppress spurious alerts. In addition, many NMSs can directly perform notification via e-mail, telephone, and pager. If your directory server software supports monitoring via SNMP, monitoring it with an NMS is a natural choice. This is especially true if you are already using an NMS to monitor the rest of your network. Even if your directory server does not support SNMP monitoring, you can install an SNMP agent such as Concord's eHealth SystemEDGE on each directory server. SystemEDGE can perform various different checks on the health of the operating system and directory server processes. We'll discuss this approach shortly.

If you do not already use an NMS in your organization and you cannot justify the cost of purchasing and deploying one, it makes sense to develop a set of tools that perform the directory monitoring function. A simple set of Perl scripts, for example, can be used to perform extensive monitoring of your directory service. Later in this chapter we offer some general design hints for developing a set of tools and show you how you can develop notification methods.

In the following sections we discuss the NMS and custom monitoring approaches in detail.

Monitoring Your Directory with SNMP and an NMS

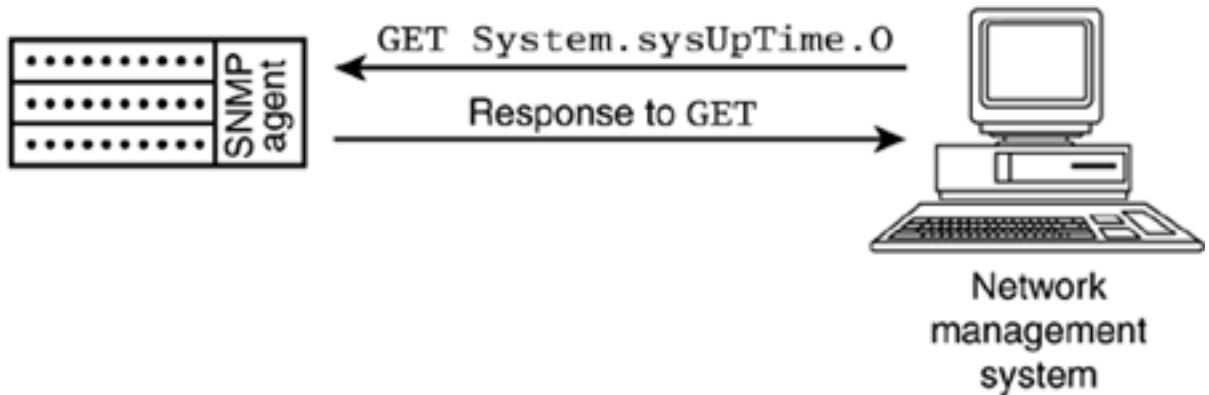
If your directory server supports SNMP, or if you use SNMP agent software on your servers, you can monitor it with one of many commercially available NMS software packages. Before we discuss this topic, however, let's examine how SNMP works.

Introduction to SNMP

Simple Network Management Protocol is an Internet standard protocol for exchanging management information. In a typical SNMP installation, a managed device runs an SNMP agent, and the management station runs an SNMP manager application. The manager may request information from the agent with an SNMP **GET** request, and the agent responds with a **GET** response containing the requested data.

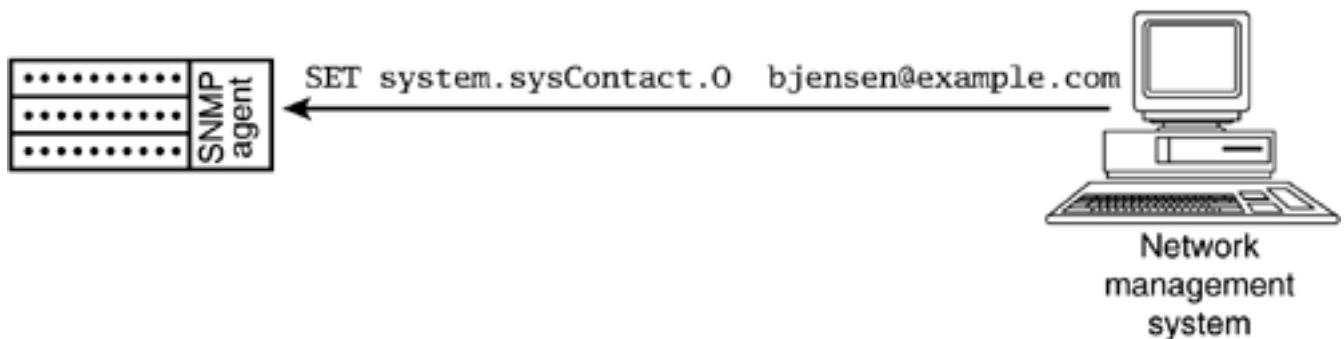
[Figure 19.2](#) shows a manager requesting a piece of information from a managed device. The NMS has issued a request to read a piece of management information from the managed device, an Ethernet hub. The device returns the requested information (`systemUpTime`, or elapsed time since the device was powered on) to the NMS.

Figure 19.2. An SNMP Manager Issues a **GET Request to a Managed Device**



SNMP also allows a manager to send a **SET** request to an agent. This request instructs the agent to modify its operational status in some way. [Figure 19.3](#) shows a manager issuing a **SET** request to a managed device. This **SET** request sets the `system.sysContact.0` Management Information Base (MIB) variable to the string `bjensen@example.com`, which represents the e-mail address of the person responsible for the managed device (we'll discuss the MIB shortly).

Figure 19.3. An SNMP Manager Issues a **SET Request to a Managed Device**

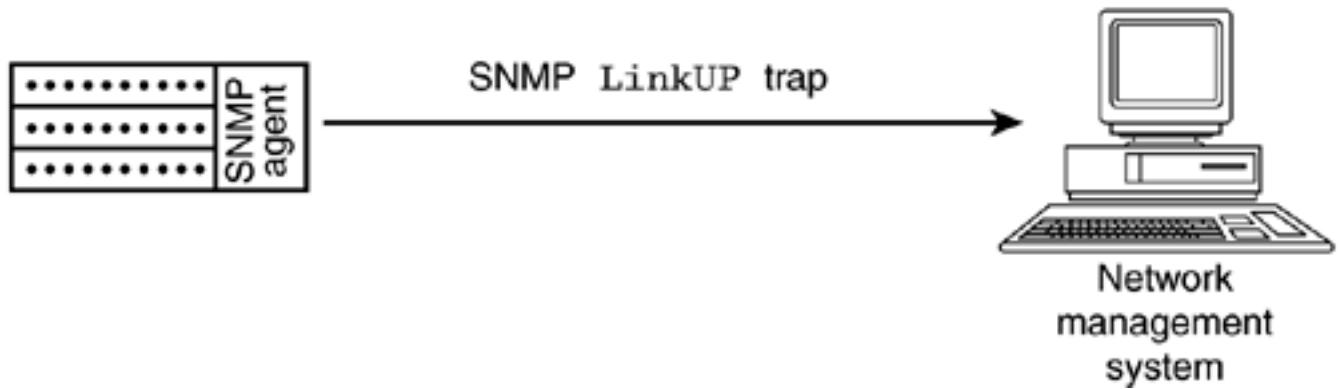


Caution

Older SNMP standards do not provide secure authentication, so SNMP **SET** commands are rarely enabled in managed devices.

Managed devices can also generate spontaneous messages called *traps*, which indicate an exceptional condition. In [Figure 19.4](#), the managed device has encountered an error and has generated a trap that it has sent to the management system. As before, the NMS may choose to take some action upon receiving a trap from a managed device.

Figure 19.4. A Network Device Generates an SNMP Trap



SNMP can be used to manage a wide range of devices on a network, including switches, routers, and hubs. But how does an SNMP management application know what types of data are made available by a particular agent? The collection of available management information for all devices in the universe is described in the MIB, a huge tree of management information. Each type of device has its own branch of the tree, and the leaves of the tree represent the actual parameters that may be managed. The structure of the MIB isn't particularly important to our discussion, however. Suffice it to say that the MIB assigns a unique identifier to every possible parameter on every type of device you might want to monitor. These identifiers let managers and agents refer precisely and unambiguously to operating parameters of every device on the network.

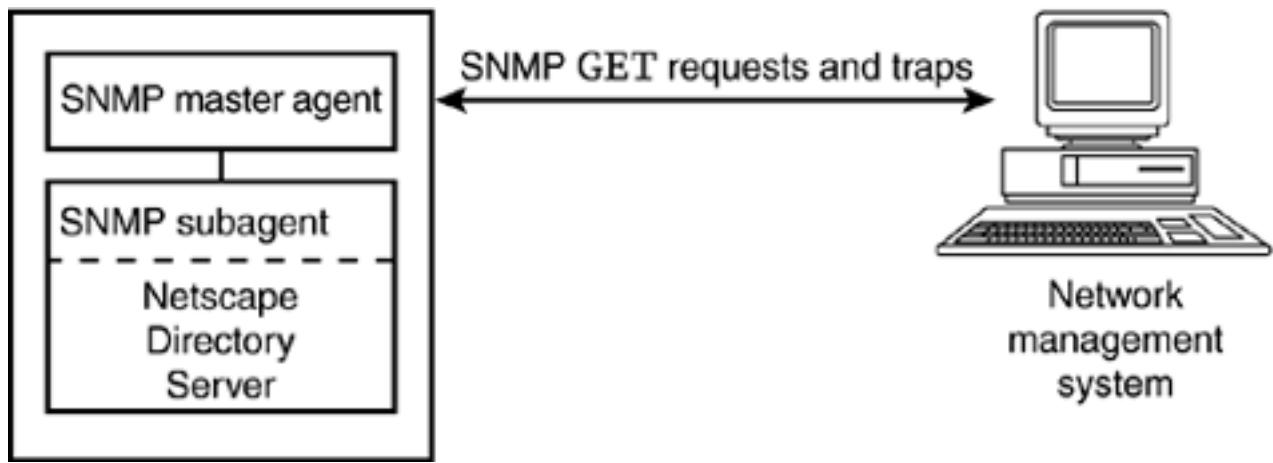
When a vendor creates a new piece of networking hardware and wants to make it manageable via SNMP, it needs to create agent software for the device and document the MIB variables that the device supports. If the device performs a common function for which variables already exist in the MIB, no additional MIB variable needs to be defined. If, on the other hand, the device provides new functionality that is to be monitored via SNMP, the vendor needs to assign and document new MIB variables that correspond to these new functions.

In a typical enterprise, one or more NMS stations monitor large numbers of network devices. These management stations poll the devices and record the collected data in a database. Other parts of the NMS display the data graphically. For example, an NMS might offer a pictorial view of a vendor's router. If one of the interfaces on the router encounters many errors in a short time, the NMS package might color the view red to indicate a problem and alert an on-call person by e-mail or pager. NMS software makes the task of managing hundreds or thousands of network devices much easier by automating the data collection and analysis processes.

Directory Servers and the Directory Server Monitoring MIB

Although SNMP is most often used to monitor networking hardware such as hubs, switches, and routers, it can also be used to monitor applications software such as an LDAP server. [Figure 19.5](#) shows how Netscape Directory Server 6 can be monitored with SNMP.

Figure 19.5. Monitoring Netscape Directory Server via SNMP



The host that runs Netscape Directory Server must run an SNMP master agent (on Unix platforms the master agent is included with Netscape Administration Server; on Windows the master agent is included with the operating system). This master agent routes all incoming communication from the NMS to the appropriate subagent (there might be more than one subagent if more than one monitored server is running on the host). The subagent processes the SNMP request and returns the result to the master agent, which routes it to the NMS. Similarly, a subagent may send a trap to the master agent, which forwards it to the NMS.

Netscape Directory Server supports a directory server monitoring MIB that is similar to a subset of the directory server monitoring MIB defined in RFC 2605.

The MIB variables supported by the Netscape server are divided into two sections, or tables: the operations table and the entries table. The operations table provides information about the operations processed by the server, and the entries table provides information about the entries stored by the server. All the counters described in the table are reset whenever the directory server is restarted. The MIB variables supported by Netscape Directory Server are listed in [Table 19.1](#).

Table 19.1. SNMP MIB Variables Supported by Netscape Directory Server

MIB Variable	Description
Operations (all totals are since server startup)	
dsAnonymousBinds	The number of anonymous bind operations processed
dsUnauthBinds	The number of unauthenticated (anonymous) bind operations processed
dsSimpleAuthBinds	The number of bind operations that used simple authentication

<code>dsStrongAuthBinds</code>	The number of bind operations that used strong authentication (SSL or a strong SASL mechanism such as HTTP digest) to authenticate the client's identity
<code>dsBindSecurityErrors</code>	The number of bind requests that failed because of invalid credentials
<code>dsInOps</code>	The number of operations forwarded to this directory server from another
<code>dsCompareOps</code>	The number of compare operations processed
<code>dsAddEntryOps</code>	The number of add operations processed
<code>dsRemoveEntryOps</code>	The number of delete operations processed
<code>dsModifyEntryOps</code>	The number of modify operations processed
<code>dsModifyRDNops</code>	The number of modify RDN operations processed
<code>dsSearchOps</code>	The number of search operations (of any scope) processed
<code>dsOneLevelSearchOps</code>	The number of <code>scope=onelevel</code> search operations processed
<code>dsWholeSubtreeSearchOps</code>	The number of <code>scope=subtree</code> search operations processed
<code>dsReferrals</code>	The number of referrals returned to clients
<code>dsSecurityErrors</code>	The number of operations forwarded to this directory server but rejected because of security problems
<code>dsErrors</code>	The number of requests that could not be processed because of errors
Entries	
<code>dsaCacheEntries</code>	The number of entries contained in the server's entry cache

In addition to making these MIB variables available via SNMP **GET** requests, the Netscape Directory Server SNMP subagent generates an enterprise-specific trap whenever the server starts up and whenever it shuts down, either normally or abnormally. You can make use of this information by configuring your NMS to generate an alert when parameters exceed some preset limits. You may also want to configure your NMS to warn you if it receives a "server down" trap, indicating that the server has shut down for some reason.

More information about using SNMP with Netscape Directory Server can be found in the *Netscape Directory Server Administrator's Guide*.

Monitoring Your Directory Server Using Host-Based SNMP Agents

It's also possible to install a general-purpose SNMP agent on all of your directory server hosts. These types of agents can monitor the overall health of your servers and report back to your NMS over SNMP. For example, the agent can be configured to monitor disk space usage. It can make the current disk utilization data available to your NMS via SNMP polling, and it can be configured to generate an SNMP trap when the disk utilization exceeds a predefined threshold.

An example of this type of agent is Concord's eHealth SystemEDGE. SystemEDGE can do all of the following:

- Verify that your directory server process is still running, and generate an SNMP trap if it is not.
- Watch for specific error messages to appear in a log file or the Windows event log, and generate an SNMP trap if the message is logged.
- Monitor system usage parameters such as disk utilization, virtual memory usage, and CPU utilization, and generate an SNMP trap when thresholds are exceeded.
- Record historical data on system usage.

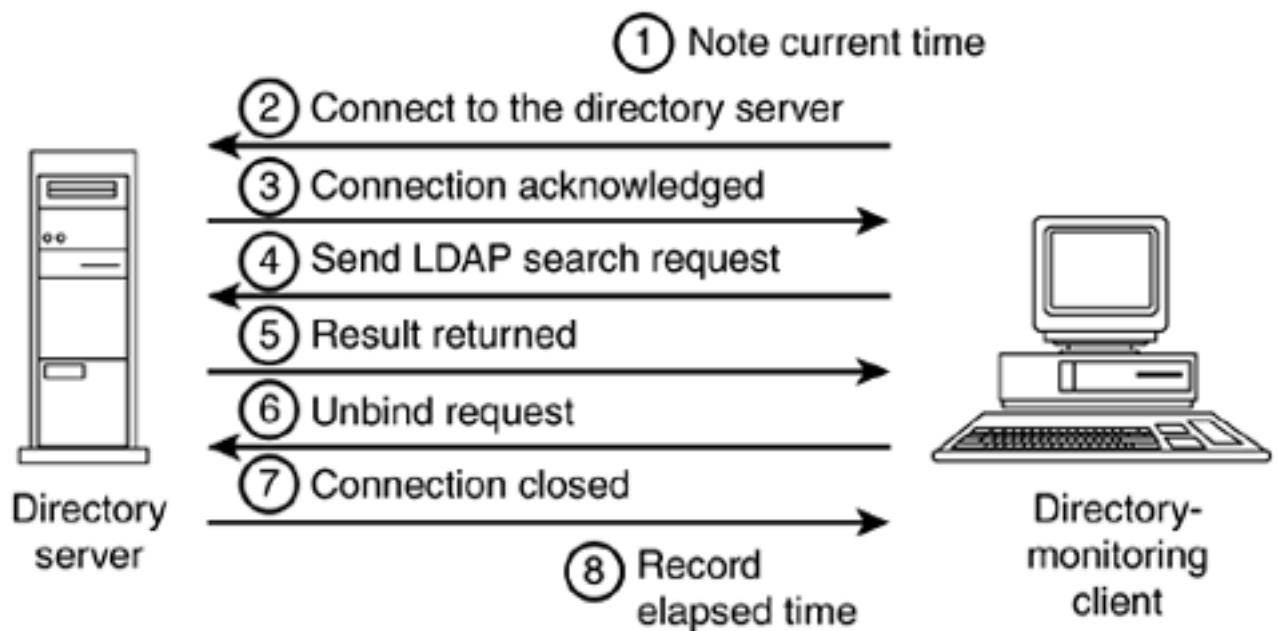
This type of monitoring is very useful, providing additional information that the Directory Services Monitoring MIB does not provide.

Monitoring Your Directory with Custom Probing Tools

If you don't have an NMS but still need to monitor your directory service, you can construct a simple set of tools using off-the-shelf components such as command-line LDAP tools or PerLDAP. In this section we suggest some general design principles for developing your own monitoring tools.

We suggest you probe your directory by performing the same general types of operations that your users and directory-enabled applications perform. For example, you might choose to probe your directory by periodically attempting to read an entry from the directory (see [Figure 19.6](#)).

Figure 19.6. Probing the Directory via LDAP



When you probe the directory in this manner, the expected result is that a single entry will be returned. If a different result is obtained, the LDAP server is unavailable, unreachable because of a network failure, or malfunctioning in some manner. [Table 19.2](#) summarizes the most common failure modes for a monitoring client.

How do you know when to declare a failure? Think about the error conditions you consider to be critical. If the directory server is completely unresponsive, this is obviously a serious condition that must be remedied. But what if the server responds slowly? Part of your development effort involves setting thresholds. For example, you might decide that a response time of longer than three seconds for a simple search constitutes a serious degradation of performance and calls for notification of the appropriate person.

If one of the first two error conditions in [Table 19.2](#) is encountered, the fault may lie with the network rather than the server. You can confirm this condition by sending an Internet Control Message Protocol (ICMP) echo request (a ping) to the router closest to the directory server. If the router isn't able to receive a ping, you really can't draw any conclusions about the state of the directory server. About the best you can do is to declare its state "unknown."

Table 19.2. Types of Monitoring Failures

Result	Explanation
No route to host.	The directory server host cannot be contacted. A network failure or DNS lookup failure is the most likely cause.
Connection times out.	The directory server host is down, or the network between the monitoring tool and the directory server is down.
Connection refused.	No directory server is running on the host.

An LDAP error code other than `LDAP_SUCCESS(0)` is returned.

The directory server encountered an error while servicing the search request.

Server responds, but too slowly.

The directory server is experiencing a problem or misconfiguration that is degrading its performance, or the server is simply overloaded.

How you handle this situation is up to you. You can consider it a failure and notify the appropriate people, or you can ignore the error condition on the assumption that the server is running but unreachable. Another option is to place the monitoring tool on the same network as the server(s) being monitored. This makes it much less likely that the monitoring host will lose contact with the monitored server. Of course, a network failure will cause you to lose contact with the monitoring host, but historical data collected will be more complete.

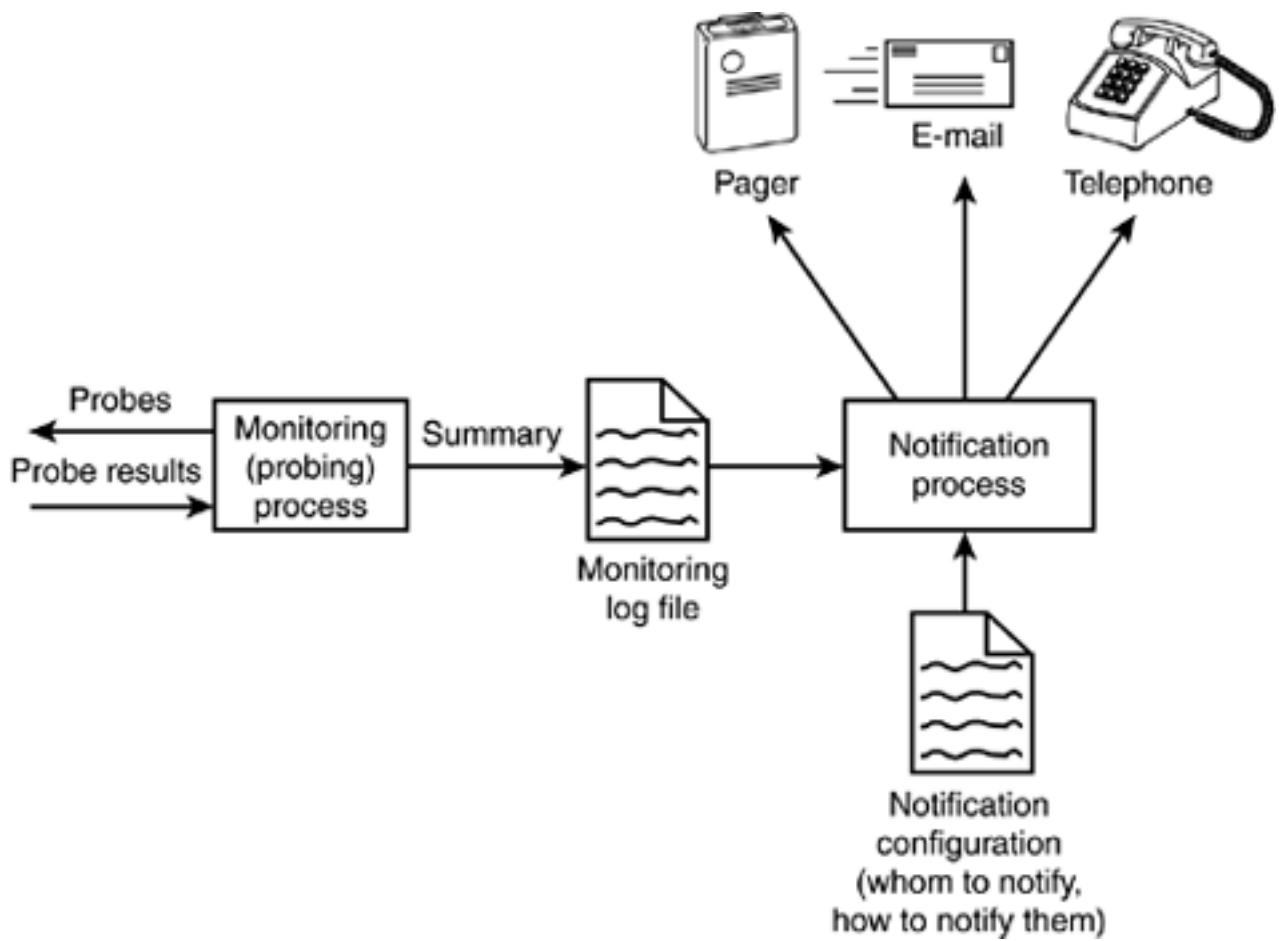
In addition to simple searches, you might consider test probes that approximate the types of activity your users and directory-enabled applications generate. For example, if a common activity is for your users to update their personal information (such as home telephone number), you might develop a script that replaces the `homeTelephoneNumber` attribute of a special test entry. If you're not sure what types of test probes would approximate user activity, examine your server logs.

Tip

The sample Perl script included at the end of this chapter implements a test like those described here: It searches for an entry and reports an error if the entry cannot be retrieved. The script is "smart"; that is, it avoids generating an alert if the directory server becomes unreachable because of a network failure.

We recommend that you use a modular approach when designing custom monitoring tools. In a modular approach, you separate the actual monitoring processes from the policy decisions about what constitutes a failure and who should be notified. One way to accomplish such a separation is to have the probing modules simply write their test results to a log file in a standard format, then have another process read and interpret the log file and implement a notification policy. For example, you might implement a policy stating that a directory server at a remote location must exhibit slow performance twice in any five-minute period before an alert is raised (doing this requires saving state information between probes). [Figure 19.7](#) depicts such an architecture.

Figure 19.7. Separation of Monitoring Functions and Alert/Notification Functions



Log File Analysis

Directory server log files can provide an effective way to monitor your servers. Most server software logs warning and error messages to a well-known location. On Unix systems, software often logs messages using the syslog facility. On Windows systems, messages are often logged to the Windows event log and may be viewed with the event viewer utility. Netscape Directory Server logs error messages to a file rather than using the system-provided logging facilities.

If you use the SystemEDGE agent described earlier in this chapter, you can configure it to watch for specific error messages to appear in the log and generate a trap if they do. Freely available tools like swatch can also be used to watch log files and take action in response to particular error messages.

Log file analysis is also the centerpiece of performance monitoring and analysis, which is discussed later in this chapter.

Operating System Performance Data

OS performance data can be a valuable aid in identifying the underlying causes of performance trends. For example, you can discover the total amount of virtual memory in use on the system or the amount of free disk space on a particular disk or disk partition on virtually any OS. These types of information can help you identify when your directory server performance is suffering because of an OS problem. For example, if your OS constantly writes to its virtual memory paging space, the physical memory may be insufficient for the applications running on the machine. You can remedy this problem by reducing the number of applications running on the machine (by moving them to another machine), or by adding additional RAM to the machine.

OS performance data can be monitored by an SNMP agent that obtains it and makes it available via SNMP, as mentioned earlier. This capability allows you to monitor the server's operating system health from your central NMS, if you have one. If you have no NMS, you can write custom scripts that periodically sample system performance parameters and log the sampled data to files for later analysis.

Monitoring Synchronization Processes and Data Quality

If your directory is synchronized from an authoritative data source such as a PeopleSoft human resources database, it's a good idea to monitor the quality of your data. For example, you can automatically spot-check a few entries randomly and verify that the data in the directory matches the data in the HR system. Discrepancies indicate that the synchronization process is not working correctly. You can also monitor the quality of data for which the directory itself is the authoritative data source. For example, you might check whether e-mail addresses in the directory are in the correct format. More information on monitoring data quality via data validation techniques can be found in [Chapter 18, Maintaining Data](#).

Indirect Monitoring

Another approach that should be part of your monitoring strategy is indirect monitoring. In this approach you don't monitor the individual services such as your directory. Instead you construct monitoring tools that measure availability and response times for the things that matter directly to your end users. For example, if you provide an electronic mail service that depends on the directory, you might periodically measure the time it takes to send an e-mail message from one test user to another.

Indirect monitoring is the real acid test for your servers and network because any single part of your complex, interconnected systems can be at fault. In combination with monitoring the individual devices, servers, and server processes that make up your directory-enabled application, indirect monitoring allows you to quickly identify and repair problems—and ultimately improve the reliability and performance of all your directory-enabled applications.

Notification Techniques

The whole point of monitoring your directory is to provide the best possible service for your users by anticipating problems before they cause a failure and detecting and repairing failures quickly. If your automated monitoring tools detect a problem, they need to notify someone so that appropriate action can be taken.

In this section we present some general principles you should follow when planning your notification strategy, and we suggest some notification methods you might use. The possibilities range from simple manual systems to sophisticated automated systems.

Basic Notification Principles

An effective notification system accomplishes four goals when a failure of the directory service is detected:

1. It notifies the people responsible for fixing the directory system.
2. It notifies the people who administer affected systems, such as electronic mail servers.
3. It notifies the people affected by the outage.
4. It notifies each person in an appropriate way.

As soon as a problem is detected, your system should notify the person who can fix it. Depending on your organization and its policies, this person might be a rotating "firefighter" who is on call and has responsibility for taking care of any serious emergencies, or it might be the person who deployed the directory service. This type of notification is typically urgent; the person should be telephoned or paged if the directory is a critical 24x7 service.

The notification system should also send a notification to whoever administers systems that might be affected by the directory failure. For example, if the directory server provides service to several electronic mail servers, the administrators of these servers may want to know about the directory server failure, especially if they receive complaints from users. This type of notification is advisory; it might take the form of an e-mail message to the administrator, or a special Web page on your intranet might list known outages. If you use e-mail as a notification method, be aware that e-mail delivery itself might be delayed by the directory outage.

Your users may also want to receive or obtain some sort of notification when the directory service is unavailable, along with an estimate of when the directory will be back in service. For this purpose, you could maintain a Web page listing all known outages so that end users can learn for themselves about system failures (instead of calling the Help Desk). You can also publish your system maintenance messages in ways that don't depend on the network itself, perhaps via a recorded telephone message. Making this type of information directly available to end users improves user satisfaction and lowers Help Desk costs.

The type of notification should be appropriate for the intended recipient. If you have a support person on 24-hour call (and she is receiving on-call pay), it's entirely appropriate to page her at 4 A.M. and tell her about the failure of a critical directory server. On the other hand, if your organization is experimenting with directory services and nobody is on call, be judicious with your use of intrusive notification methods such as pages and telephone calls.

The last thing you want to do when piloting a directory service is to alienate the people whom you need to make the project work! For end users and system administrators responsible for dependent systems, an on-demand notification system such as a Web page or recorded telephone message is usually appropriate.

An important point to remember is that you should avoid "crying wolf." In other words, your notification system should try to get someone's attention only when there is a real problem. If it generates false alarms, the credibility of future notification messages that your system generates will be suspect. For example, suppose a network router fails, disconnecting your directory server from the rest of the network, including your monitoring station. You shouldn't drag the directory administrator out of bed in the middle of the night; there's nothing she can do about the problem (unless, of course, she also manages the router). Whoever or whatever looks at multiple alerts from your monitoring system needs to have the ability to analyze those alerts and determine the root cause.

When a notification message does need to be generated, the message should be tailored to the intended audience. For a directory administrator, we suggest that the notification message describe the test that failed, in what manner it failed, and when it failed. For example, if the directory is monitored via SNMP and a "server down" trap is generated, the message should state exactly that fact, as well as the time the trap was received by the NMS. The following message would be appropriate for a directory administrator:

```
Directory server ldap2.example.com SNMP trap (server down) at 02:32:23
```

For end users, however, the message should be more descriptive of the effect the problem will have—for example,

```
Directory server ldap2.example.com is unavailable as of 2:32 AM
```

If the person or system generating the alert message has knowledge of services that depend on the failed server, an even more informative message, like the following, can be generated:

```
Directory server ldap2.example.com is unavailable as of 8:45 AM.
```

```
Electronic mail for users with accounts on mail2.example.com and  
mail4.example.com will be inaccessible. Service should be  
restored by 10:00 AM.
```

Ideally your notification should allow a person fixing the problem to annotate any status information you provide. For example, if your notification messages are generated by an operator in a 24x7 network operations center, a directory administrator repairing a failed directory server should be able to call the center, give the operator an estimated time for the repair, and expect the network operations center operator in turn to be able to update any status messages via the Web, automatic phone messages, or other means.

Notification Methods

There are various different methods you might use to notify people of problems with your directory service. If you already have a monitoring and notification infrastructure in your organization, it probably makes sense just to use that instead of developing something new. The following sections might provide some new ideas you can incorporate into an existing notification system or a custom system built from scratch.

One approach is to have an operator sit at a monitoring console and watch for alerts. If an alert is generated for a directory service, the operator can restart the directory service or call an appropriate person. With this type of manual approach, the operator needs to have clear procedures to follow, especially if he is not an expert on directory systems. The procedures should state when it is appropriate to restart the directory and when an expert should be called. Some sort of audit trail should also be generated, especially if the operator repairs the problem without expert intervention. For example, if the directory service fails at the same time each day and is simply rebooted by the operations staff, the deployment staff needs to learn about this so that it can analyze and remedy the problem.

Another approach is to use or build a software program that performs the notification function. Conceptually, the program receives an alert that names a device, server, or application, which is looked up in a table and mapped to a set of people and notification methods. Such a table might look like [Figure 19.8](#). This excerpt from a larger notification table indicates that when a failure is detected with server `ldap-hq.example.com`, one of two actions is to be taken, depending on the time of day. Between 8 A.M. and 5 P.M. (0800–1700), a page is sent to the telephone number 1-800-555-1234 using PIN (pager identification number) 9511, and e-mail is sent to `bjensen@example.com`. Between 5 P.M. and 8 A.M., a different pager is signaled, and e-mail is sent to `oncall-directory@example.com`. A similar procedure is in effect for the LDAP server `ldap-cleveland.example.com`. (Our sample Perl scripts at the end of the chapter include a simple, table-driven notification package.)

Figure 19.8. A Portion of a Notification Table

<code>ldap-hq.example.com</code>	0800-1700 page 1 800 555 1234 PIN 9511 email <code>bjensen@example.com</code>
	1700-0800 page 1 800 555 1234 PIN 2927 email <code>oncall-directory@example.com</code>
<code>ldap-cleveland.example.com</code>	0800-1700 page 1 216 555 1234 PIN 2193 email <code>hsmith@example.com</code>
	1700-0800 page 1 800 555 1234 PIN 2927 email <code>oncall-directory@example.com</code>

Testing Your Notification System

In the same way that the Emergency Broadcast System in the United States is periodically tested to ensure its functionality, you should test your notification system from time to time. There are several approaches you might use.

The notification system itself might periodically issue a special alert that causes a special test notification to be sent. Any notification generated, such as pages or e-mail messages, should be obviously marked as a test. For example, a test of the e-mail notification method

might be worded like this:

Testing automatic notification system. No action is required on your part.

In addition, it may be prudent to test the notification tables by injecting all possible alerts into the system and checking that the correct notifications are made. Of course, you need to schedule such a test well in advance and inform all the people who will receive notification during this process. You also need to take special precautions to watch for actual failures that occur during the test.

Team LiB

◀ PREVIOUS

NEXT ▶

Taking Action

As soon as a failure or degradation of your directory service is detected, a person or an automated process needs to do something about it. In this section we describe some general principles of problem evaluation and resolution, along with some things you should be aware of when remedying problems with a directory service.

Planning Your Course of Action

When you note a failure of the directory service, take a moment to plan your course of action. Although it is certainly important to get the directory service running correctly as soon as possible, you should also be sure to proceed in a methodical fashion. Before taking any action, ask yourself the following questions:

- Exactly what part of the directory service has failed? In what way has it failed?
- Which users and directory-enabled applications depend on the failed service?
- What exactly is the effect? Is something irreparable happening because of this outage (for example, is e-mail bouncing), or has the system simply stopped performing useful work? A guiding principle here is that delayed results or degraded performance are better than downtime or incorrect results.
- Are there any log files or other evidence that might be erased or overwritten? If so, copy these files as soon as possible so that they don't get lost.

As soon as you know who is affected by the failure, provide some feedback to let them know the scope of the problem and that you are beginning to work on a resolution. Include an estimated time for repair, if possible. Notifying your end users is important; although it may seem like something you don't have time for, end users deal with outages much better when they aren't kept in the dark.

Minimizing the Effect

After you've achieved a basic understanding of the scope of the problem and who and what it affects, try to minimize the effects of the outage. Doing this can buy you additional time to understand the problem. For example, if you maintain several replicas of the failed server, you may be able to reconfigure your dependent services so that they access the replicas. In some cases, failover to a replica is handled automatically by the directory software.

It may also be possible to shut down the dependent services temporarily to minimize the effects. For example, if you have a batch update process that merges changes from a foreign data source, and this process places a heavy load on the directory, you might consider waiting to run that process until the service has been repaired.

Directory-enabled electronic message transfer agents (MTAs) that handle mail from external sites are a good example of a dependent service that can be shut down to reduce directory load. Although no inbound mail would be received if the MTA were shut down, external mail servers would queue the mail and periodically attempt to deliver it. MTAs that handle outbound mail should be kept online, if possible, so that users can send mail.

Understanding the Root Cause

After you've isolated the failing components and informed the affected parties, you need to understand the root cause of the problem as thoroughly as possible. If you don't eventually figure out why the failure occurred, it's quite likely that you'll have the same problem in the

future.

Troubleshooting a complex, distributed system is tricky business, and we can't possibly hope to explain it all in this text. We do, however, outline a basic step-by-step strategy you might follow when analyzing directory service failures:

Step 1. Gather evidence, which can include any of the following:

- Directory usage and error logs
- Usage and error logs of dependent applications
- User reports of problems
- Operating system logs
- Alerts generated by your monitoring system
- Core dumps from directory server software (Unix systems) or Dr. Watson logs (Windows NT)

Step 2. After you've collected the evidence, look for things out of the ordinary that occurred around the time of the failure. You'll often know only the time that a user complaint was received or that the monitoring system noticed a problem with the directory.

Step 3. Starting at that time, work backward through the log files, looking for any suspicious error messages. Arrange these chronologically to get a better picture of the sequence of events leading up to the failure (but watch out for clock skew between computer systems).

These steps often lead you directly to the root cause. For example, if the directory service was unable to read database entries from a failing disk drive, you would probably find a close correlation between the time the directory failed and the time the operating system began to log failures.

If the root cause is still not obvious, you may want to involve your directory vendor's technical support staff. The log files and other evidence you collect should help the vendor understand and troubleshoot the problem. Another option you may have, depending on your software, is to enable more verbose log output and put the directory server back into service. This action would probably degrade service somewhat, but the additional log output might be very revealing, especially to your vendor's support staff. Be sure to reconfigure the server for normal logging after you've reproduced the problem with verbose logging enabled.

If the problem is intermittent and hard to reproduce, you may find that a test bed environment allows you to investigate the problem more thoroughly without affecting your production service. A test bed is also a great place to test new software versions and configuration changes before rolling them out into your production environment.

Correcting the Problem

The process of directory problem resolution is covered in detail in [Chapter 20](#), Troubleshooting. After you've corrected the problem, use the opportunity to improve your directory service. Ask yourself the following questions:

- Could anything have been done to avoid the failure in the first place? Is there a workaround that can minimize the effect if the problem happens again?
- Was the correct person notified promptly? Can the notification system be improved?
- Should a specific procedure be followed if this problem occurs again? If the problem recurs, and someone else needs to deal with it, any documentation you can provide will help him or her.

Documenting What Happened

Producing a detailed problem/resolution report is a great idea for several reasons. First, it serves as a learning tool for your fellow system administrators. Second, if the root cause is a bug in the directory server software, a detailed problem report can be extremely helpful to your vendor. Components of a good problem report include the following:

- Version numbers of the directory server software.
- Version numbers of the operating system software.
- Any patches or service packs installed for the operating system.
- Any optional or third-party software installed on the system, such as the Veritas file system option on the Solaris operating system.
- Relevant portions of directory server logs, operating system logs, and dependent software logs.
- Copies of directory database files (be aware of the contents of these files—they may contain sensitive information).
- Steps to re-create the problem, if known.
- An accurate description of the observed behaviors. Try not to make any assumptions here; instead, simply report what you saw. In other words, offer "just the facts."
- Your assessment of the root cause of the problem, if any.

After a problem has been resolved, it's a good idea to conduct a postmortem and ask some questions of yourself. Would troubleshooting have been easier if additional documentation were available? Did you have all the network maps for your organization? Was it clear which applications used the directory and which servers were used by those applications? Did you have all the necessary administrative passwords to gain access to the affected systems? Did you have the telephone and pager numbers of everyone you needed to contact? Always think about how to make your troubleshooting process more effective.

A Sample Directory Monitoring Utility

The following Perl scripts implement a very simple monitoring and notification system for LDAP servers. You can use the scripts as they appear here or as a starting point for a more elaborate monitoring system.

The first script, `ldap_probe.pl`, probes the directory server once per minute. It attempts to retrieve the entry whose distinguished name is given on the command line. For example, the following command:

```
./ldap_probe.pl dir.example.com 389 "cn=Test Entry, dc=example, dc=com"
```

attempts to read the entry `cn=Test Entry,dc=example,dc=com` from the LDAP server. If the probe succeeds, the script waits 60 seconds and repeats the probe. If the probe fails for any reason, the script invokes the `notify.pl` notification script. The notification script is passed a host identifier string and an indication of whether the server has gone down or has come back up. The `notify.pl` script then looks up the host/port combination in its configuration file (`notify.conf`) and performs the appropriate notification.

This division of work illustrates a concept we introduced earlier: keeping the probing and notification functions separate. To alter the notification actions performed, all that is necessary is to edit the `notify.conf` configuration file. Notification via e-mail is supported by the script, although it can easily be extended to support text paging.

The scripts assume that you are using a Unix workstation, although modifying them to run under Windows should be simple. They also assume that the Netscape LDAP command-line client `ldapsearch` utility is available in a directory contained in your search path.

[Listing 19.1](#) contains the `ldap_probe.pl` script, which performs the probing functions. [Listing 19.2](#) contains the `notify.pl` script, which is called by `ldap_probe.pl` when someone is to be notified about a problem. [Listing 19.3](#) contains a sample `notify.conf` file. This file contains the configuration information that describes whom should be notified for each type of failure detected by the `ldap_probe.pl` script.

Listing 19.1 The `ldap_probe.pl` Script

```
#!/usr/local/bin/perl

#
# usage: ldap_probe.pl host port DN [router]
#
# This script periodically probes an LDAP server to check whether
# it is still responding to queries. If the server does not
# respond, the notify.pl script is called to generate a
# notification. The notify.pl script is called on each up-down
# or down-up transition.
#
# For each probe, the script connects to the LDAP server
# running on the given host and port, and attempts to read
```

```

# the entry given by "DN". If the entry cannot be read, and
# the error returned indicates that the directory server
# could not be contacted, "router" (if given) is pinged. If
# the router cannot be pinged, then the script does not
# notify, on the assumption that the directory server is
# "hidden" behind a network failure. Otherwise, notification
# is performed.

$ping_timeout = 10;      # Wait 10 seconds for a response from ping
$test_interval = 60;     # 60 seconds between probes

# Check arguments

if ($#ARGV < 2 || $#ARGV > 3) {
    print "usage: ldap_probe.pl host port DN [router]\n";
    exit;
}

# Get arguments

$host = $ARGV[0];
$port = $ARGV[1];
$dn = $ARGV[2];
if ($#ARGV == 3) {
    $router = $ARGV[3];
} else {
    $router = "";
}

$is_down = 0;

# Loop forever
for (;;) {

    # Initialize state for this loop
    $prev_is_down = $is_down;
    $is_down = 0;
}

```

```

$do_check_router = 0;

$transition = "";

# Attempt to read the entry named by "DN"

$search_result = system "ldapsearch -h $host -p $port -s base -b \"\$dn\" \"(
➥ objectclass=*\")\\" cn > /dev/null 2>&1";

$search_result = $search_result / 256;

# Check for errors indicating that the server is not

# running or is unreachable, or that the domain name could

# not be looked up.

if ($search_result == 91                               # LDAP_CONNECT_ERROR

    || $search_result == 85                           # LDAP_TIMEOUT

    || $search_result == 81) {                         # LDAP_SERVER_DOWN

    # These errors are generated if the server is not running or

    # is unreachable, or if the domain name could not be looked

    # up.

    $do_check_router = 1;

} elsif ($search_result != 0) {

    # Some other error occurred.

    $is_down = 1;

}

# If the server is down or unreachable, check the router by

# pinging it (if a router address was provided). If the router

# is unpingable, we can't know about the state of the server.

if ($do_check_router) {

    if ($router ne "") {

        $ping_result = system "ping $router $ping_timeout > /dev/null 2>&1";

        $ping_result = $ping_result / 256;

        if ($ping_result == 0) {

            # Router was pingable, so assume that the LDAP

            # server is down.

            $is_down = 1;

```

```

        }

    } else {

        # No router address provided.

        $is_down = 1;

    }

}

# Did we just notice a transition?

if (!$prev_is_down && $is_down) {

    # Up - down transition

    $transition = "down";

} elsif ($prev_is_down && !$is_down) {

    # Down - up transition

    $transition = "up";

} else {

    $transition = " ";

}

if ($transition ne "") {

    # Call the notification script

    system ("notify.pl $host:$port ldap_probe $transition");

}

# Wait a while until testing again

sleep($test_interval);

}

```

Listing 19.2 The `notify.pl` Script

```

#!/usr/local/bin/perl

#
# usage: notify.pl identifier test transition
#
# This script reads the file notify.conf and locates lines where the
# identifier, test, and transition match those given as input to this

```

```

# script. For each match, the notification method given by the fourth
# argument is performed. For example, if notify.conf contains the
# following line:

#
#directory.example.com ldap_probe down mail bjensen@example.com "directory.example.com
# ↪ LDAP down"

#
# the script will send an e-mail message to bjensen@example.com
# with the text "directory.example.com LDAP down"
#
# There are three types of notification methods:
#
# mail:      an electronic mail message is generated and sent.
#
# page:      a text page is generated (you must supply your own
# command).
#
# "shell":   if the fourth argument begins with a slash (/), the
#             argument is treated as a shell command that is
#             executed. Any additional arguments on the line
#             are quoted and passed to the shell.
#
# See notify.conf for more details.

require "shellwords.pl";
require "ctime.pl";

#
# Check arguments

if ($#ARGV < 2) {
    print "usage: notify.pl entity testname transition\n";
    exit;
}

$entity = $ARGV[0];
$testname = $ARGV[1];
$transition = $ARGV[2];

```

```

# Open the configuration file

if (!open(CONF, "notify.conf")) {
    print STDERR "Cannot open configuration file notify.conf.\n";
    exit;
}

# Note the current time

$now = &ctime(time);

chop $now;

$found = 0;

while (<CONF>) {
    # Skip comment lines.

    if (/^ *#/) {
        next;
    }

    # Split the line according to Unix shell-quoting conventions

    @_ = &shellwords($_);

    if ($entity eq $_[0] && $testname eq $_[1] && $transition eq $_[2]) {
        # Found a match. Perform the notification.

        $found = 1;
        &do_notify;
    }
}

if (!$found) {
    # There were no matching lines. Generate a warning.

    # You could also perform a default notification here; e.g.,
    # send e-mail to the maintainer of the notify.conf file
    # to let him/her know about the error.

    print STDERR "Could not find a notification method for $entity $testname $transition\n";
}

# Subroutine to perform the notification.

```

```

sub do_notify {
    $method = $_[3];

    if ($method eq "page") {
        # The "page" notification method is not implemented. You
        # can activate it by adding code to dial out to your
        # paging service.

        # The pager telephone number is the 4th argument,
        # and the pager's PIN is the 5th argument. For
        # example, the following line in notify.conf might
        # be used:

        #dir.example.com ldap_probe down 9,1-800-555-1212 3233 "dir.example.com LDAP down"
        $pager = $_[4];
        $pager_pin = $_[5];
        $message = $_[6];
        print "Paging $pager, PIN $pager_pin, message \"$message\"\n";
        # Insert your custom paging code here.

    } elsif ($method eq "mail") {
        # Send an e-mail notification.

        $email_addr = $_[4];
        $message = $_[5];
        print "Sending e-mail to $email_addr, message \"$message\"\n";
        open(MAIL, "|/bin/mail $email_addr");
        print MAIL "To: $email_addr\n";
        print MAIL "Subject: $entity $testname $transition\n";
        print MAIL "\n";
        print MAIL "At $now\n";
        print MAIL "$message\n";
        close(MAIL);

    } elsif (substr($method, 0, 1) eq "/") {
        # Any method that begins with a / is treated as a shell command.
        shift(@_);
    }
}

```

```

    shift(@_);

    shift(@_);

    shift(@_);

    # Quote each argument.

    for ($i = 0; $i < @_; $i++) {

        $_[$i] = "\"$_[$i]\"";

    }

$not_args = join(" ", @_);

print "Sending the following command to the shell: $method $not_args\n";

system "$method $not_args";

}

}

```

Listing 19.3 The `notify.conf` Configuration File

```

# notify.conf

# This file contains the configuration information for notify.pl.

#
# Format:
#
# identifier test transition action [arguments...]
#
# Where:
#
# "identifier" is a string that identifies the service
# (typically a host name)
#
# "test" is a string that identifies the type of test performed
#
# "transition" is either "up" or "down"
#
# "action" describes the type of notification to perform. The
# notify.pl program knows about the following types:
#
#     "mail email-address message"
#
# Electronic mail is sent to "email-address". The text
# of the message is "message".
#

```

```
#      "/shell-command [arguments...]

# The command "/shell-command" is executed. Any arguments
# are passed to the shell command.

#
#

# Lines beginning with "#" are comments and are ignored.

dir.example.com:389 ldap_probe down mail bjensen@example.com "dir.example.com:389 down"
dir.example.com:389 ldap_probe up mail bjensen@example.com "dir.example.com:389 up"
dir.example.com:389 ldap_probe down /usr/bin/logger -p user.err "dir.example.com:389 down"
dir.example.com:389 ldap_probe up /usr/bin/logger -p user.err "dir.example.com:389 up"
```

Team LiB

◀ PREVIOUS

NEXT ▶

Performance Analysis

So far in this chapter, we've focused on techniques to monitor your directory service and notify you when a problem occurs. This type of reactive monitoring is essential to the success of your directory service, but it's only half of the picture. Your directory service will be more successful if you also monitor the performance of your directory servers, and use the raw performance data to anticipate necessary configuration changes and upgrades to your service.

When you initially deploy your directory service, you implicitly or explicitly perform some initial capacity planning. You decide how many servers to deploy; how much memory, disk, and network I/O capacity each server should have; and so on. You make these decisions on the basis of your knowledge of the directory-enabled applications that will initially be deployed, and perhaps build in some extra capacity to accommodate anticipated growth. Once your directory service is deployed, however, you will find that you need to revisit your assumptions about how your service will be used. The only effective way to do this is to observe the service in operation, and this means obtaining and analyzing usage and performance data, and drawing conclusions that help you in capacity planning.

In this chapter we focus on

- How to obtain the raw data essential to performance analysis and capacity planning
- How to digest and analyze performance data
- How to draw conclusions

Obtaining Raw Usage Data

There are two sources for raw usage and performance data: directory server access logs and operating system logs.

Directory Server Access Logs

If you use Netscape Directory Server, you can configure it to record every client access to a file. This file is usually called `access` and is contained in the `logs` directory beneath the server instance directory. If your server is not configured to record activity in the access log, you can use Netscape Server Administrator to enable such recording. Netscape Directory Server records the following events in its access log:

- Establishment of incoming connections.
- Closure of a connection.
- Receipt of an LDAP operation. Bind, unbind, search, compare, add, delete, modify, modify DN, and extended operations are all logged.
- The result of each LDAP operation sent to the client.

Each of these events is recorded on a separate line in the access log, marked with the date and time it occurred.

Access logs are stored on a disk local to the directory server machine. You'll want to periodically move them to a central location where you perform analysis, rather than performing analysis on the server itself. One common method of collecting directory server access logs is to periodically copy all of the access logs from your servers to a central location and delete the original copy on each server. Once the logs have been copied to the central location, you can perform whatever log analysis you wish, then delete the raw data logs. This task is made easier if the directory server software supports automatic log rotation, in which the active log file is closed and renamed when it reaches a predetermined size or age. With log rotation, you don't need to worry about the fact that the server is writing to the access log you're copying; you simply copy the rotated logs.

Other server software may require that you stop the server, rename the logs, and restart the server in order to rotate the log files. Consult your server documentation for specifics.

Operating System Logs

Many operating systems maintain usage counters and provide utilities for recording historical values of

those usage counters. For example, Solaris provides the sar utility, which can record many usage parameters of the operating system, and can generate tables and graphs of the data. Microsoft Windows systems provide the perfmon utility, which can perform many of the same tasks.

You should arrange to sample operating system parameters on a periodic basis (say, once every five minutes) and record the raw data to a log. Periodically you can produce a summary report of the raw data and delete the raw data log. The summary reports should be saved indefinitely so that you can review the long-term trends.

For example, the sar utility on Solaris can sample multiple operating system parameters. Typically, sar is run from the cron facility and samples all relevant parameters, storing the results in a file. The sar utility can later be run to extract information from that file.

Digesting and Analyzing Raw Performance Data

After you have the raw usage logs for your directory server in a central location, you need to make sense of the data. One way to do this is to divide the log file into portions of equal time, and summarize the activity for each period. For example, you might produce a summary of activity for each hour (the sample Perl log analysis utility included in this chapter does exactly this). For looking at longer-term trends, you can always combine hourly summaries to produce daily, weekly, or monthly summaries as you desire.

The Perl script `ldap_analyzer.pl`, shown in [Listing 19.4](#), can read a Netscape Directory Server access log and writes hourly summaries of activity using comma-separated value (CSV) format. CSV files can be read into spreadsheet programs like Microsoft Excel for analysis.

Listing 19.4 The `ldap_analyzer.pl` Script

```
#!/usr/local/bin/perl

#
# ldap_analyzer.pl
#
# Analyze Netscape Directory Server access logs
#
# Usage: perl ldap_analyzer.pl < log-file
#
# This script reads a Netscape Directory Server access
# log and outputs a CSV (comma-separated value) file that can
# be imported into a spreadsheet or database program for further
# analysis. The output file consists of a header line that gives
# a description of each column, then one row of values for every
# hour covered by the access log. The columns and their meanings
# are:
#
# Date/Time - The date and time of the period covered.
# Total Connections - The total number of connections handled.
# SSL Connections - The number of SSL connections handled.
```

```

# Operations - The total number of LDAP operations handled.

# Bind Operations - The total number of bind operations handled.

# V2 Binds - The number of LDAP version 2 bind operations handled.

# V3 Binds - The number of LDAP version 3 bind operations handled.

# Anonymous Binds - The number of anonymous bind operations handled.

# This number is always zero for Netscape Directory
# Server 4.x and earlier log formats.

# Non-Anonymous Binds - The number of nonanonymous bind operations
# handled. This number is always zero for Netscape
# Directory Server 4.x and earlier log formats.

# Unbinds - The number of unbind operations handled.

# Search Operations - The total number of search operations handled.

# Base-level Searches - The number of base-level searches handled.

# Onelevel Searches - The number of one-level searches handled.

# Subtree Searches - The number of subtree searches handled.

# Unindexed Searches - The number of searches that had to be serviced
# without the aid of an index.

# Avg Search ET - Average elapsed time for all search operations.

# Add Operations - Number of add operations handled.

# Avg Add ET - Average elapsed time for all add operations.

# Modify Operations - Number of modify operations handled.

# Avg Mod ET - Average elapsed time for all modify operations.

# Delete Operations - Number of delete operations handled.

# Avg Del ET - Average elapsed time for all delete operations.

# ModifyDN Operations - Number of modify DN (rename) operations handled.

# Avg ModDN ET - Average elapsed time for all modify DN operations.

# Compute an average.

sub compute_avg {
    my($total, $count, $avg);
    $total = $_[0];
    $count = $_[1];
    if ($count > 0) {

```

```

$avg = $total / $count;

} else {
    $avg = 0;
}

return $avg
}

# Print summary information for the time period.

sub print_summary {
    my($now);

    $now = $_[0];

    if ($now ne "xxx") {

        $av_srch_et = &compute_avg($tot_srch_et, $num_srchs);

        $av_add_et = &compute_avg($tot_add_et, $num_adds);

        $av_mod_et = &compute_avg($tot_mod_et, $num_mods);

        $av_del_et = &compute_avg($tot_del_et, $num_dels);

        $av_moddn_et = &compute_avg($tot_moddn_et, $num_moddns);

        print "$now,";

        print "$num_conns,";

        print "$num_ssl_conns,";

        print "$num_ops,";

        print "$num_binds,";

        print "$num_v2_binds,";

        print "$num_v3_binds,";

        print "$num_anon_binds,";

        print "$num_non_anon_binds,";

        print "$num_unbinds,";

        print "$num_srchs,";

        print "$num_base_srchs,";

        print "$num_onelvel_srchs,";

        print "$num_subtree_srchs,";

        print "$num_unindexed_srchs,";

        printf "%4f", $av_srch_et;

        print "$num_adds,";
    }
}

```

```

printf "%.4f,", $av_add_et;
print "$num_mods,";
printf "%.4f,", $av_mod_et;
print "$num_dels,";
printf "%.4f,", $av_del_et;
print "$num_moddns,";
printf "%.4f", $av_moddn_et;
print "\n";
}

&init_interval_counters();
}

# Print the header for the CSV file.

sub print_header {
    # Print header
    print "Date/Time,";
    print "Total Connections,";
    print "SSL Connections,";
    print "Operations,";
    print "Bind Operations,";
    print "V2 Binds,";
    print "V3 Binds,";
    print "Anonymous Binds,";
    print "Non-anonymous Binds,";
    print "Unbinds,";
    print "Search Operations,";
    print "Base-level Searches,";
    print "Onelevel Searches,";
    print "Subtree Searches,";
    print "Unindexed Searches,";
    print "Avg Search ET,";
    print "Add Operations,";
    print "Avg Add ET,";
    print "Modify Operations,";
}

```

```

print "Avg Mod ET,";

print "Delete Operations,";

print "Avg Del ET,";

print "ModifyDN Operations,";

print "Avg ModDN ET";

print "\n";

}

# Reinitialize all counters.

sub init_interval_counters {

    $num_conns = 0; # Number of non-SSL connections handled

    $num_ssl_conns = 0; # Number of SSL connections handled

    $num_ops = 0; # Number of operations handled

    $num_binds = 0; # Number of bind operations handled

    $num_v2_binds = 0; # Number of LDAPv2 binds handled

    $num_v3_binds = 0; # Number of LDAPv3 binds handled

    $num_anon_binds = 0; # Number of anonymous binds handled

    $num_non_anon_binds = 0; # Number of non-anonymous binds handled

    $num.unbinds = 0; # Number of unbind operations handled

    $num_srchs = 0; # Number of search operations handled

    $num_base_srchs = 0; # Number of base-level search operations handled

    $num_onelvel_srchs = 0; # Number of onelvel search operations handled

    $num_subtree_srchs = 0; # Number of subtree search operations handled

    $tot_srch_entries = 0; # Total number of search entries handled

    $num_unindexed_srchs = 0; # Total number of search entries handled

    $tot_srch_et = 0; # Total elapsed time spent on searches

    $num_adds = 0; # Number of add operations handled

    $tot_add_et = 0; # Total elapsed time spent on adds

    $num_mods = 0; # Number of modify operations handled

    $tot_mod_et = 0; # Total elapsed time spent on mods

    $num_dels = 0; # Number of delete operations handled

    $tot_del_et = 0; # Total elapsed time spent on dels

    $num_moddns = 0; # Number of moddn (rename) operations handled

    $tot_moddn_et = 0; # Total elapsed time spent on moddns

    $num_entries = 0; # Number of entries sent to clients
}

```

```

}

&print_header();

$curr_date_time = "xxx";

while (<>) {

    # Check for date/time in correct format

    if (/^[\(\d{2}\)\/(\w{3})\/(\d{4}):\(\d{2}\):\(\d{2}\):\(\d{2}\) [+-](\d{4})\]$/) {
        $day = ${1};
        $month = ${2};
        $year = ${3};
        $hour = ${4};
        $minute = ${5};
        $second = ${6};
        $gmt_offset = ${7};

        # Detect when we have rolled over to a new hour

        $date_time = "${month}/${day}/${year} ${hour}:00";

        if ($curr_date_time ne $date_time) {
            &print_summary(${curr_date_time});
            $curr_date_time = $date_time;
        }
    }

    # Now look for specific types of log format

    # BIND operation

    if (/.*/ conn=(\d+) op=(\d+) BIND dn="(.*") method=(\d+) version=(\d+).*$/) {
        $conn = ${1};
        $op = ${2};
        $binddn = ${3};
        $bindmethod = ${4};
        $ldapversion = ${5};
        $num_ops++;
        $num_binds++;

        if ($ldapversion == "2") {
            $num_v2_binds++;
        } elsif ($ldapversion == "3") {

```

```

$num_v3_binds++;

}

}

# UNBIND operation

if (/.* conn=(\d+) op=(\d+) UNBIND$/) {

$num_ops++;

$num_unbinds++;

}

# SEARCH operation

if (/.* conn=(\d+) op=(\d+) SRCH base="( .*)" scope=(\d+) filter="( .*)"$/) {

$conn = ${1};

$op = ${2};

$base=${3};

$scope=${4};

$filter=${5};

$num_ops++;

$num_srcchs++;

# What is the search scope?

if ($scope == 0) {

$num_base_srcchs++;

} elsif ($scope == 1) {

$num_onelvel_srcchs++;

} elsif ($scope == 2) {

$num_subtree_srcchs++;

}

}

# ADD operation

if (/.* conn=(\d+) op=(\d+) ADD dn="( .*)"$/) {

$conn = ${1};

$op = ${2};

$dn=${3};

$num_ops++;

$num_adds++;

}

# DEL operation

```

```

if (/.* conn=(\d+) op=(\d+) DEL dn="( .* )"$/) {
    $conn = ${1};
    $op = ${2};
    $dn=${3};
    $num_ops++;
    $num_dels++;
}

# MOD operation

if (/.* conn=(\d+) op=(\d+) MOD dn="( .* )"$/) {
    $conn = ${1};
    $op = ${2};
    $dn=${3};
    $num_ops++;
    $num_mods++;
}

# MODRDN operation

if (/.* conn=(\d+) op=(\d+) MODRDN dn="( .* )"$/) {
    $conn = ${1};
    $op = ${2};
    $dn=${3};
    $num_ops++;
    $num_modrdns++;
}

# RESULT log entries. There are several variants.

if (/.* conn=(\d+) op=(\d+) RESULT err=(\d+) tag=(\d+) nentries=(\d+) etime=(\d+).*$/) {
    $conn = ${1};
    $op = ${2};
    $err = ${3};
    $tag=${4};
    $nentries = ${5};
    $etime = ${6};
    $num_results++;
}

# BIND result

```

```

if (*.* conn=(\d+) op=(\d+) RESULT err=(\d+) tag=97 nentries=(\d+) etime=(\d+)( dn=( .
➥ +))*$/) {
    $dn = ${7};
    $tot_bind_et += $etime;
    if (length($dn) > 0) {
        if (${dn} =~ "\\"") {
            $num_anon_binds++;
        } else {
            $num_non_anon_binds++;
        }
    }
}

# SEARCH result

if (*.* conn=(\d+) op=(\d+) RESULT err=(\d+) tag=101 nentries=(\d+) etime=(\d+)( notes=( .
➥ \w+))*$/) {
    $tot_srch_entries += $nentries;
    $tot_srch_et += $etime;
    $notes=${7};
    if (${notes} eq "U") {
        $num_unindexed_srchs++;
    }
}

# MOD result

if (*.* conn=(\d+) op=(\d+) RESULT err=(\d+) tag=103 nentries=(\d+) etime=(\d+)( notes=( .
➥ .+))*$/) {
    $tot_mod_et += $etime;
}

# ADD result

if (*.* conn=(\d+) op=(\d+) RESULT err=(\d+) tag=105 nentries=(\d+) etime=(\d+)( notes=( .
➥ .+))*$/) {
    $tot_add_et += $etime;
}

```

```

# DELETE result

if (/.* conn=(\d+) op=(\d+) RESULT err=(\d+) tag=107 nentries=(\d+) etime=(\d+)$/) {
    $tot_del_et += $etime;
}

# MODDN result

if (/.* conn=(\d+) op=(\d+) RESULT err=(\d+) tag=109 nentries=(\d+) etime=(\d+)$/) {
    $tot_moddn_et += $etime;
}

# COMPARE result

if (/.* conn=(\d+) op=(\d+) RESULT err=(\d+) tag=111 nentries=(\d+) etime=(\d+)$/) {
    $tot_compare_et += $etime;
}

# EXTOP result

if (/.* conn=(\d+) op=(\d+) RESULT err=(\d+) tag=120 nentries=(\d+) etime=(\d+)$/) {
    $tot_extop_et += $etime;
}

}

# connection initiation log entry

if (/.* conn=(\d+) fd=(\w+)( SSL)? connection from (.*) to (.*)$/)
{
    $conn = ${1};
    $fd = ${2};
    $slot = ${3};
    $ssl = ${4};
    $remote_ip = ${5};
    $local_ip = ${6};

    $num_conns++;

    if (${ssl} eq " SSL") {
        $num_ssl_conns++;
    }
}

```

```

# connection closure variant one

if (/.* conn=(\d+) op=(.*) fd=(.*) closed - (.*)$/) {
    $conn = ${1};
    $op = ${2};
    $fd = ${3};
    $reason = ${4};

}

# connection closure variant two

if (/.* conn=(\d+) op=(.*) fd=(.*) closed error (.*) \((.*)\) - (.*)$/) {
    $conn = ${1};
    $op = ${2};
    $fd = ${3};
    $expl = ${4};
    $reason = ${5};

}

} else {
    # Unknown format
}
}

# Print summary for the final time segment.

&print_summary(${date_time});

```

[Listing 19.5](#) shows what output from the Perl script looks like. The output is terse and designed to be imported into a spreadsheet for further analysis. For example, the graph in [Figure 19.9](#) was generated with Microsoft Excel. It shows the number of bind operations serviced by the server during a 24-hour period.

Listing 19.5 Sample Output from the `ldap_analyzer.pl` Script

```

Date/Time,Total Connections,SSL Connections,Operations,Bind Operations,V2 Binds,V3 Binds,
➡ Anonymous Binds,Non-anonymous Binds,Unbinds,Search Operations,Base-level Searches,
➡ Onelevel Searches,Subtree Searches,Unindexed Searches,Avg Search ET,Add Operations,Avg
➡ Add ET,Modify Operations,Avg Mod ET,Delete Operations,Avg Del ET,ModifyDN Operations,
Avg
➡ ModDN ET

Nov/18/2001 15:00,316,0,15351,79,75,4,0,0,79,15109,15085,0,24,0,0.0007, 0,0.0000,84,0.
➡ 0357,0,0.0000,0,0.0000

Nov/18/2001 16:00,284,0,6187,47,43,4,0,0,47,6009,5985,0,24,0,0.0008, 0,0.0000,84,0.0238,0,

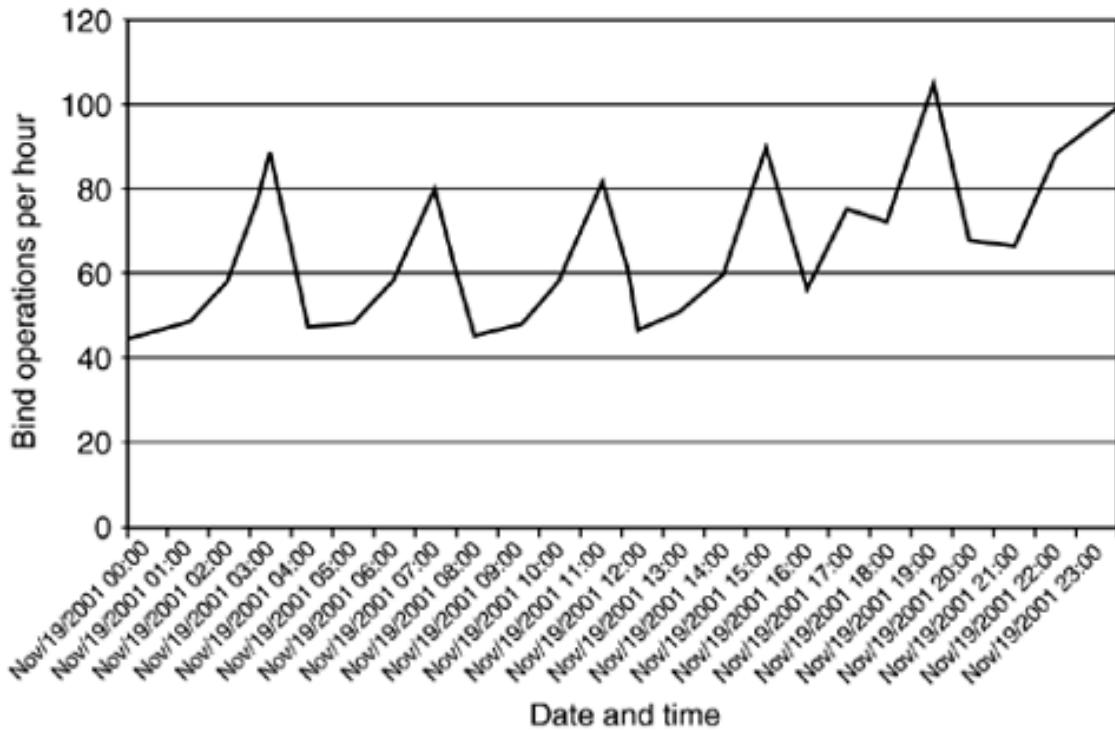
```

→ 0.0000,0,0.0000

Nov/18/2001 17:00,287,0,5432,49,44,5,0,0,49,5247,5223,0,24,0,0.0006, 0,0.0000,87,0.0230,0,

→ 0.0000,0,0.0000

Figure 19.9. Bind Operations Serviced in a 24-Hour Period



In [Figure 19.9](#), notice that the number of bind operations increases every four hours. In this example, which is drawn from a real server installation, an automated synchronization process runs every four hours and performs multiple bind operations, resulting in the usage spikes shown in the figure. Graphing the CSV files generated by the `ldap_analyzer.pl` script can help you visualize the types of load your directory server is handling.

Drawing Conclusions

Now that you have obtained the raw usage data and digested it into a format that is easily understood, what conclusions can you draw?

Spotting Problems

If your directory server is experiencing performance problems, you might try graphing the number of operations serviced by your server, and compare the result to a graph of the average elapsed time for operations. If your server is simply being overwhelmed by client load, you'll see a close correspondence between periods of high usage and poor performance.

On the other hand, if your directory performance degrades without a corresponding increase in client load, you may want to examine operating system parameters such as CPU utilization, memory utilization, and disk throughput. You may discover, for example, that another process on the server is competing for machine resources.

Spotting Trends

If you have been collecting usage data for a while, you can graph usage over longer time periods and start to look for long-term trends. For example, if you have hourly usage data for the previous six months, you might sum the usage data on a weekly basis and examine how directory usage is changing over time. If

you have an idea of the maximum throughput your server can handle (perhaps you measured this empirically during your directory pilot), you can predict the point in time when your server will become overloaded. Be aware, however, that the deployment of a new application may cause an unanticipated increase in directory usage.

Monitoring Checklist

To review, here are the steps involved in designing and implementing a monitoring strategy for your directory service:

- Understand the monitoring tools at hand. Do you have, or do you plan to run, a full-fledged network management system? Or do you plan to develop your own standalone monitoring tools? Does your directory server software support SNMP?
- Make sure that any custom-built monitoring tools simulate actual directory operations.
- Decide what will be considered a failure of the directory. Set acceptable performance levels, taking into account any service-level agreements that may be in place.
- Implement log file analysis.
- Implement OS performance data analysis.
- Implement indirect monitoring.
- Decide on a notification strategy, implement notification methods, and develop a test plan for them.
- When an incident occurs, keep detailed records and use the knowledge gained while resolving the problem to improve the quality of your directory service.
- Analyzing usage and performance data collected from your directory server and operating system can provide you valuable insight into how your directory is being used, and how it is performing.

Further Reading

How to Manage Your Network Using SNMP: The Networking Management Practicum. M. T. Rose and K. McCloghrie, Prentice Hall, 1995.

Netscape Directory Server 6 Administrator's Guide: Chapter 13, Monitoring Directory Server Using SNMP. Available on the World Wide Web at <http://enterprise.netscape.com/docs/directory>.

SNMP, SNMPv2, SNMPv3, and RMON 1 and 2. W. Stallings, Addison-Wesley, 1999.

Total SNMP: Exploring the Simple Network Management Protocol. S. J. Harnedy, Prentice Hall, 1997.

Understanding SNMP MIBs. D. Perkins and E. McGinnis, Prentice Hall, 1996.

Looking Ahead

In this chapter we have discussed approaches you might use to monitor your directory service, as well as some general techniques for notifying people of failures. We have also discussed ways to collect and analyze performance data. In [Chapter 20](#), Troubleshooting, we will focus on specific techniques for remedying problems you might encounter with your directory data, your directory server software, and the hardware on which your directory service runs.

Chapter 20. Troubleshooting

- Discovering Problems
- Types of Problems
- Troubleshooting and Resolving Problems
- Troubleshooting Checklist
- Further Reading
- Looking Ahead

From time to time during your directory service's lifetime, things will go wrong. Sometimes these problems will degrade performance. Sometimes they will cause the directory to fail completely. When something does go wrong, your objective should be to minimize the damage, return the directory to full service as quickly as possible, and understand the problem so that you can take steps to prevent its recurrence.

Directory problems can be broken down into four major groups:

1. Directory outages as a result of hardware or software failure
2. Performance problems
3. Problems with directory data
4. Security problems

In this chapter we examine each of these categories in detail and provide examples of each. We also suggest a systematic approach for dealing with directory problems as they arise. We conclude with troubleshooting checklists that help you resolve some typical directory problems.

Discovering Problems

How do you discover directory problems in the first place? A problem might come to the attention of the directory administrator in any of the following ways:

- The monitoring system (if you have one) may automatically detect a failure or degradation of the directory and notify you about it.
- The maintenance or operations staff may notice problems with the directory as they go about their routine directory maintenance functions.
- Administrators of dependent services such as e-mail servers may notice and report problems.
- End users may notice a problem and report it to your Help Desk. The problem might be described by end users as a failure in a dependent system, such as an Internet Message Access Protocol (IMAP) server, or as a problem with a desktop application that relies on the directory.
- Your performance monitoring system, if you have one, may alert you to degraded performance on one or more of your directory servers.

A failure may be detected and reported via all these methods nearly simultaneously. For example, if a directory server becomes unavailable, you might be notified by your network management system (NMS) software at the same time your Help Desk receives numerous calls from end users. Those end users may report problems with any of the dependent applications, such as address books, e-mail servers, and authenticated Web server access. Your operations staff might also be unable to run a regular data update procedure.

Receiving multiple reports of system failure from a variety of sources is typical of distributed systems in which the individual portions of the system are interdependent. Part of the problem resolution process, therefore, involves correlating the various reports and identifying the underlying causes. As soon as you know what the real problem is, you can then inform users of all the affected services.

Ideally you should strive to eliminate the possibility that your users will discover problems before you do. You can accomplish this through a well-designed, proactive monitoring system. Careful planning is required, but the payoffs are significant. More information on proactive monitoring can be found in [Chapter 19, Monitoring](#).

When there is a problem, you should clearly communicate the following information to your users and frontline staff:

- The fact that there is a known problem
- How long you expect it to take to fix the problem
- Any workarounds or alternative services that your users can employ in the meantime

Plan well in advance how you will notify your users about directory problems. Common methods include publishing outage information to Web pages, posting outage information to Usenet newsgroups, and providing status information in a recorded telephone message. Whatever method you use, make sure that it does not depend on the directory itself.

Providing good information to your end users and frontline support staff serves two important purposes. First, it allows your end users to remain productive because they know which applications are affected by the failure and how long the applications will be unavailable. Second, your frontline support staff can be more effective in helping your end users understand the implications of the outage. Your end users will also have a better overall impression of your directory service and dependent applications.

Types of Problems

As mentioned earlier in this chapter, there are four main classifications of directory problems. In this section we provide an overview of the various issues that can arise with each type of problem.

Directory Outages

The first type of problem we explore is the directory outage. In an *outage*, part or all of your directory service becomes unavailable. Outages can occur when one or more of the directory servers have become unreachable because of a network problem, or because the directory server software or hardware has failed in some way. When an outage occurs, users receive no service at all.

Causes

Causes of directory outages fall into two broad categories: hardware failures and software failures. Hardware that can fail includes network components such as routers, switches, network interface cards, and network cabling; and server components such as CPU boards, disk drives, memory, power supplies, and so on. An outage can also result from a power outage. Software failures can include failures of the operating system itself or of the directory server software. Other software running on the server can also malfunction and cause the directory server software to fail or become unresponsive.

Implications

When a directory outage occurs, your users and directory-enabled applications receive no directory service at all. Because LDAP is a client/server protocol, outages are generally noticed as a failure to connect to the directory service. Outages produce three different symptoms: connection timeouts, refused connections, and hung connections.

In a *connection timeout*, the client attempts to open a connection to the server but receives no response at all. In this case the client waits a fixed period of time for a response. Eventually the client times out and reports an error; however, this timeout period is typically on the order of minutes—longer than most users are willing to wait. Connection timeouts can occur if the directory server becomes unreachable because of a network failure, if the server itself is powered off, or if the operating system has crashed.

With a *refused connection*, the directory server's operating system is operating correctly and the server is reachable on the network, but no server process is listening for incoming LDAP connections. In this case the operating system's TCP/IP stack returns a "connection refused" message to the client. Refused connections typically happen when the directory server process has terminated abnormally because of a software bug. They can also occur if the server machine has restarted, perhaps because of a power failure, but the directory server software is not configured to start automatically upon system reboot.

With a *hung connection*, the server is reachable and a directory server process is running, but the service malfunctions. In this case the connection is still accepted, but no further data flows from the server to the client. Hung connections can result from software bugs or hardware failure, such as a hung SCSI (Small Computer System Interface) bus on the server.

Resolution

If hardware failure is the underlying cause of an outage, the usual course of action is to replace and restart the failed hardware. For example, if the cause is determined to be a bad network cable, the obvious remedy is to plug in a new cable.

In some cases, however, outages caused by hardware failure can be more challenging to correct, either because the needed replacement parts are not available or because the directory data was damaged (which can happen if a disk drive fails). Ideally you should maintain a stock of spare parts or purchase an on-site service agreement with a guaranteed maximum response time. Depending on the availability requirements and any service-level agreements you might have, even a short outage may be unacceptable. In this case you need to look into providing a highly available directory service, as discussed in [Chapter 17, Backups and Disaster Recovery](#).

When software failure is the root of a directory outage, more vigilance is required. Simply restarting the failed software component may bring the service back online, but if the underlying condition that triggered the software failure is still present, the service may soon fail in exactly the same way. For example, suppose that the directory server's cache is configured to be much too large and the server becomes unresponsive because of excessive paging activity. Restarting the server would probably make the service perform acceptably for a while, but as soon as the cache filled, the system might again experience the problem.

When the cause of a software outage is unclear, try to note as much detail about the state of the service while the problem is occurring. Is the server process running? If so, how much memory is it using? Is it consuming any CPU time? Is there disk activity? Do the operating system or directory service log files contain any useful information? If the cause of the problem is still unclear, it may be appropriate to restart the service and watch carefully to see whether the service degrades slowly or fails suddenly. For example, you might notice that the service fails whenever a particular query arrives at the server from a particular directory-enabled application. Information like this can be valuable when you're working with your vendor's technical support organization.

Performance Problems

Another common type of directory problem is poor performance, which can manifest itself in various ways. The overall performance of your directory may be poor, or a specific type of directory operation such as a delete operation might be slow. Performance problems may be consistent, or they may be intermittent. Troubleshooting these problems requires careful analysis and attention to detail.

Causes

Misconfigured software is the most common cause of directory performance problems. A misconfigured directory server might not perform optimally, or it might not function at all. For example, most directory server software uses a RAM-based cache to improve performance. If this cache is too small, the directory's performance can suffer, perhaps to the point where the server seems to be hung. To correct this problem, increase the size of the cache. On the other hand, if the cache configuration is too large, the server's virtual memory system may experience excessive paging, thereby slowing performance. Most operating systems offer a utility for observing virtual memory system paging activity (perfmon on Windows NT and vmstat on Solaris, for example).

Another common misconfiguration problem results from not maintaining appropriate indexes for the types of searches your server handles. Netscape Directory Server 6, for example, permits searches on any attribute. However, search performance is poor on unindexed attributes because the server might need to look through every entry in the database to locate matching entries. If your server takes a long time to respond to search requests and

consumes a large amount of CPU time, you may want to check whether clients are using unindexed attributes in search filters.

If you notice that your Netscape Directory Server 6 is performing poorly, you can examine the server access log for clues. The access log is found in the `logs` directory beneath the server root and is named `access`.

First look in the access log for the string "notes=U" in the `RESULT` log entry for search operations. This string indicates that the server was unable to use an index to service the search operation.

Note that Netscape Directory Server 6 writes two lines to the server log for each client operation it services. The first log entry is written when an operation is received from a client, and the second is written when the result message is sent to the client. Because the server can handle many client operations concurrently, log entries from other client operations may be interleaved in the log, and the two entries for a given client operation may not be adjacent.

To correlate a client operation initiation log entry with its corresponding result message, you need to examine the connection and operation information. Each incoming connection to the directory server is assigned an increasing connection number, which is logged as `conn=c` in the access log, where `c` is the connection number. Each operation serviced for a given connection is assigned an increasing operation number, which is logged as `op=o`, where `o` is the operation number. The following example shows what Netscape Directory Server might log in response to a client's search operation:

```
[11/Jan/2002:08:05:37 -0800] conn=24 op=8 SRCH  
base="dc=example,dc=com" scope=2 filter="(description=*engineer*)" attrs=ALL  
other log entries...  
  
[11/Jan/2002:08:05:44 -0800] conn=24 op=8 RESULT err=0 tag=101  
nentries=7 etime=7 notes=U
```

Notice that the client has issued a search with the filter `(description=*engineer*)`, and that the corresponding `RESULT` log message indicates that no index was available to service the search operation (because `notes=U` was logged). Netscape Directory Server 6 requires a substring index on the `description` attribute to service this search efficiently. In this case, if clients frequently perform substring searches on the `description` attribute, consider adding such an index to the server's configuration.

Also examine the elapsed time for each operation, recorded in `RESULT` messages as the string `etime=n`, where `n` is the elapsed time for the operation in seconds. In the previous example, notice that the search operation took seven seconds to complete, which is longer than expected. Most directory operations should complete in less than two seconds. If you find that some operations are taking longer to complete, you should verify that the proper indexes are being maintained if the operation is a search, or that the server is not overloaded if the operation is an update. The following Unix command will display a summary of elapsed times for all operations in the access log:

```
grep RESULT access | awk '{print $9}' | sort | uniq -c
```

The output from the command might look like this:

```
32924 etime=0  
112 etime=1  
1 etime=2  
1 etime=3
```

This output indicates that 32,924 operations completed in less than one second, 112 operations completed in one second, and only two operations took longer than one second. This type of distribution is what you should expect to see on a server that is performing well. If you run this command periodically against your access log, you can quickly determine if any operations are running slowly, warranting further investigation.

Your software may also encounter a specific limit within either the server software or the operating system. For example, most versions of Unix have a hard limit on the number of file descriptors that a single Unix process may use (one file descriptor is used for every open TCP/IP connection and every open file). This limit is usually in the thousands of connections, but it may be altered via Unix commands.

Another type of limitation you may encounter is the size of the TCP listen queue. This parameter controls how many incoming TCP connections can simultaneously be in the process of being opened. TCP connections are usually established quickly and are removed from the listen queue as soon as they are completely established. However, if many clients attempt to connect to a server at the same time, or if a problem prevents the connection from being quickly established, the listen queue can fill up and new incoming connections may appear to hang temporarily until the queue drains.

Older versions of the Unix operating system had a small listen queue (often fixed at only five incoming connections) that was inadequate for high-volume network servers. Newer operating system versions are configurable, and you may need to significantly increase the size of the listen queue. HTTP servers are particularly susceptible to this problem because Web browsers tend to open many short-lived connections. It's often necessary to increase the listen queue to 64 connections or more on a heavily used Web server. LDAP clients typically use longer-lived connections, but you still may need to increase the listen queue size on an LDAP server.

If your server software runs inside a single process, it is subject to an operating system connection limit. This means that there is a cap on the maximum simultaneous number of connections that can be handled by a single process. If there are so many clients that your server encounters this limit, you will notice that the server refuses connections (clients receive a "connection refused" error from the TCP/IP stack). The proper action in this case is to increase the limit, add additional replicas to handle the load, or reduce the number of client connections (if the client software opens connections unnecessarily or neglects to close connections when finished).

Some vendors provide utility software that can help you identify problems with operating system tuning parameters, such as an incorrectly sized listen queue, and can recommend more appropriate settings. For example, Netscape Directory Server 6 is bundled with a

utility named dsktune that analyzes your operating system. It verifies that all required patches are installed, checks that the per-process limit on file descriptors is set appropriately, and confirms that several important TCP parameters are set to values that will yield optimum performance.

Finally, you may encounter performance-degrading software bugs in the directory server software or the operating system. Software vendors increasingly are using the World Wide Web to distribute patches and publish knowledge bases full of information about their products. In addition, Usenet newsgroups are a tremendous resource for learning about known bugs, workarounds, and patches.

If you find a previously unknown bug, make sure that you report it to the software vendor in as clear a fashion as possible so that the bug can be fixed. For information on submitting a good bug report, see [Chapter 19](#), Monitoring.

Implications

The implications of performance problems can range from slight degradation of the service to outright failure. The symptoms can affect all users equally, or they might affect only a subset of directory users. For example, whereas a misconfigured cache can result in poor performance for all users and directory-enabled applications, a missing attribute index might result in poor search performance for only users and applications who search on the unindexed attribute (unless many users do so, in which case the server's overall performance may suffer).

Resolution

When you're resolving performance problems, it's important to proceed logically and deliberately. Take notes that describe the problems exactly as reported or observed, and then try to reproduce the problem yourself. For example, if your users complain that address book searches in Netscape 7 take a long time, try the same search yourself. Ask the users how the search dialog is configured and perform the same search. If you can reproduce the problem, you're well on your way to understanding the root cause. If you can't reproduce the problem, ask yourself what's different about your environment and the user's environment. Are you connected to the same server? Are you authenticated or bound anonymously? Are you closer to the server within the network? Does the server access log contain any telltale clues? Try to eliminate each difference, one by one, until you can duplicate the problem.

Remedying the problem may be simple if a small configuration change is required, or it may be complicated if a bug has been discovered in your software. If no fix is available, is there a workaround? Can you mitigate the effects of the problem by reconfiguring your directory in some way? For example, installing more memory, adding another CPU, or adding an additional replica may provide the additional capacity you need if you encounter a limit on your software.

No matter what the remedy, take the time to document the problem, the cause, and your workaround and/or long-term fix. Software problems can be complex, and the more you can share troubleshooting knowledge with your peers, the more effective your organization will be at providing a high-quality service. These details can also be useful to the operating system or directory server software vendor if a software bug is the root cause.

Tip

It's helpful to understand and be able to interpret the types of information that

your server software can log. For example, Netscape Directory Server writes detailed information to its access log (refer to [Chapter 19](#), Monitoring, for information on the access log and techniques for analyzing it).

The access log can help you understand problems. For example, if a user complains of slow search performance, you can search the logs for the IP address of the user's machine. When you find the log entries corresponding to the user's session, you can see exactly the type of search base and search filters that the user's client software presented to the server. This information may help you determine whether the problem is the result of a misconfiguration in the user's software or a problem with the server itself.

Problems with Directory Data

Directory data problems may be the result of missing, extra, or incorrect information. In the worst case, the database files that your directory server software uses may become corrupted because of software bugs, operating system bugs, or operator errors. In our experience with actual directory deployments, this is the most common type of problem.

Data problems are often a consequence of another problem, such as misconfigured software. In other cases, data problems can result from incorrect actions on the part of data administrators. Problems with data itself can also be a cause of many other problems. For example, if access control attributes have been erroneously changed or removed, users and applications may not be able to access needed directory entries.

Causes

When incorrect data appears in your directory, someone or some process must have put it there. For example, a confused departmental administrator might remove the entry for an active employee instead of a terminated employee. On a larger scale, an automated update process that reconciles database records from a human resources database might, as a result of a bug, place incorrect employee information in the directory or remove needed information.

Typical monitoring software won't detect this type of problem unless the data is so damaged that the server crashes or cannot even start up. You will usually learn of this type of problem via end-user reports unless you proactively monitor data quality.

To be more proactive, you might consider developing tools to monitor the quality of data in your directory or build data validation tools into the software that you can use to synchronize your directory with external data sources. Such tools can detect problems with data before they are noticed by users. More information on data quality monitoring can be found in [Chapter 18](#), Maintaining Data.

Implications

When incorrect data ends up in your directory, dependent applications can start behaving incorrectly. For example, if your directory is used to authenticate users accessing your internal Web servers, but some users have been incorrectly removed from the directory, they will be unable to access the protected resources. Or if a user's directory entry has been removed, e-mail destined for that user may be returned to the sender. In general, if the directory appears to be operating correctly but one or more of your users are having

problems, check the contents of the relevant directory entries.

Even more subtle errors can occur if the directory holds information about network resources such as file servers and printers. If the directory entries corresponding to these devices are removed or damaged in some way, the services provided by those devices may become unavailable.

If database files become corrupted, symptoms may be either obvious or subtle. All the entries in the directory may disappear (which is easy to notice), or certain entries may simply not be returned when certain types of searches are performed. Robust server software prevents these types of inconsistencies from arising as a part of normal operation, but operator mistakes can cause any number of unanticipated problems. For example, although Netscape Directory Server uses a transactional database that ensures data consistency even in the case of operating system crashes, manually removing one of the attribute index files and restarting the server can cause unexpected and unwanted results.

If the corruption is subtle, it may go unnoticed for some time. When dealing with corrupted data, always be open to the possibility that the damage actually occurred some time ago and has only now been noticed.

Resolution

If you determine that you have a problem with the data in your directory, the first thing to do is determine the extent of the damage. To do this, of course, you need to have some idea of what should actually be in the directory. A good starting point is to look at the directory contents. Do you see approximately the correct number of entries in your directory? If you see too few entries, indicating that entries have been erroneously deleted, it may be prudent to shut down certain dependent services. For example, if you know that the entries for an entire department are missing from the directory, you should probably shut down the servers that handle e-mail for those people. If the mail server cannot locate a user's e-mail address in the directory, it may incorrectly conclude that there is no such user and return the mail to the sender.

Sometimes the safest thing is to shut down the affected servers. Directory-enabled applications generally notice that the directory is unavailable, report a meaningful error, and retry the operation later. However, if the data is incorrect or missing but the directory is not shut down, applications may behave incorrectly.

As soon as you know the extent of the damage, you need to set about repairing it. How you do this depends on the damage and the knowledge you have about the correct contents of the directory. For example, if only a single user's entry has been deleted, it's probably most appropriate simply to re-create the entry. On the other hand, if your directory's entire contents have been wiped out by a buggy automated update process, you need to restore your data from a comprehensive source such as a set of backup files or tapes. We suggest that any update scripts you develop yourself include the capability to log their actions to a file so that you can analyze the scripts' actions later if necessary.

After you restore your directory, you need to understand how the damage happened. Did incorrectly configured access control allow removal of an entry? Did a data merge process go awry? You need to examine log files and other records to determine when and how the damage occurred. This step is important for preventing similar problems. As we've learned in our own deployment experience, problems rarely resolve themselves permanently!

Security Problems

The final category of problems is related to security. The most serious type of security problem is unauthorized access to directory data. An attacker may attempt to compromise the security of the directory with the intent of reading or damaging sensitive directory data or rendering the service useless for other users via a denial-of-service attack. The topics of directory access control and security are covered in detail in [Chapter 12](#), Privacy and Security Design. The steps outlined there should protect you against many common types of break-ins, but how do you notice and respond to security problems if they occur in spite of your best efforts?

A good way to detect compromised security is to be on the alert for telltale signs that the directory has been tampered with. Such signs, often subtle, might include access to your directory from an unexpected location on the Internet (if your directory is accessible from the Internet at all), or a report from a user that his or her directory entry has been altered unexpectedly. This is often just a consequence of a normal automated update, but it can also signal that the user's password or other credentials have been compromised. If you suspect unauthorized access, directory server logs can help track down the date and time when the tampering occurred and the origin of the LDAP connection.

A denial-of-service attack, on the other hand, has one purpose: to render the directory unusable. The attacker seeks to consume all available resources, perhaps by issuing thousands of repeated unindexed searches against the directory, or by exploiting a known bug in the directory or operating system software and causing a crash.

Causes

If your directory is directly accessible from the Internet, it may be subject to attacks from any place in the world. Or if your directory provides authentication and personalization services for a Web-based application that is Internet-accessible, an attack on the Web application may overwhelm your directory server.

On the other hand, if your directory is accessible only from inside your corporate network, you are less susceptible to attack from the outside—but you are not immune. Given enough time and motivation, disgruntled employees can certainly wreak havoc on a directory server.

What motivates people to mount denial-of-service attacks or attempt to break in to your directory? In some cases the sheer challenge of breaking in is sufficient. In other cases a disgruntled employee may be angry enough to attempt to compromise your directory as a way of achieving revenge.

Implications

A denial-of-service attack can render a directory unresponsive by consuming excessive resources, or it can take down a directory by exploiting known bugs. A security breach is serious, especially if your directory is used for authentication and access control for your critical business resources. The implications of compromised security are highly dependent on the type of data stored in your directory.

Resolution

Discovering a denial-of-service attack is usually simple because the affected servers and dependent services become unresponsive or unavailable. A well-designed monitoring system will note sudden peaks in server load. When you suspect a denial-of-service attack, be sure to save any relevant log files from the time of the attack. The logs may be useful in identifying the source of the attack launched against your servers. Be aware, however, that distributed denial-of-service attacks enlist compromised systems, or "zombies," to do their

dirty work. The original source of the attack may not be reflected in your logs.

If you determine that the attack originated from a location inside your company, you probably have some recourse to stop the attacker. Be careful when tracing the attack back to the original source, however; the fact that a particular machine was the origin of the attack doesn't necessarily mean that the owner was the attacker. A skilled hacker will cover his footprints carefully and avoid using his own desktop machine to originate an attack.

Tip

If the origin of the attack is outside your company, you might be able to use a firewall product such as Cisco's PIX Firewall to block access to your directory from the originating network.

It's also entirely possible for a user to accidentally cause a heavy load on the directory without knowing it. For example, if a user attempts to search the directory for an unindexed attribute but grows impatient and submits the search several times, the directory may become less responsive. In other cases, clever users may write scripts that collect directory data for legitimate purposes, but do so inefficiently or too frequently. These are not malicious attacks, and you need to be aware of this possibility when you're tracing the problem to its source.

Reconfiguring the directory to index the attribute or setting smaller administrative limits may be an effective way to protect the directory from such inadvertent denial-of-service attacks. Setting reasonable size limits (which control the number of entries returned in response to a search request) and time limits (which control the maximum amount of time the directory will spend responding to a search request) can help make your directory more robust. However, not all directory software can be configured in this manner, so be sure to consult your documentation.

If you suspect that the security of your directory has been compromised in some way, immediate action is required. First have a plan for who needs to be contacted. In some cases, you may have on-site computer security experts who will respond to the situation and escalate it as necessary. In other cases, you may be the in-house expert, and you may need to escalate the situation to your internal security department or even local, state, or federal law enforcement agencies.

In an extremely high-security environment, it may be appropriate to shut off access to the compromised services, including the directory itself and any dependent services such as authenticated intranet Web access. However, shutting off services abruptly will almost certainly tip off any attacker that his actions have been detected. If you can learn the network address from which the hacker is connecting, it may be possible to observe the actions closely and understand the extent of the damage. Gathering more evidence may help you catch the intruder and may prove useful if you decide to involve law enforcement agencies. Always keep a detailed log of evidence when you suspect the presence of an intruder. Compromised security is a serious problem and is covered in more detail in [Chapter 12, Privacy and Security Design](#).

Troubleshooting and Resolving Problems

When presented with a problem, you should follow a methodical, step-by-step approach to troubleshooting. The steps we suggest are covered in this section.

Step 1: Assess the Problem, and Inform Affected Persons

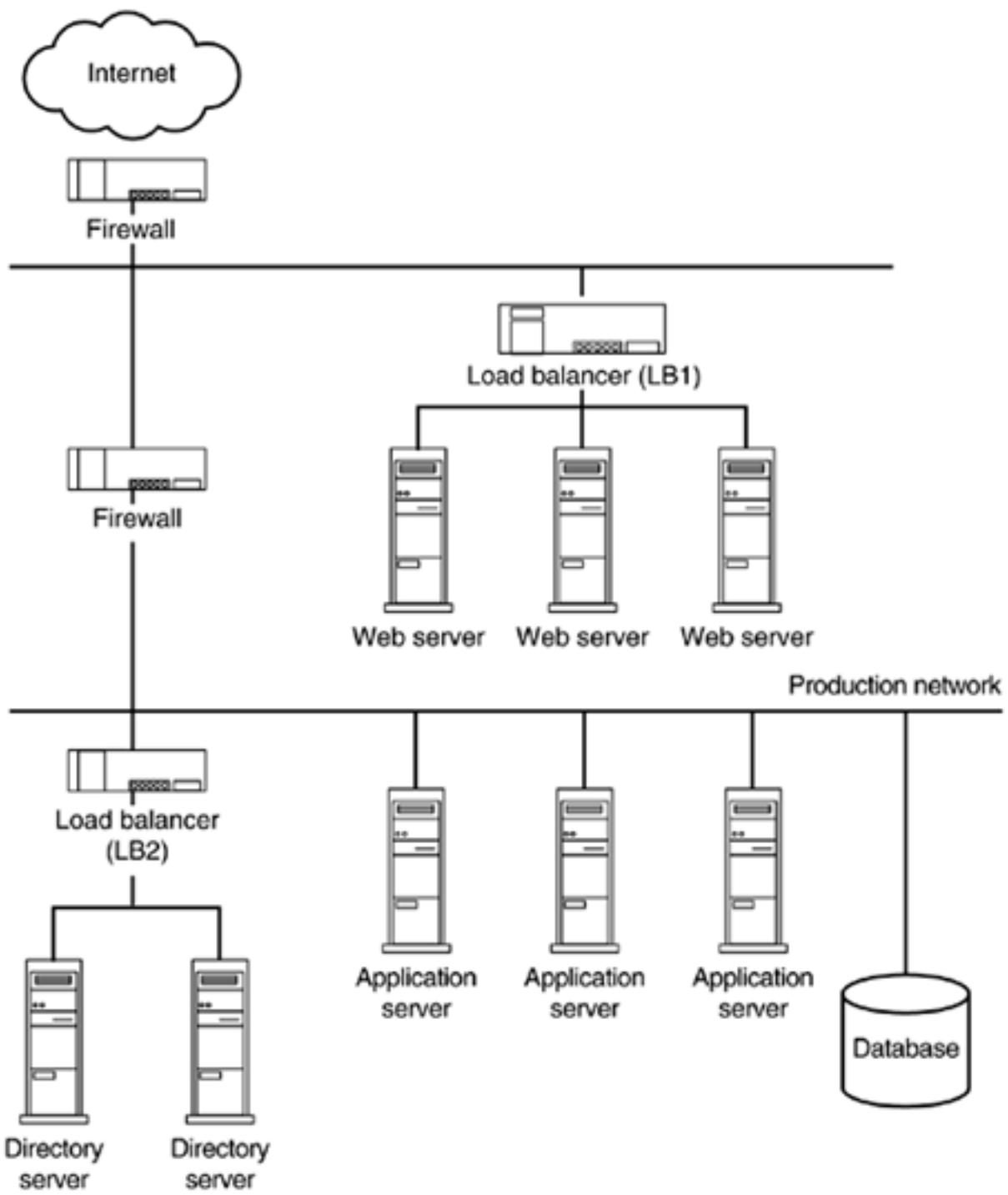
When you first become aware of a problem, perform a quick initial assessment of the evidence and try to understand the scope of the problem. At this stage the underlying cause of the problem may not be obvious. You may have only a few unrelated problem reports from your Help Desk that may or may not clearly point to the cause. When the underlying cause isn't clear, try to think of all the possible causes and investigate each one.

Depending on the size of your organization and how your computing support is provided, you may need to contact other people to understand the possible causes. For example, if you are in charge of running the LDAP servers, another group is responsible for the physical network, and yet another group is responsible for the network routing, you may need to get in touch with all these groups to understand whether the problem is the fault of a system you administer. If you work in this type of organization, it's valuable to cultivate good working relationships with the groups that provide services you depend on and the groups that use the services you provide. Having clear escalation processes and up-to-date contact information is also crucial to this type of distributed problem solving.

For example, if you receive notification that users cannot access Web pages that require authentication, you don't really know whether the problem is with the Web server itself, the directory, or an infrastructure piece such as the network between them. Is the Web server accessible? If you access it yourself, can you observe the same problem the end users encountered? Is the directory server running? Is there a problem with the directory server data? Make a list of possible causes.

In completing this initial assessment, a diagram of your application's components and the network devices they depend on is helpful. [Figure 20.1](#) shows a typical Web application.

Figure 20.1. A Typical Web Application



Suppose that users are able to reach your application's login page but receive an error message after providing their user name and password. Also assume that user authentication is implemented in your application layer by a Java 2 Enterprise Edition bean that runs on your application server and uses the Java Naming and Directory Interface (JNDI) to verify user credentials against the LDAP servers.

As a first step, you could verify whether your Web servers have connectivity to the application servers. If other pages on your site are implemented with support from the application layer, then try to view one of those pages. If they work properly, you can assume that your Web servers are able to talk to the application servers. Assume that you've completed this step and have verified that your application servers are up and are accessible from the Web tier.

When you're sure that the problem lies in the interaction between the application servers and the directory servers, look at the components involved in providing LDAP service to your application tier. Those components, and some troubleshooting techniques for diagnosing them, are as follows:

- The Java code that implements the authentication logic on the application servers. Application server logs can be helpful in determining the proper functioning of application-layer components.
- The directory servers themselves. Are the server processes running? Are they responding to LDAP requests? A cursory examination of the directory server logs may prove useful.
- The load balancer (LB2 on [Figure 20.1](#)) that provides failover for the directory servers. Is the load balancer functional? Is it properly relaying requests to the directory servers? Load balancers are complex devices and can easily be misconfigured.
- The physical network(s) that connect the application servers, load balancer LB2, and the directory servers. The ping utility is helpful in determining whether the network is at fault. If network performance is degraded because of faulty wiring, you may be able to observe errors on the network interfaces of your servers. The `netstat -i` command is useful for observing interface errors.

After you complete this initial assessment, inform the affected persons. Let them know that you are aware of the problem and are working to resolve it. If you have a good idea of how long it will take to solve the problem, provide an estimate (perhaps padding it a bit to give yourself time to proceed methodically). To the greatest extent possible, inform your users and Help Desk which services are affected by the problem and what the symptoms might be. For example, if the directory is inaccessible because of a network failure, what will users of the corporate address book application see? What will happen to incoming and outbound e-mail? Providing information to the Help Desk staff will help them understand the problem and communicate more effectively with the end users.

Step 2: Contain the Damage

After you've thought through the possible causes, can you identify any that might result in long-term data loss or corruption? For example, if the data in the directory has been damaged and entries for some employees are missing, what will happen to e-mail addressed to those people? Most e-mail server software will bounce the message (return to sender) with an explanation that no such address exists. Or if a disk drive appears to be failing and an automatic update script is about to run and put a heavy update load on the directory, what will happen to the data updates? Will they be lost?

If there is any possibility of such damage, it's often best to shut down the affected parts of the directory. Remember, delayed results or service unavailability are better than incorrect results or data loss.

Step 3: Put the System Back into Service by Applying a Short-Term Fix

After you've completed your initial assessment and contained any possible damage, it's time to start putting the directory back into service with a short-term fix, if appropriate.

For example, if your master directory server has a bad logic board and it will be several days before you can obtain a replacement part, perhaps an appropriate short-term fix is to place another machine in service. Or if you've determined that an automatic directory update process has erroneously removed entries from the directory, an appropriate short-term fix might be to restore the directory from a backup tape and shut off the automatic update process until you can analyze and resolve the problem with it.

The short-term fix often leaves you with somewhat reduced capacity. For example, you might put a replacement machine into service that isn't as fast as the machine it replaces. A

failed directory server might simply be taken out of service until it can be repaired, if you have a sufficient number of replicas to handle the remaining client load without serious degradation. This temporary solution is usually acceptable as long as the situation is eventually resolved and the full capacity of the directory is restored.

Step 4: Fully Understand the Problem, and Devise a Long-Term Fix

As soon as your directory is running with your short-term fix, it's time to examine all the evidence and fully comprehend what happened. For some types of problems, you will already understand the problem fully: Perhaps a power supply failed and has to be replaced, or a bug in update software needs to be fixed. In these cases a long-term fix isn't necessary.

In other cases, however, you might not fully understand what happened. Perhaps the directory server machine became unresponsive and had to be rebooted, or maybe replica directory servers got out of sync and had to be rebuilt from the master server. If you aren't 100 percent sure why the problem occurred in the first place, spend some time analyzing the failure. The following evidence can help with this step:

- Directory server usage and error log files
- Directory audit log files showing the changes made to the directory
- Log files generated by the operating system, including system logs (usually found in `/var/log` or `/var/adm` on Unix systems) or the Windows event log
- Log files from dependent applications such as Web servers, messaging servers, and so on
- Output from your NMS software
- Problem reports from end users
- Problem reports from your maintenance staff

Creating a timeline of these events will often help you understand the chronology and cause-effect relationships. For example, you may note that the directory server began reporting errors writing to its files at the same time the operating system began to log errors with the SCSI bus on the host. This correlation might lead you to suspect a problem with the cable attaching the disk drive to the server, other SCSI peripherals attached to the SCSI bus, or the disk itself.

Tip

Most servers log events chronologically and mark each event with a timestamp. Correlating logs across multiple servers is much easier if the server's clocks are all in sync. Use a time synchronization protocol such as the Network Time Protocol (NTP) to keep your server clocks synchronized.

If the underlying problem isn't something you can fix right away, you need to develop a long-term fix. The long-term fix might be straightforward, such as scheduling downtime to replace a failed power supply when the replacement part arrives; or it might be complicated

—for example, involving fixing a bug in an automatic update procedure.

Complicated long-term fixes should ideally be tested before deployment. Maintaining a lab environment in which you can test fixes before applying them to your production environment is a great way to address this need. For large or mission-critical directories, it's an absolute requirement.

Step 5: Implement the Long-Term Fix, and Take Steps to Prevent the Problem from Recurring

When your long-term fix has been identified and tested, it's time to deploy it. If the long-term fix requires any directory downtime, schedule it well in advance, ideally during hours of low directory use. For example, if you've put a replacement server in place while waiting for replacement parts for your primary server, you'll need some scheduled downtime to put the primary server in place.

In some cases you may be able to avoid downtime entirely. For example, suppose that your software supports multimaster replication and you need to exchange a server for its replacement. You could put the repaired server into place, make it a read/write replica, and then remove the temporary server from service. A transition carried out in this manner is entirely transparent to end users.

When you have everything up and running, ask yourself whether there is a way either to prevent the problem from happening again or to mitigate the negative consequences. For example, if an outage was caused by the failure of a server, could the impact have been lessened if more replicas of the data had been available? If the outage was caused by a bug in the server software or operating system, have appropriate patches been deployed to all servers? Think of each incident as providing you with valuable input for improving the quality of your directory service.

Tip

One way to help ensure that changes to your directory environment are as nondisruptive as possible is to implement a *change control policy*. Such a policy describes the lead time required before a change can be implemented and who must be notified. An example of a change control policy might include the following:

- At least 24 hours before any change happens, a change control notice must be submitted to all relevant persons, including administrators of dependent systems. This notice should describe the change being made, the reason for the change, and the anticipated effects on dependent services.
- During this 24-hour period, the change may be vetoed if one of the affected administrators believes it will conflict with a previously scheduled change. If vetoed, the change must be rescheduled for a later time.
- Emergency changes must, of course, be permitted.

Such a policy will help ensure that all affected staff are prepared for the maintenance or outage and know whom to contact if the outage causes problems with dependent systems.

Step 6: Arrange to Monitor for the Problem

If your monitoring system did not detect the problem, can your monitoring strategy be updated? If you are able to update your monitoring system to catch the problem in the future, you should be able to improve the response time. If the problem is the result of a bug in operating system or directory server software but a patch is not yet available, monitoring for the condition and taking an action such as rebooting a server can improve the reliability of the directory until a patch is available and can be installed.

Step 7: Document What Happened

Finally, take the time to produce a report of the problem, the steps you followed to determine the cause, and the resolution. A workflow diagram describing these steps may be useful in the future, especially if you need to create a decision tree to aid in troubleshooting. In addition, record the total time the service was interrupted. A collection of these reports can be a valuable resource for new technical staff. These reports can also be useful when you're communicating with management, especially if you want to make the case for additional funding for things such as increased server capacity or improved monitoring support.

Troubleshooting Checklist

The following are checklists of items to note and information to gather when you have a problem with your directory.

Directory Outages

- Are directory clients timing out, or are their connections being refused by the server?
- Are all network components (routers, hubs, switches, cables) between the clients and the servers functional?
- Is the directory server machine running? If not, has the hardware failed?
- Are all hardware components on the directory server machine functioning properly? Have any operating system logs recorded hardware failures?
- Is the directory server process running? If so, is it consuming any CPU time? Is it consuming too much CPU time? Is it causing any disk activity?
- If the directory server process is not running, did it fail when processing a particular client request? Does it fail each time it receives such a request? Repeated failures of the same type of request might represent a denial-of-service attack, or it might simply be a bug.

Performance Problems

- Are specific types of directory operations performing poorly, or is the overall performance of the server poor?
- Are appropriate attribute indexes being maintained on the directory server?
- Is the directory server process too large? If so, does it become too large immediately on startup or gradually over time?
- Are the cache sizes of the directory (if any) configured appropriately (neither too small nor too large)?
- Are other processes running on the directory server machine causing the poor performance?
- Is the directory under a particularly heavy load? Is this load expected? If it is excessive, is one particular client or application accounting for most of the load?
- Is the operating system correctly tuned, and are all appropriate patches installed? The dsktune utility provided with Netscape Directory Server can help you answer these questions.

Problems with Directory Data

- Is data missing or incorrect?
- Does the data appear to be corrupted in a catastrophic manner? Such damage indicates a serious hardware or software problem.
- Is the data damaged in a specific way? For example, have certain entries been erroneously deleted? Can the source of the erroneous modifications be determined by examination of directory server logs?

Security Problems

- Are there telltale signs of a break-in, such as connections from an unexpected location or unexpected modifications to directory entries?
- Do directory logs show unexpected client activity?
- Is the directory experiencing a denial-of-service attack? (Such an attack usually overwhelms available server resources.) If so, is the source of the attack known?

Further Reading

Netscape Directory Server 6 Administrator's Guide. Available on the World Wide Web at <http://enterprise.netscape.com/docs/directory>.

Network Time Protocol (Version 3) Specification, Implementation and Analysis (RFC 1305). D. Mills, 1992. Available on the World Wide Web at <http://www.ietf.org/rfc/rfc1305.txt>.

Looking Ahead

This chapter concludes [Part IV](#), Maintaining Your Directory Service. Now that we've discussed planning, deploying, and maintaining your directory service, in [Part V](#), Leveraging Your Directory Service, we will turn our attention to the topic of leveraging your directory service and using it in new ways to lower costs and improve the effectiveness of your computing applications.

Part V: Leveraging Your Directory Service

[21. Developing New Applications](#)

[22. Directory-Enabling Existing Applications](#)

[23. Directory Coexistence](#)

Chapter 21. Developing New Applications

- Reasons to Develop Directory-Enabled Applications
- Common Ways That Applications Use Directories
- Tools for Developing LDAP Applications
- Advice for LDAP Application Developers
- Example 1: setpwd, a Password-Resetting Utility
- Example 2: SimpleSite, a Web Site with User Profile Storage
- Developing New Applications Checklist
- Further Reading
- Looking Ahead

In the next few chapters we describe how to leverage your deployed directory service and get more out of your directory investment. One of the most common ways to leverage a directory is to create new applications that use it. As discussed in [Chapter 6](#), Defining Your Directory Needs, serving the needs of one or more directory-enabled applications is often the driving force behind the initial deployment of a directory service. The first applications deployed against a directory service are typically off-the-shelf commercial or open-source applications. In this chapter we take the next step and look at creating new directory-enabled applications.

From an application developer's point of view, an LDAP directory service is a wonderful thing. LDAP provides a simple, secure, and standard way to access a shared, flexible store of information. Several high-quality software development kits (SDKs) for writing LDAP applications are available at minimal or no cost. These SDKs free developers from worrying about low-level protocol details and allow them to focus on what they're trying to accomplish with their applications.

In this chapter we first discuss why it makes sense to develop directory applications. Next, we describe the most common ways that applications use directories. We then provide an overview of the available tools for developing LDAP applications. We also include a section that contains helpful advice for writing an LDAP directory-enabled application, and we conclude with two sample applications. This chapter will expose you to some of the many ways that applications can use LDAP, help you generate ideas for new applications that will add value to your own directory service, and show you that developing applications using LDAP is fun, easy, and rewarding. So don't be afraid to get your hands dirty. Write some code!

Reasons to Develop Directory-Enabled Applications

Creating LDAP directory-enabled applications enables you to

- **Lower your data management costs.** By using a shared directory to avoid use of application-specific data stores, your organization saves money that would otherwise be spent maintaining redundant data.
- **Adapt the directory to fit your organization.** One example of such adaptation is the creation of custom tools to streamline various directory management and application management tasks.
- **Save on application development, deployment, and maintenance costs.** LDAP applications are easy to develop, and compared to a relational database management system (RDBMS) data store, LDAP directories are often cheaper to deploy and maintain. LDAP applications are also very portable: If written correctly, they will work with a variety of standards-compliant directory servers. In addition, the value of many kinds of applications is enhanced if they can easily access the timely, accurate data that is often stored in a directory service.
- **Create entirely new kinds of applications.** By leveraging the directory infrastructure, you can relatively easily create applications that would normally be difficult to develop.

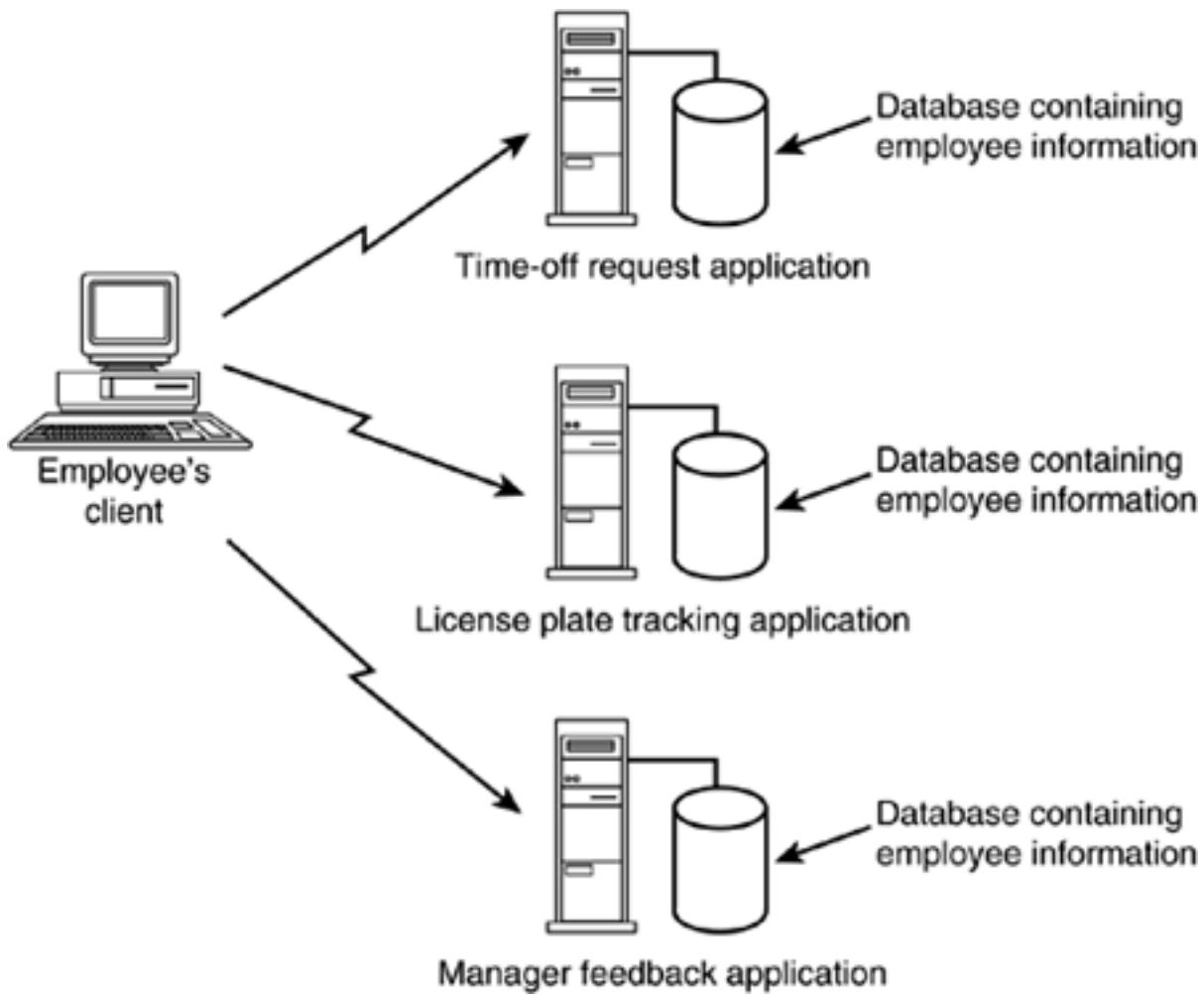
Each of these benefits is discussed further in the following sections. After that, we provide a few reasons why it may *not* make sense to directory-enable an application.

Lowering Your Data Management Costs

If the application being developed needs access to data already stored in your directory, or if the data it uses must be shared with other applications, using a private data store is an expensive mistake. By leveraging your deployed directory service, you avoid creating a new data store that must be designed, deployed, and maintained. By using your directory service as a rendezvous point for applications that share data elements, you avoid the need for troublesome data synchronization between data stores.

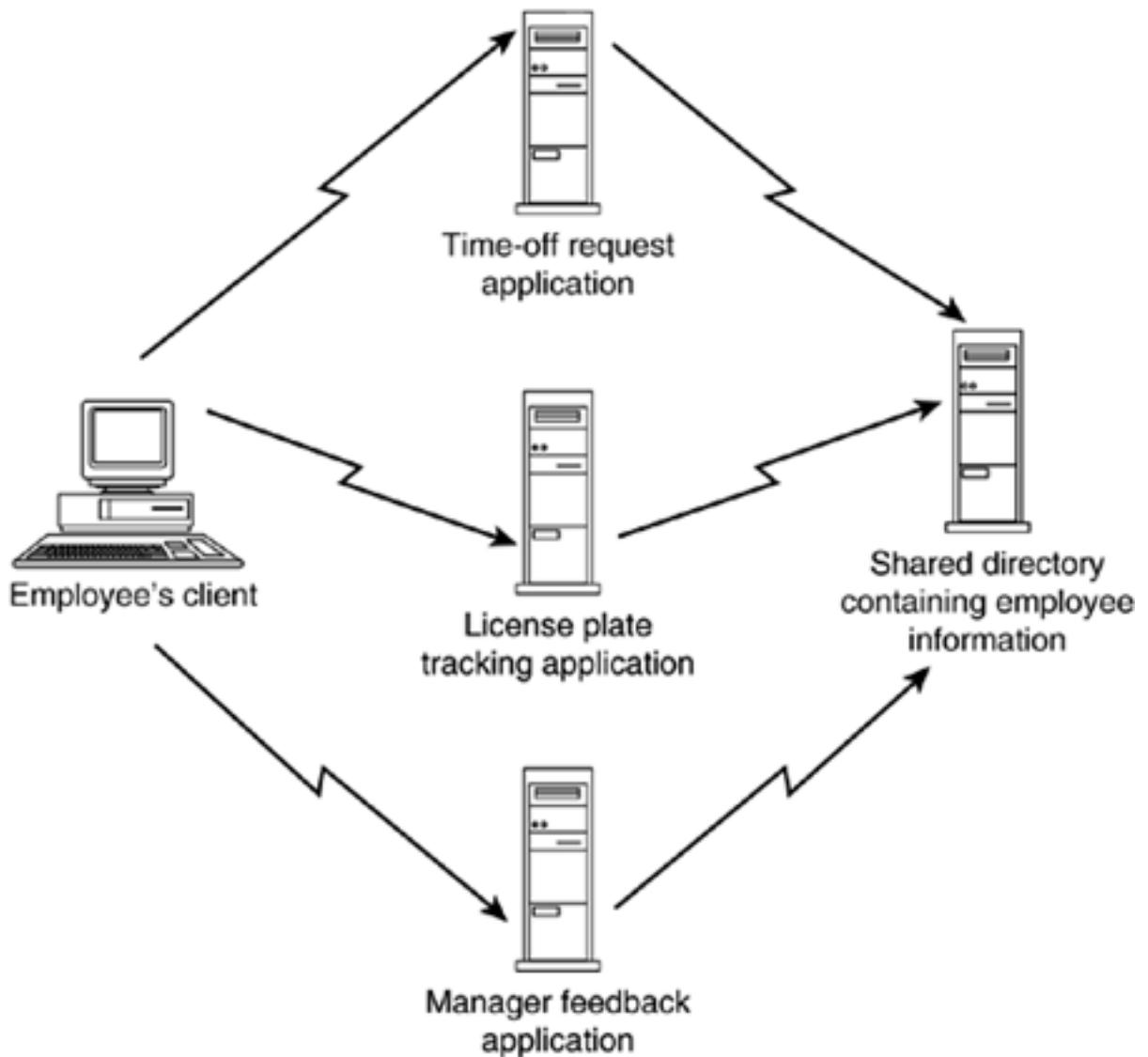
Suppose you plan to develop a set of Web-based applications to automate common employee tasks such as requesting time off, registering car license plate numbers for use by the security office, and providing feedback about manager performance. Without a central directory service, each application might have its own database of information about employees, including user passwords and the names of their managers. [Figure 21.1](#) illustrates this unhappy situation.

Figure 21.1. Applications with Redundant Data Stores



A better approach is to design each application to use your directory service. The time-off request and manager feedback applications both need access to the manager's name, and all the applications need to authenticate employees with user IDs and passwords. Storing all the shared information in your directory service eliminates redundant data elements and extra databases. [Figure 21.2](#) shows the revised approach.

Figure 21.2. Applications That Share a Directory Service



The picture is much simpler now, and in this case a simpler architecture translates directly into reduced data maintenance costs.

Adapting the Directory to Fit Your Organization

It is rare to find a perfect fit between the way different people want to work and available off-the-shelf directory tools. You'll most likely want to develop new LDAP applications yourself to help streamline maintenance and everyday use of your directory service.

For example, a common task for a Help Desk is resetting user passwords. Help Desk employees can accomplish this simple task by using any of the user and group management interfaces that directory vendors ship with their products. However, you may want to develop a more focused tool that fits your preferred way of handling password resets. For example, your new tool might generate a new random password to increase security (by avoiding repeated use of the same password by Help Desk employees).

Such a custom password-resetting tool might work as shown in [Listing 21.1](#). Here the Help Desk employee identified as `kvaughan` resets the password for the user identified as `bjensen`. Source code for the `setpwd` utility is provided later in this chapter. Custom applications of this kind range from small, focused utilities such as `setpwd` to large, complex applications that perform a variety of management tasks.

Listing 21.1 Using a Custom Password-Resetting Application

```
setpwd bjensen
```

```
Password for kvaughan: secret
```

```
Contacting LDAP server...
```

```
Finding bjensen's entry...
```

```
Modifying bjensen's entry...
```

```
Password reset. bjensen's new password is: scroll.insofar7.
```

```
Don't forget to remind bjensen to reset his/her password right away!
```

Saving on Deployment and Maintenance Costs

After you have deployed a directory service, the incremental cost of supporting additional applications is quite low. There is no need to discard what you have already done and start over. Most of the directory design, maintenance, and support infrastructure already in place can be expanded incrementally to accommodate new applications. For applications that require a data store of some kind, you must often choose between using your directory service and using a traditional RDBMS such as an Oracle or IBM DB2 system. Directory servers are typically less expensive to purchase, deploy, and maintain than RDBMS servers. Through the use of replication, directories can scale up to provide high-performance access for many applications. Directories are flexible and therefore easy to change to support the needs of new applications.

For example, adding new schema elements (columns) to an RDBMS table usually requires that the existing database be shut down and rebuilt with a special utility. This is a tedious, time-consuming process that may disrupt database service for existing applications. In contrast, new schema elements (object classes and attribute types) can be added to most LDAP directories at any time; there is no need to shut down the service.

Another important factor is that often a lot of effort is invested in deploying a directory service that contains timely, accurate data. Applications that need access to that kind of data will be simpler to create and will work better if they use such a directory service. For example, many companies require that their critical customer support employees carry pagers, and they use a scheme in which different employees are *on call* (eligible to be paged) during different portions of the day. If a directory service contains accurate information about which employees are on call, then all software that automatically pages employees should access the directory to determine where to direct a page. This sounds simple and obvious—and it is—but many systems have been deployed in which each software application that sends pages has its own database of pager numbers and other information that must be maintained independently or synchronized with a directory service.

Creating Entirely New Kinds of Applications

An open, standards-based directory provides a great rendezvous point for distributed applications. It also frees information that was previously locked away inside proprietary data stores for use by innovative application developers. You may discover that some kinds of applications that were impossible or very difficult to create without a central directory service are now easy to create.

In many organizations, for example, a small army of departmental administrative assistants

is responsible for creating and publishing organization charts. New charts are typically created through a manual process and then published inside the company about once a month and distributed in paper form. By leveraging a directory service that contains basic information about employee relationships, you can develop an application that creates organization charts on demand. Advantages of such an application include reduced costs for maintenance of the data, elimination of distribution costs, and access to more up-to-date organization charts. [Figure 25.10](#), which is part of [Chapter 25](#), Case Study: A Large Multinational Enterprise, shows a prototype of a directory-based organizational chart application.

The convergence of LDAP, HTTP, HTML, XML, and other Internet standards means that it's easier than ever before to build applications that pull together information and make it available to wide audiences at low cost.

When It Does Not Make Sense to Directory-Enable

It doesn't make sense to integrate every new application with your LDAP directory service. In [Chapter 1](#), Directory Services Overview and History, you learned what a directory service is and is not good for. Generally, the kind of applications that would not benefit from being directory-enabled are those that use only private, application-specific data that does not need to be shared with other applications.

Tip

In a few cases there may be some benefit to storing private data in a centralized or private directory service—that is, if doing so reduces the pain of managing the data, allows new end-user features to be provided, or is more convenient for other reasons. For example, by storing data in a directory service instead of on a local PC, applications can provide location independence for users (also called *roaming*). Storing application-specific data in a shared, network-accessible directory instead of in a local file system allows an application to retrieve the information regardless of where on the network it executes. And it may occasionally be advantageous to use a private, application-specific directory service based on LDAP because of LDAP's rich data model and the availability of good tools, SDKs, and servers.

Another guideline is that applications whose data-related requirements violate your data policy should not be directory-enabled (see [Chapter 7](#), Data Design, for information on creating a data policy). Unless you are willing to change your policy, it makes no sense to go through the trouble of writing the application so that it can use your directory service. For example, suppose your data policy stipulates that no private data such as a person's date of birth, photograph, or U.S. Social Security number should be stored in your directory. In that case, you shouldn't bother directory-enabling any applications that need to store Social Security numbers. That data should instead be stored in secure, private, application-specific databases.

Common Ways That Applications Use Directories

Because directory services are so flexible, applications use them in many different ways. Some of the most common tasks for which applications use directories include the following:

- Locating and sharing information
- Verifying authentication credentials
- Aiding the deployment of other services
- Making access control decisions
- Enabling location independence

These examples are discussed in the following sections.

Locating and Sharing Information

The most common use of a directory service is to locate and share information among end users and networked applications. Information held in directories is typically published and maintained by a small set of people and applications, but it is accessed by many. Storing information in a shared directory service has several advantages:

- The information can be maintained over the network with LDAP, and management tasks can be distributed to many people.
- Multiple instances of the same application, as well as instances of unrelated applications, can access the data no matter where they are on the network.
- Access control provided by the directory service can be used to ensure that each person and each application has only the necessary and appropriate level of access to information.
- Information about people and resources can be consolidated into one logical location and managed in a consistent way through the use of directory-enabled management interfaces.

Most of the applications mentioned in this book do store some shared information in a directory. Whether the information is shared with just one management tool or with hundreds of other applications, placing it in a directory service makes it more widely available and easier to manage.

Verifying Authentication Credentials

A simple but powerful use of a directory service is to provide authentication. The basic idea is for the application to ask the directory to verify authentication credentials presented by users or by other applications. This can usually be done through an LDAP bind request, in which the success or failure of the bind translates directly into success or failure of the authentication attempt within the application.

[Figure 21.3](#) shows how a Web server might use a directory server to authenticate users before allowing access to a set of documents. The process goes like this:

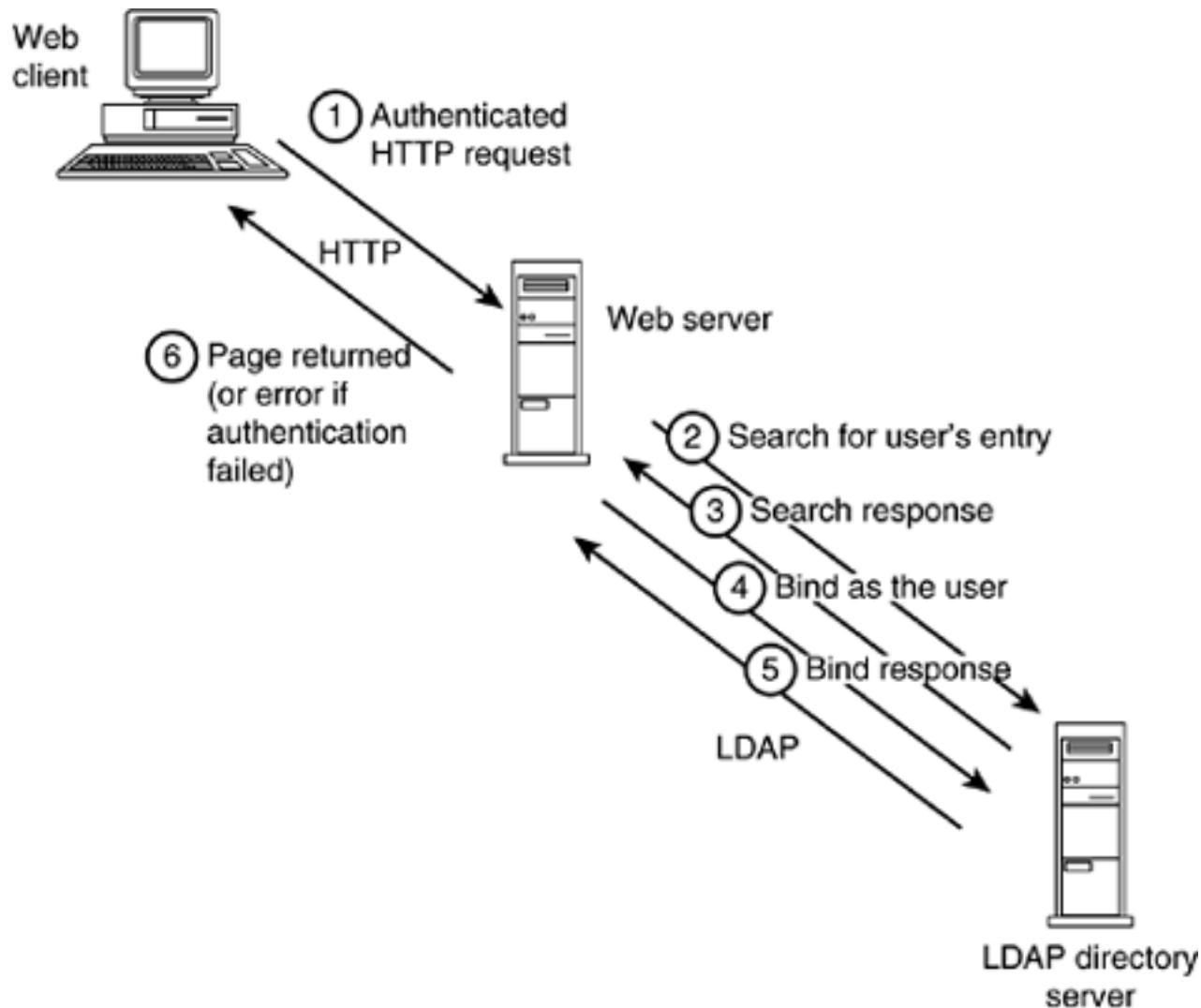
- Step 1.** The Web server obtains a user ID and password through an authenticated HTTP request sent by a Web browser or a Web services application.
- Step 2.** The Web server sends an LDAP search request to find the user.
- Step 3.** The server responds to the search request.

Step 4. If the search is successful, the Web server sends an LDAP bind request to check the user's password.

Step 5. The server responds to the bind request.

Step 6. The Web server uses the result of the bind operation to decide whether to return the requested Web page or an error.

Figure 21.3. A Web Server That Uses a Directory for Authentication



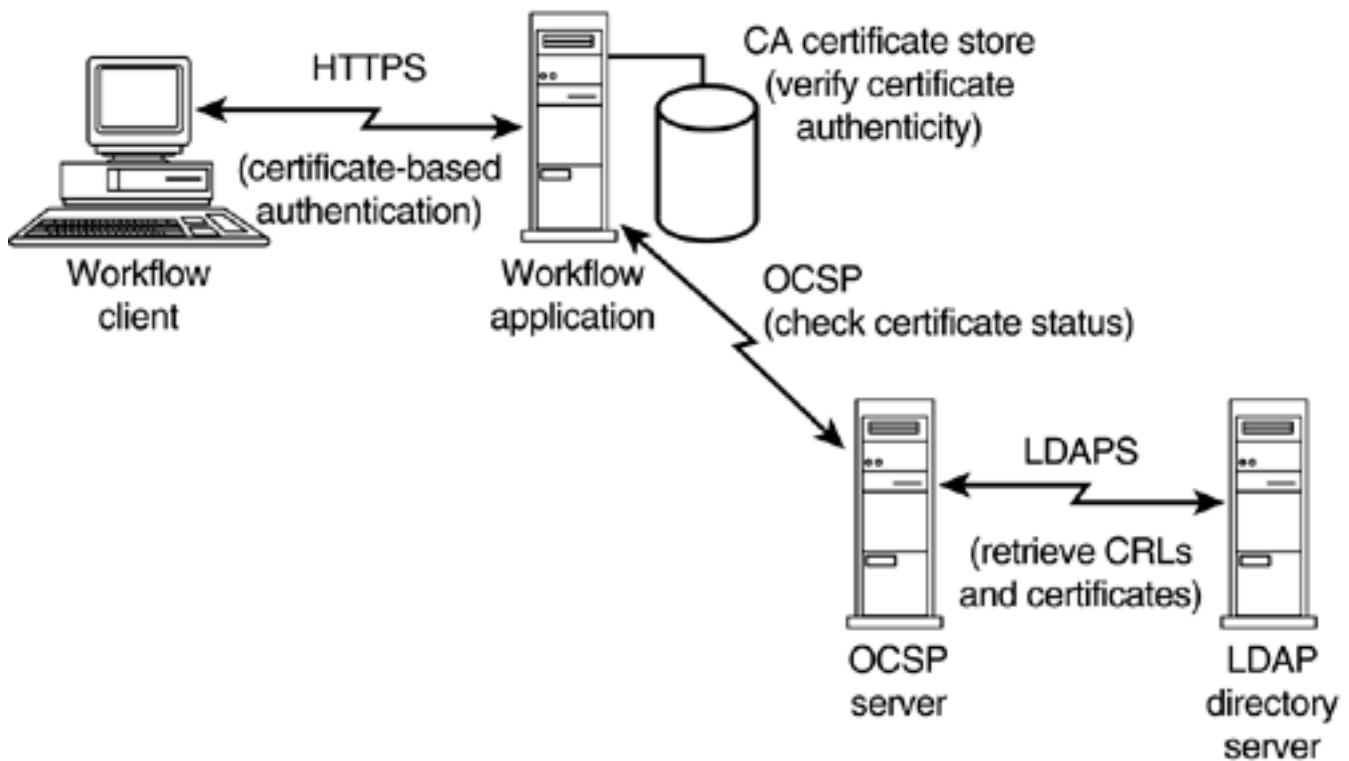
Tip

Sometimes it is inconvenient or less secure to ask a directory server to verify a user's authentication credentials on behalf of the application. In this case it may still make sense to *store* in the directory service the credentials or other information required to verify each user's identity. That way, the authentication credentials can be shared among all of the applications that need them even if the directory itself is not being used to check the authentication credentials.

Using a directory as a shared authentication service allows end users to access all application services with just one set of authentication credentials. Typically this means that each end user needs to remember only one password or carry one public key certificate—which makes users happier and leads to fewer calls to the Help Desk.

As an example of using a directory service to store authentication credentials, consider a public key certificate-based authentication system that needs to do revocation checking. Usually an application can itself verify that a certificate presented by an end user is valid and not expired, but it is more difficult for an application to know which certificates have been revoked. Certificates are typically revoked after a person's public key has been compromised or when a person leaves an organization. In traditional public key infrastructure (PKI) designs, certificate revocation lists (CRLs) must be distributed to every application. If certificates or CRLs are stored in a directory service, revocation can be checked by online revocation checks done over LDAP itself, or indirectly through a protocol designed for revocation checking, such as the Online Certificate Status Protocol (OCSP). [Figure 21.4](#) shows one possible configuration.

Figure 21.4. Using a Directory to Check for Certification Revocation



The workflow application shown in [Figure 21.4](#) uses public key certificates for authentication, and it verifies the authenticity of the certificate using locally stored information about the certificate authority (CA) that signed the certificate. An OCSP server that has an LDAP directory server on the back end is consulted to ensure that the presented certificate has not been revoked. The OCSP server's job is to tell applications whether a certificate is still active and whether it should be honored.

Aiding the Deployment of Other Services

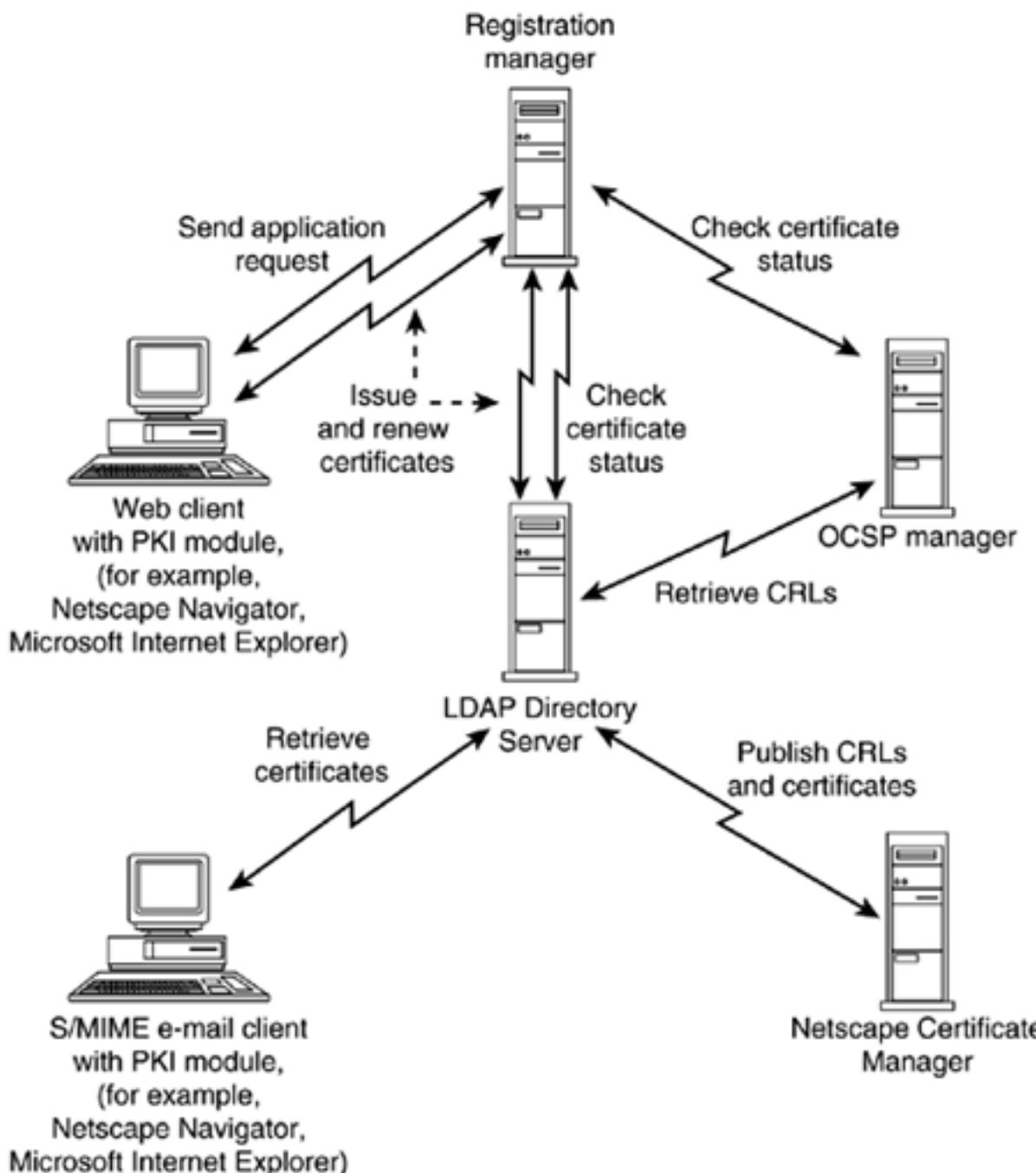
Because most directories provide access to data over a standard protocol (LDAP), directories are helpful in tying together a set of cooperating applications. This property of directories is especially important if you work with computing services that have historically proven to be costly to deploy and maintain. By using a directory service to facilitate self-service by end users, as a rendezvous point for a set of cooperating applications, and as a central point for service management, you can reduce the cost of providing a service.

One infamous example of a technology that has proven difficult and costly to deploy is PKI, which we already discussed briefly in the previous section, Verifying Authentication

Credentials. With PKI, you can increase security and improve efficiency by issuing public key certificates to end users and applications. The process of issuing, revoking, and renewing certificates is commonly referred to as *certificate life cycle management*—the major headache encountered in most real-world PKI deployments.

By using a directory service to assist in the certificate life cycle management process, products such as the Netscape Certificate Management System (CMS) are making PKI much more palatable. [Figure 21.5](#) shows the main components of a complete PKI solution based on a directory.

Figure 21.5. Using a Directory Service to Facilitate PKI Deployment



You can create certificates containing users' public keys by contacting the registration manager. The certificate manager component publishes user certificates by adding them to each person's directory entry so that they can be used by applications such as S/MIME e-mail. CRLs are also published in the directory server, where they are used directly by applications or through a set of OCSP Manager responders. Finally, when a Web browser is launched, the PKI module within it checks a special attribute in the user's directory entry to

see if any PKI-related actions should be taken. The last technique is used to support automated renewal of certificates that are about to expire, and to initiate any other PKI-related process that requires the user to do something.

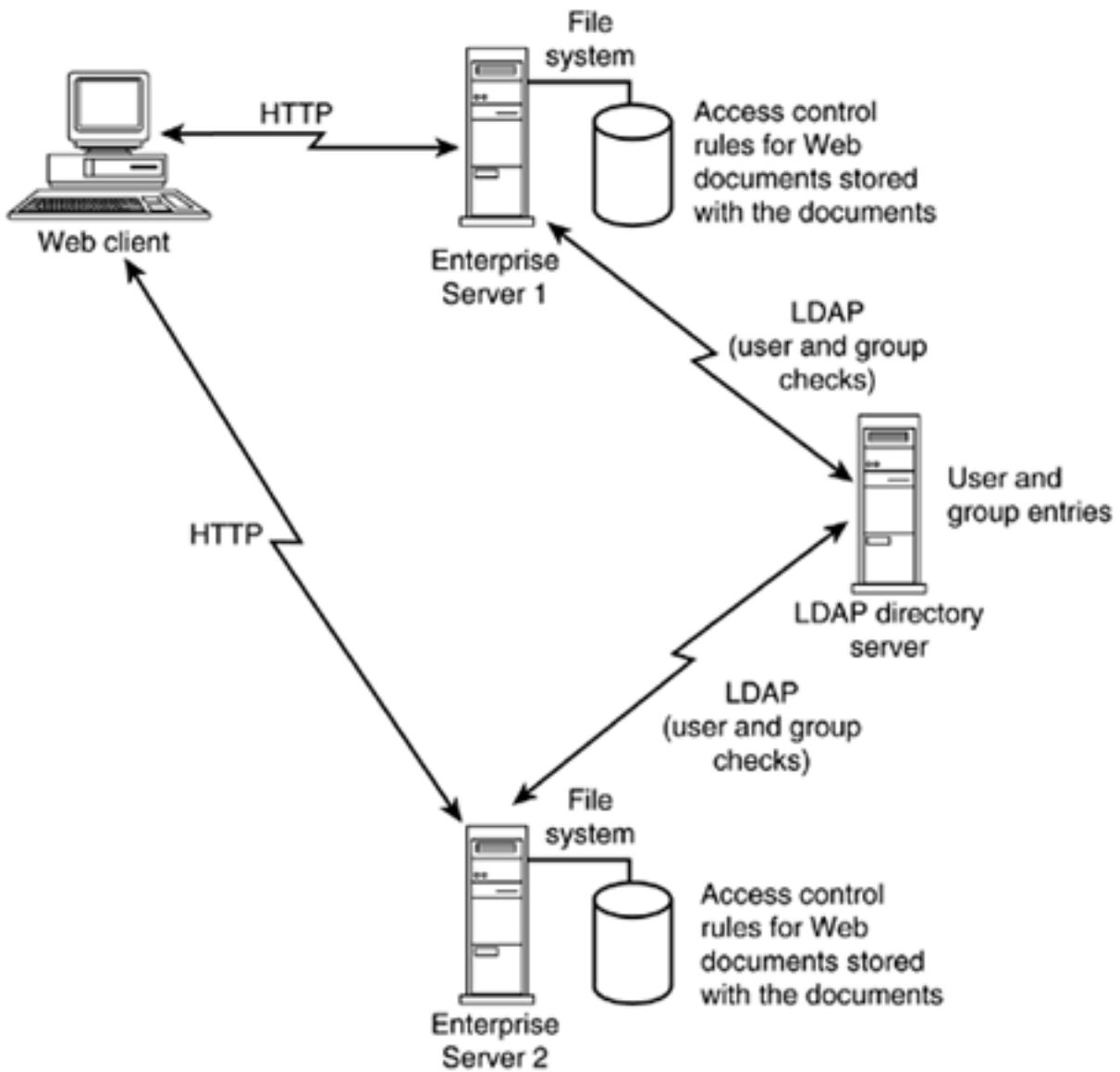
Making Access Control Decisions

Another way that applications can take advantage of a directory is to help make decisions about who is allowed to do what. This decision-making process is usually referred to as *authorization* or *access control*. Directory services offer several advantages in assisting with access control decisions:

- Administration of the access control information can be distributed among several people, and a new protocol does not need to be invented for maintenance. LDAP itself does the job nicely.
- The information on which access control is based can be shared by several applications, so redundant access control information is eliminated, thereby reducing management costs and lowering security risks.
- When appropriate, access control information can be used for related purposes as well. For example, a group can be used both to perform access control checks and to distribute electronic mail.
- To meet the performance and reliability needs of applications, the access control information can be made widely available through directory replication.

Applications use directories in several ways to help make access control decisions. One approach is for the application to store a set of access control rules locally that refer to directory information (such as user and group entries). For example, a Web server such as Netscape Enterprise Server 6 stores access control rules in a local file system with the documents it serves up. However, Enterprise Server's access control rules can refer to user and group entries that are stored in an LDAP server. [Figure 21.6](#) shows a installation of Enterprise Server that is configured to use LDAP to assist with access control.

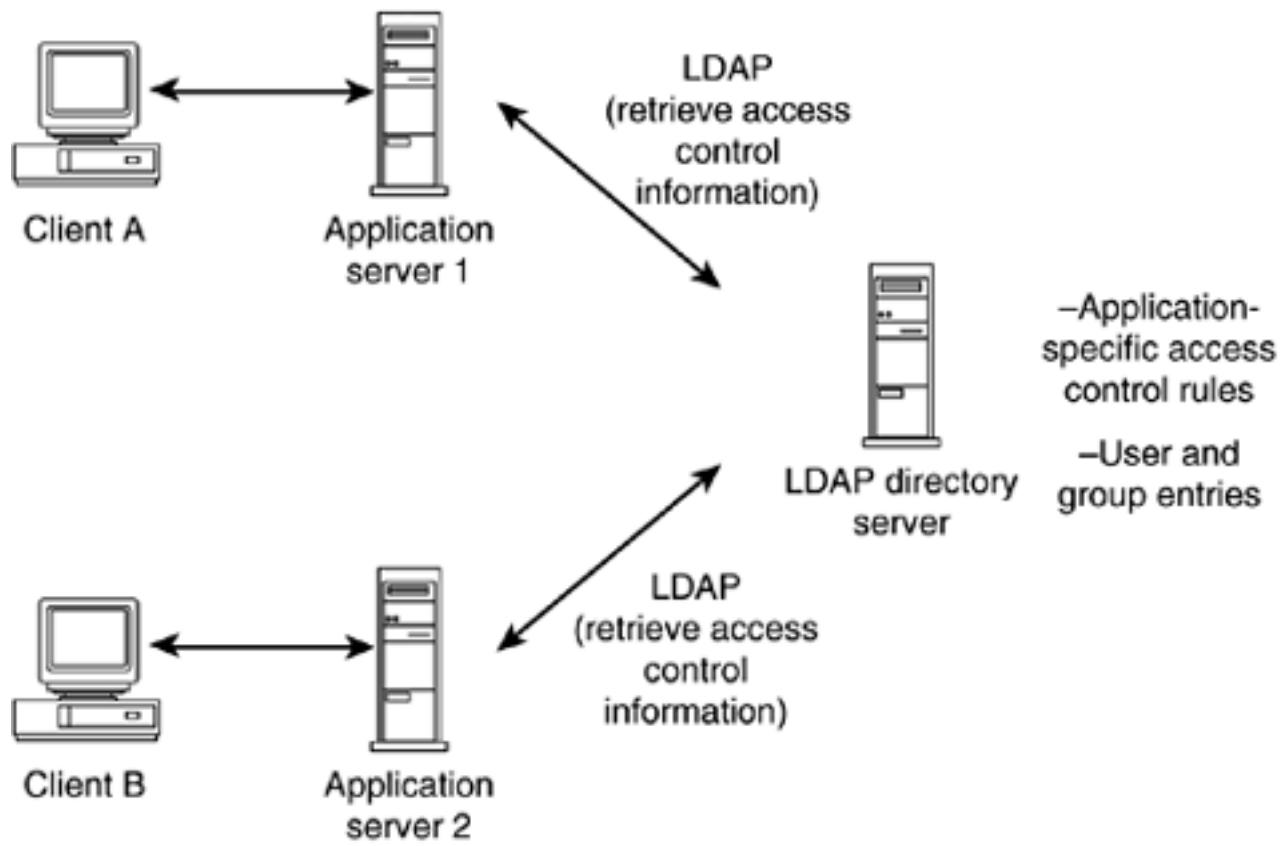
Figure 21.6. How Netscape Enterprise Server Uses LDAP for Access Control



Access control rules are evaluated locally by the Web server, and information about users and groups is retrieved from the directory as needed. Some advantages of this approach are that the application is free to implement an access control scheme that perfectly fits its needs, and the access control information itself can be stored close to the objects to which access control is applied.

A second approach is for the application to store its own application-specific access control information in the directory service. The information is read from the directory and evaluated by the application. [Figure 21.7](#) shows an application that uses this approach.

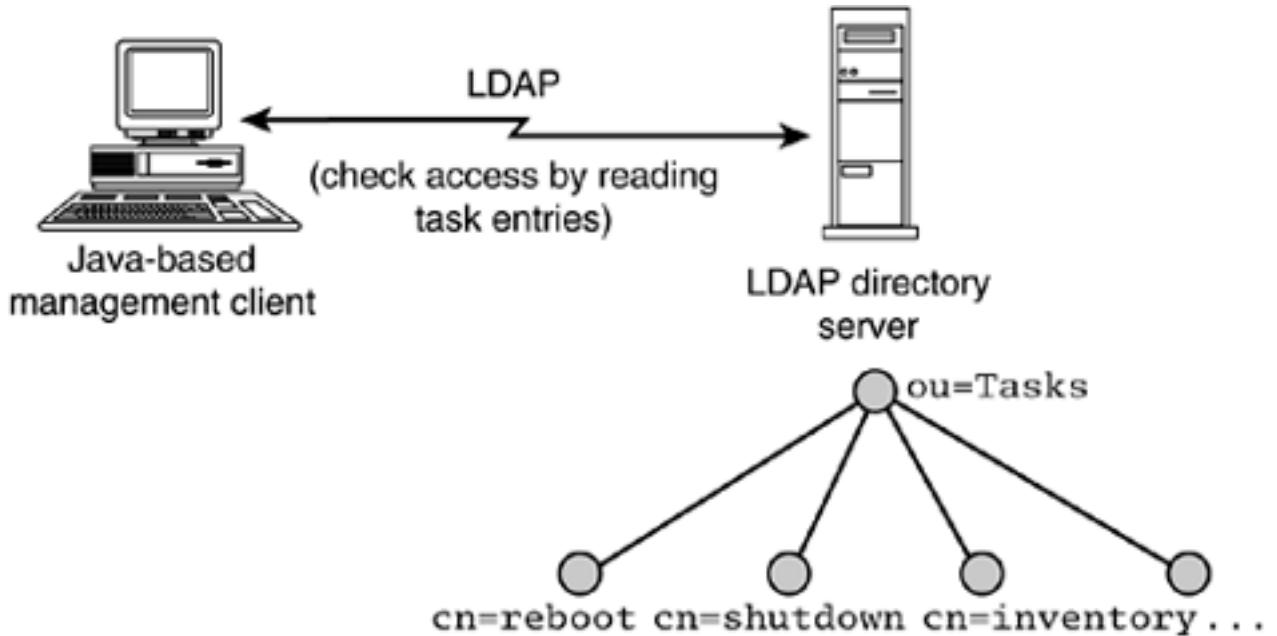
Figure 21.7. An Application That Stores Access Control Rules in an LDAP Directory Service



Two instances of the same application are running on two different application servers. The access control information for the application, as well as the user and group information, is obtained from the LDAP directory server. Because the access control information is stored in a shared directory that can be replicated, the information can be centrally managed and shared among all instances of the application (or with related applications).

A third approach for using a directory service to support access control is for the application to take advantage of the access control evaluation features of the directory itself. In this case the objects or tasks for which access control needs to be enforced are represented in the directory service. Appropriate directory access control rules are created to govern access to these entries. After authenticating as the entity that is requesting access, the application can use the success or failure of an LDAP operation to grant or deny access. [Figure 21.8](#) shows an example of this approach. The computer system management application shown allows the user to perform a task only if the user is granted read access to a corresponding task entry stored in the directory service.

Figure 21.8. Controlling Access to a Set of Tasks



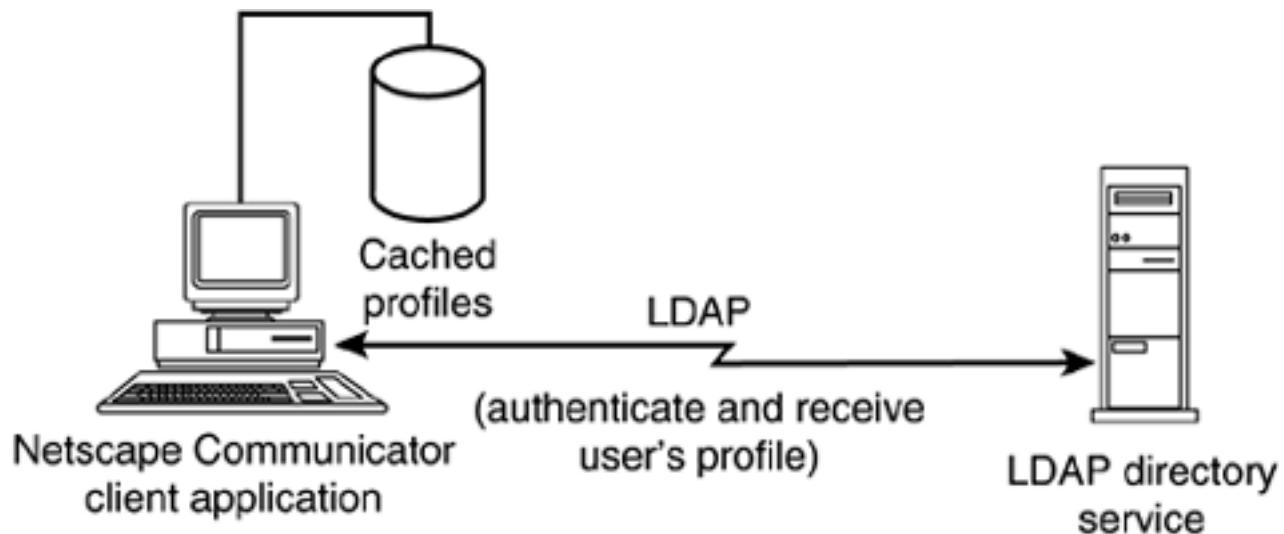
Netscape Console, which is used for managing many of Netscape's server products, uses this approach. One advantage of leveraging the directory service's access control evaluation machinery in this way is that the application doesn't need to implement an access control evaluation engine, design a syntax for access control rules, or provide specialized tools for creating and editing the rules; directory server software vendors typically provide all these things. As long as the access control features provided by the directory software meet the needs of your application, this approach can save a lot of work. There is one disadvantage to this approach, however: Until access control is made part of the LDAP standard, writing an application that uses directory access control may tie the application to a specific LDAP server implementation.

Enabling Location Independence

An increasingly popular reason for software developers to use LDAP is to enable location independence features in their applications. Location independence, or *roaming*, is achieved when end users and network applications can access the information they need (such as application preferences) regardless of the machine they're running on and where they're connected to the network. As long as the directory can be accessed securely from anywhere on your organization's network or on the Internet (for an outward-facing directory application), your directory service is a good tool for implementing location independence. Instead of storing configuration and preference information in local files or the Windows Registry, your applications can store them in an LDAP directory.

Location independence is a powerful feature that promises to unchain end users from their desktops and allow effective roaming within the network. A good example of an application that demonstrates these benefits is the Netscape Communicator Web browser and e-mail suite. Like traditional applications, Communicator can store end-user preferences (called *user profiles*) in a local file system. However, as shown in [Figure 21.9](#), Communicator can also be configured to retrieve user profile information from an LDAP server.

Figure 21.9. Using a Directory Service to Enable Location Independence



When configured in this way, the Communicator application asks the end user for a name and password at startup. It uses this information to authenticate with a directory server, and then it downloads the user's profile information to the local disk, where it is cached. If the user makes any changes to his or her preferences while using Communicator, the changed information is written back to the directory service. For example, if the user adds or deletes a bookmark, the new bookmark data is recorded in the directory.

Tools for Developing LDAP Applications

In this section we examine some different categories of SDKs and scripting tools that can be used to develop LDAP applications. The Further Reading section near the end of this chapter includes pointers to more detailed information on all the tools we mention. Directory software vendors naturally encourage developers to write directory-enabled applications, so many of these tools can be freely downloaded from the Internet and used at no cost.

LDAP software development tools that can be used to write directory-enabled applications can be organized into four categories:

1. LDAP SDKs
2. LDAP command-line tools
3. LDAP tag libraries for Web development
4. Directory-agnostic SDKs

Practically speaking, your choice of LDAP software development tools is constrained by the programming languages you're comfortable using. The next two most important criteria for choosing a tool are its quality and its documentation. You should experiment with several packages to see which one you like best before you begin writing code.

LDAP SDKs

LDAP SDKs typically provide complete, high-performance, native access to all the features of LDAP. The application programming interfaces (APIs) they provide may conform to the C or Java LDAP API specifications being developed by the Internet Engineering Task Force (IETF). At the time of this writing, the programming languages directly supported include C, C++, Perl, Java, Python, and Ruby. Popular LDAP SDKs include the following:

- **Netscape LDAP C SDK** (also available in source code form through mozilla.org). This SDK closely tracks the emerging IETF standard for an LDAP C API.
- **Netscape LDAP Java SDK** (also available in source code form through mozilla.org). This SDK closely tracks the emerging IETF standard for an LDAP Java API.
- **Net::LDAP Perl-LDAP modules** by Graham Barr and contributors. This is a native Perl implementation of LDAP; no platform-specific code is used.
- **PerLDAP Perl module**. PerLDAP calls through to the Netscape LDAP C SDK.
- **Python-LDAP**, which provides LDAP access from the Python language. Python-LDAP calls through to an LDAP C SDK.
- **Ruby/LDAP**, which provides LDAP access from the Ruby language. Ruby/LDAP calls through to an LDAP C SDK.

An LDAP SDK is a good choice if the application you need to LDAP-enable is written in one of the supported languages. All popular hardware and OS platforms are supported, and in some cases (such as with the Netscape SDKs), source code is available to allow for ports to other platforms. If your programming language of choice is not directly supported, you may still be able to use one of these SDKs. Most widely used programming and scripting

languages can call C, C++, or Java code. For example, Microsoft Visual Basic code can use C library functions, and JavaScript can use Java classes and methods.

LDAP Command-Line Tools

Some vendors provide a set of command-line tools that provide access to LDAP directories. These tools are useful for simple LDAP integration tasks, especially those for which performance isn't critical. LDAP command-line tools are good choices for batch-oriented applications as well.

The LDAP command-line tools can be called from a variety of languages, including compiled languages such as C++, Windows command language, any Unix shell-scripting language, and Perl. Netscape provides the following command-line tools with its directory SDKs and its Directory Server product:

- **ldapcompare**. Executes a series of LDAP compare operations.
- **ldapdelete**. Deletes one or more LDAP entries.
- **ldapmodify**. Adds, deletes, modifies, or renames one or more LDAP entries. The ldapmodify tool accepts LDIF as input to describe the changes to be made to the directory.
- **ldapsearch**. Executes a series of LDAP search operations. The output produced by the ldapsearch tool conforms to the LDIF specification.

Several other directory software vendors, such as OpenLDAP, provide similar tools.

LDAP Tag Libraries for Web Development

Templates and tag libraries have become popular in the Web development community as a way to cleanly separate the user interface from the application logic. To accomplish this separation, HTML or XML templates are used to describe the user interface in conjunction with embedded tags that trigger calls to application-specific code. The most popular Web page template frameworks are Microsoft's Active Server Pages (ASP) and Sun's JavaServer Pages (JSP).

Because of Java's popularity as a platform for writing portable Web applications, JSPs are widely used. JSP files are standard HTML files with some extra JSP tags thrown in. Several JSP tag libraries are available for accessing LDAP directories. One such library is Simya Consultancy's LDAP JSP Tag Library, which is available at <http://www.simya.net/products.html>. Listing 21.2 shows a simple JSP file that uses Simya's tag library to search an LDAP directory server and produce a table from the entries returned.

Listing 21.2 An LDAP JSP Example: *list-jensens.jsp*

```
1. <%@ taglib uri="/ldap" prefix="ldap" %>
2. <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
3. <html><head><title>User List</title></head>
```

```

4. <body>

5. <ldap:connect url="ldap://ldap.example.com">

6. <h2>List of Jensens in example.com</h2>

7. <table bgcolor="#F0F8FF" cellspacing="0" cellpadding="2" border="1">

8. <tr bgcolor="Silver">

9. <th>RDN</th>

10. <th>User ID</th>

11. <th>User Name</th>

12. <th>Telephone</th>

13. <th>Department</th></tr>

14. <ldap:query id="ldapresult" basedn="ou=People,dc=example,dc=com"

15. filter="(sn=jensen)" limit="100">

16. <tr>

17. <td><%= ldapresult.getDN() %></td>

18. <td><%= ldapresult.getStringAttribute("uid") %></td>

19. <td><ldap:getAttribute name="cn"/></td>

20. <td><ldap:getAttribute name="telephonenumber"/></td>

21. <td><ldap:getAttribute name="ou" delimiter="  
" /></td>

22. </tr>

23. </ldap:query>

24. </table>

25. </ldap:connect>

26. </body>

27. </html>

```

Line 1 contains a JSP tag library directive that causes the Simya LDAP tag library to be loaded. The JSP tags that begin with "<ldap:" are the Simya tags themselves that pull information out of the `example.com` LDAP server (for example, line 5 contains an `ldap:connect` tag). The tags that begin with "<%=" are JSP expressions (for example, line 17 includes an embedded expression that retrieves the distinguished name, or DN, of an entry in the list of search results). [Listing 21.2](#) also contains many standard HTML tags, some of which are used to produce a table. [Figure 21.10](#) shows a sample of the output produced by this JSP file, as viewed in a Web browser.

Figure 21.10. The `list-jensens.jsp` Example in Action

The screenshot shows a Netscape 6 browser window with the title bar 'User List - Netscape 6'. The menu bar includes 'File', 'Edit', 'View', 'Search', 'Go', 'Bookmarks', 'Tasks', and 'Help'. Below the menu is a toolbar with icons for Back, Forward, Stop, Home, and Search. The address bar shows the URL 'http://localhost:1080/ldap-jsp-simya/list-jensens.jsp'. A search bar with the placeholder 'Search' is also present. The main content area displays a table titled 'List of Jensens in example.com' with the following data:

RDN	User ID	User Name	Telephone	Department
uid=kjensen	kjensen	Kurt Jensen	+1 408 555 6127	Product Development People
uid=bjensen	bjensen	Barbara Jensen, Babs Jensen	+1 408 555 1862	Product Development People
uid=gjensen	gjensen	Gern Jensen	+1 408 555 3299	Human Resources People
uid=jjensen	jjensen	Jody Jensen	+1 408 555 7587	Accounting People
uid=ajensen	ajensen	Allison Jensen	+1 408 555 7892	Product Development People
uid=bjense2	bjense2	Bjorn Jensen	+1 408 555 5655	Accounting People
uid=tjensen	tjensen	Ted Jensen	+1 408 555 8622	Accounting People
uid=rjensen	rjensen	Richard Jensen	+1 408 555 5957	Accounting People
uid=rjense2	rjense2	Randy Jensen	+1 408 555 9045	Product Testing People

The Further Reading section near the end of this chapter provides pointers to additional JSP resources.

DSML Tools and SDKs

The Directory Services Markup Language (DSML) is a standard for representing directory data and operations using XML. The basic DSML version 1 specification (DSMLv1), which simply defines a way to represent directory data using XML, has existed since 1999. Some LDAP SDKs provide facilities for parsing DSMLv1, servers such as Netscape Directory Server 6 support DSMLv1 import and export, and some DSMLv1-specific tools do exist. But the real excitement centers on DSML version 2 (DSMLv2), which draws from the LDAP protocol standards to allow the complete set of LDAP directory operations to be represented with XML.

In addition, DSMLv2 defines a standard way to transport DSML over the Simple Object Access Protocol (SOAP), which is the preferred protocol for creating new Web services applications (SOAP is the foundation for Microsoft's .NET initiative, among other efforts). The combination of DSMLv2 and SOAP will open directory services to a new class of application developers, and it is widely expected that the prominent directory software vendors will add support for DSMLv2-over-SOAP to their products.

In April 2002, the DSMLv2 specification was approved by the members of the Organization for the Advancement of Structured Information Standards (OASIS).

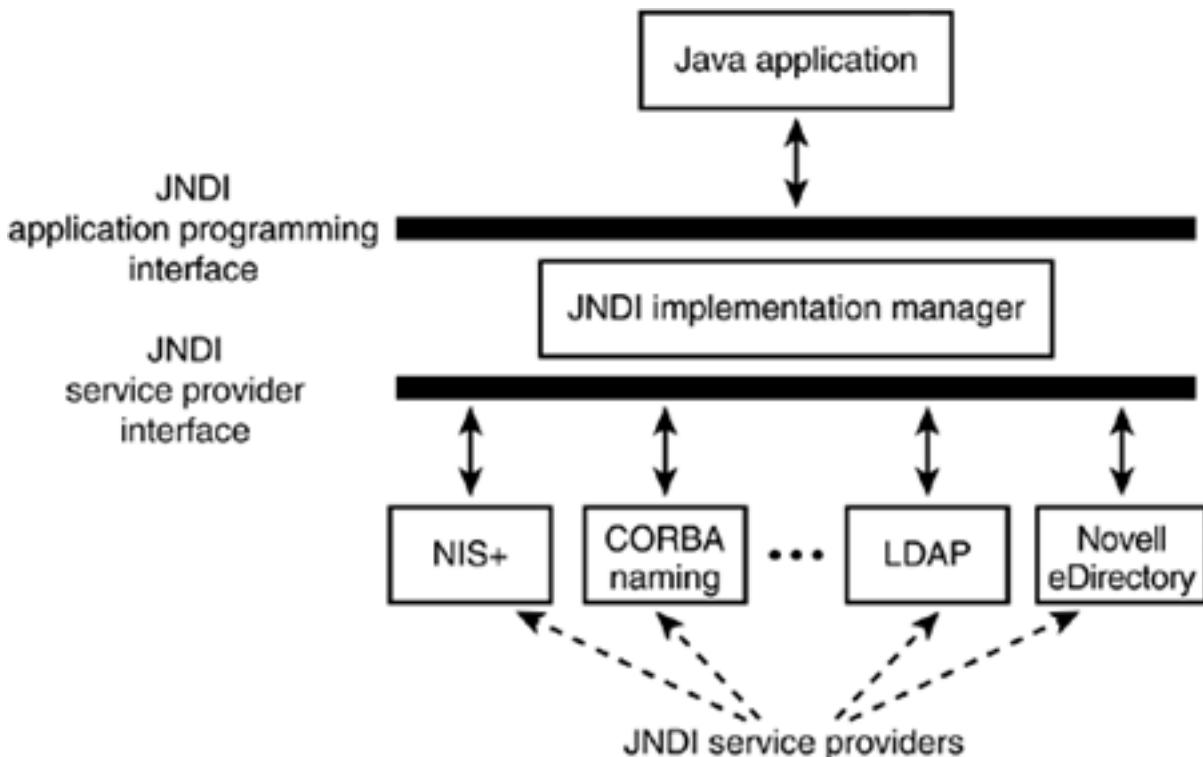
Now that DSMLv2 is a standard, expect to see a flurry of activity around DSML, including the release of a variety of new DSML tools and SDKs.

Directory-Agnostic SDKs

Your remaining choice is to use an SDK that provides access to a variety of directory and directory-like data sources (including LDAP). We use the term *directory-agnostic* to describe these SDKs. Examples include Microsoft's Active Directory Services Interface (ADSI) for the Windows platform and Sun's Java Naming and Directory Interface (JNDI). ADSI can be used with a variety of programming languages, including C++, Visual Basic, and VBScript. JNDI can be used only with Java.

Access to a variety of data sources is provided by a common, directory-independent API that application programmers may use. These SDKs are internally designed and constructed so that a variety of directory service modules can be plugged in beneath the API. A good example of this kind of SDK architecture is JNDI. [Figure 21.11](#) shows the internal architecture of the JNDI implementation.

Figure 21.11. The JNDI Architecture



A directory-agnostic SDK may be a good choice if your application needs access to directory services other than LDAP. The main disadvantages of these SDKs are that in some cases they do not provide full access to LDAP's capabilities and they do not evolve as fast as the SDKs that focus purely on LDAP. Other disadvantages include increased overhead and code size as a result of the extra layers and directory service modules included. In the case of ADSI, it may be difficult to call the SDK functions from existing applications that are not based on Microsoft's Component Object Model (COM). JNDI, on the other hand, has been enhanced to support most LDAP features, including LDAPv3 controls and extended operations. JNDI is included in Sun's Java Development Kits (JDKs) and is widely used.

Advice for LDAP Application Developers

As with most projects, the process of developing an LDAP directory-enabled application is filled with many choices. You can save yourself a lot of grief if you plan ahead and think about how your application will interact with the directory service and other applications. This section provides some advice to help you develop good LDAP applications. Additional information on data-related design considerations can be found in [Chapter 18](#), Maintaining Data.

Striving to Fit In

A common mistake made by new LDAP application developers is to treat the directory as a private application-specific store. Although in rare cases this approach might be valid, most LDAP directories focus on sharing information among applications, end users, and administrators. Also keep in mind that the design of the directory may be altered in the future to meet the needs of other applications or to accommodate organizational changes.

You should design and write your application to assume as little as possible about the directory design and to coexist well with other applications—including ones that have not yet been written. To accomplish these goals, follow these guidelines when designing and implementing your application:

- Use standard schema elements whenever practical. Using standard schema elements promotes interoperability and makes it easier to coexist with other applications, while also helping to simplify directory service software upgrades.
- When extending schemas, follow the procedures outlined in [Chapter 8](#), Schema Design. It is best to do these two things: (1) create auxiliary object classes so that you can add attributes to existing objects such as persons and (2) use an application- or organization-specific prefix when naming new attributes and object classes so that you avoid naming conflicts.
- Avoid using mandatory attributes when you define your auxiliary object classes. Suppose another application includes the same attribute type as an optional one in its own auxiliary object class. The application may encounter an error when it tries to add its own object class and attributes to an entry because it may not realize that the attribute is mandatory.
- Assume as little as possible about the directory namespace, topology, replication, and security design. Minimizing assumptions aids coexistence and helps your application continue to work when changes are made to the directory service deployment. Ideally you should anticipate what kind of directory design changes will affect your application and provide configuration settings or built-in intelligence to avoid problems resulting from such changes.

The key to developing an application that coexists well with other applications and can survive directory design changes is in knowing when it is OK to make assumptions. For example, it is probably unacceptable to assume much about how the people and group entries are arranged in the directory information tree (DIT). On the other hand, it is probably OK for your application to dictate the DIT structure within a portion of the directory namespace that it uses to store its own application-specific configuration information.

Tip

Make your application as configurable as possible with respect to how it uses the directory service. Do not hardwire knowledge of the DIT structure, access control rules, or other key parts of the directory design into your application. If

you do not expect certain things to change, it is OK to make them somewhat difficult to reconfigure. However, it is important to ensure that there is some way to change the configuration without rewriting or recompiling the application. Otherwise, if a critical directory-related assumption you made ceased to hold true, you would need to modify the application and distribute new copies everywhere. Examples of techniques you can use to build a flexible application include using configuration files or templates for directory-related settings and writing your application so that it reads schema information from the directory service itself.

Communicating Your Application's Directory Needs

As part of your application design process, document how the application interacts with the directory service. Share this information with the people responsible for the design and deployment of the directory service. Sometimes deployment of a new application requires modifications to the directory design or software upgrades, both of which may be perfectly acceptable to directory administrators if known in advance.

The directory administrators should be made aware of any anticipated directory changes as soon as possible so that the required planning and deployment tasks can be handled *before* your application is deployed. Some of the things you should communicate to them are

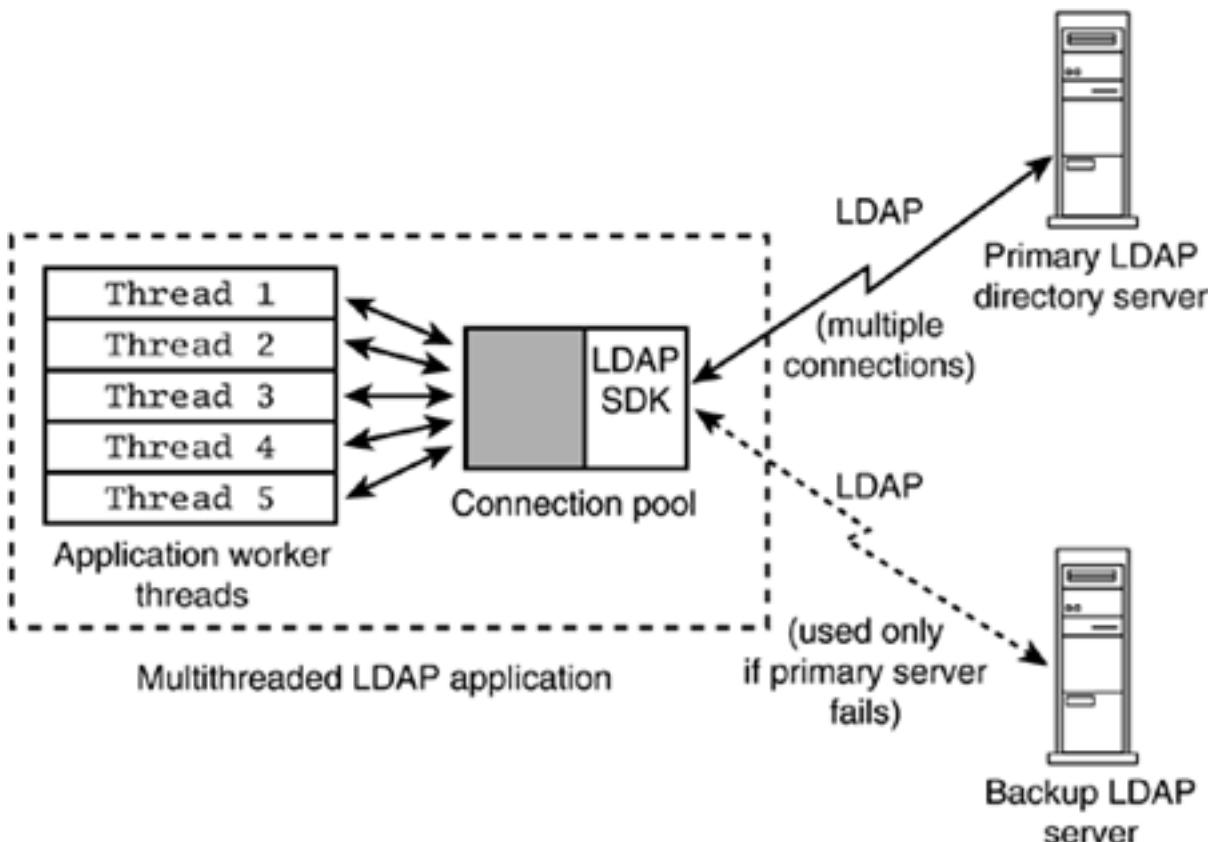
- Schema extensions required for your application.
- Security and access control changes needed to support your application.
- Privacy issues surrounding new data that your application may introduce into the directory.
- Additional directory management tasks required for your application. For example, it might make sense to extend certain existing administrative tools to support a new attribute used by your application.
- Data integration requirements for the application. For example, you may need to establish additional data synchronization procedures with nondirectory data sources to obtain some of the entries and attribute values needed by your application.
- Special LDAP protocol features you rely on. For example, you may be planning to use an LDAPv3 protocol extension that is new enough to require an upgrade of the directory server software. Sometimes such a feature may be required for your application, but if possible, try to create a mode in which the application works in the absence of the extension (perhaps with some loss of functionality). Doing so will ease deployment and allow a phased approach if directory server software upgrades are needed.
- An estimate of the additional load the application will impose on the service. This is typically measured in terms of the number of LDAP operations performed in a given time period. Don't forget to estimate peak loads and consider each operation separately. For example, search operations typically can be handled more quickly by an LDAP server than modify or add operations can.
- Server configuration changes required by your application, such as indexing of additional attributes.

Designing for Good Performance and Scalability

No two applications, and no two directory deployments, have the same performance and scalability requirements. In most organizations, however, the use of computing services and networks is rapidly increasing. To be safe, you should design your application to perform well with and accommodate a larger number of directory entries than are currently stored in your directory service. By considering performance and scalability of the directory-related aspects of your application up front, you may avoid a costly redesign later.

When you're designing your application, avoid architectural choices that limit performance, require you to open many connections to the directory service, or make it difficult to use performance-enhancing techniques such as caching. If your application uses LDAP heavily, you may want to implement a shared pool of connections to the directory service. [Figure 21.12](#) shows an example of a connection pool architecture.

Figure 21.12. An LDAP Connection Pool Architecture



A shared pool of connections offers several advantages:

- The pool provides a central place for caching search, bind, and compare operations and for sharing results among threads within your application.
- You can easily arrange to share one connection among several application threads, thereby reducing the number of simultaneous connections your application needs to make to the directory service.
- The code to support robust features (such as automatic failover to an alternate server) can be implemented in one place in the connection pool.

Although your application may not require a complex structure that includes a shared connection pool, do consider what kind of architecture will help your application use the directory service efficiently and perform well.

Tip

One danger in using any abstraction layer (including the LDAP connection pool just described) is that you typically sacrifice some fidelity and lose the ability to take full advantage of all the LDAP features provided by the underlying LDAP SDK. For example, suppose you implement a connection pool that does not allow you to use arbitrary LDAPv3 controls. If later you found that your application would work better if it used an LDAPv3 extension that requires you

to send a control to the server (such as a server-side sorting control), you would need to redesign your abstraction layer. Plan ahead, and use abstraction layers only when there is a clear advantage in doing so.

All LDAP applications perform better if they avoid issuing extraneous operations and if they use simple operations as much as possible. By minimizing your application's impact on CPU, disk, memory, and other directory server resources, you improve the performance of all applications. Here are some simple guidelines to follow:

- When searching and reading entries, ask for only the attributes you need. Do not retrieve all attributes from every entry.
- When modifying an entry, compare the new values with the ones already present in the entry and avoid sending unnecessary modify requests to the directory server.
- Include useful information such as an (`objectClass=person`) component in search filters in order to reduce the number of entries the server will need to examine to return the entries requested.
- Use efficient search filters whenever possible. For example, use an exact filter first, and then try a substring filter only if no entries are returned.
- Avoid building assumptions into your application about the total number of entries or the shape of the directory tree (for example, flat or deep).
- Think carefully before you create an application that relies on being able to find all the entries in the directory at once. This may not be feasible.
- If you expect a lot of entries to be returned from a search operation, take advantage of the LDAPv3 Virtual List View (VLV) control to build a scalable user interface that processes the entries in manageable chunks. Refer to [Chapter 3](#), LDAPv3 Extensions, for more information about the VLV control.
- If your application includes a user interface for selecting directory entries, make sure it supports a search-based interface instead of (or in addition to) a list-based interface. A search-based interface will help your application handle thousands of entries and scale up gracefully as the directory service itself scales up.

Developing a Prototype and Conducting a Pilot

One of the best ways to prove your design and try out alternatives is to develop a working prototype. The prototype need not be complete or have the prettiest interface. You can create a prototype using scripting tools or other rapid application development (RAD) techniques, even if you plan to use different development tools to build the final application. However, your prototype should work and interact with other systems (including the directory service) in the same way you expect the final version of your application to do so.

As soon as a working prototype exists, conduct a pilot test to solicit feedback and learn about problems that must be fixed before you put the application into production service. Test your application's features, measure its performance, and solicit feedback from beta testers. Capture log files and other performance information and use them to conduct scaling tests. Use the data from these tests to estimate the load your application will place on the production directory service.

Use all the data obtained during the pilot to improve your application, including how it interacts with end users, with the directory service, and with other applications. After your application is deployed, substantial changes to it may dictate that you conduct another pilot. It is always better to find problems in pilot mode rather than in production mode.

Leveraging Existing Code

In many cases source code is available that can assist in developing your own applications. Most SDKs come with sample code, and vendors such as Netscape, Novell, Sun, and Microsoft make additional sample code available informally in newsgroups, in developer newsletters, and on developer CD-ROMs. The Internet at large is a great resource for locating complete directory-enabled applications; applications with freely available source code include LDAP-enabled message transfer agents (MTAs), several HTTP-to-LDAP gateways, LDAP modules for popular servers such as Apache, and LDAP-enabled e-mail clients. One good place to search for LDAP projects is <http://sourceforge.net>.

You may want to create a Web site to publish LDAP-related source code within your own organization or publicly on the Internet on a site such as <http://sourceforge.net> so that the code can be reused by other people. Code modules explicitly designed to be reusable should be distributed in binary form as shared libraries or Java class files. Over time, try to build a large collection of code to serve as the starting point for new applications. You should, of course, ensure that any code promoted for reuse is of good quality and demonstrates appropriate use of the directory service.

Avoiding Common Mistakes

As when writing applications for any service that provides a lot of flexibility and functionality, the process of writing LDAP applications has some pitfalls. Many newcomers to LDAP application development make the same mistakes as those who came before them. Here are some common mistakes you should avoid:

- **Allowing abusive searches to get through to the directory service.** A search that consumes a lot of resources on a directory server is an *abusive search*. Abusive searches include unindexed searches, searches that return a huge number of entries, and searches that read more information than is needed for the task at hand. If your application provides an interface with which end users interact to create LDAP search filters, implement simple checks to avoid abuse of the server. The kinds of searches that are abusive vary from one directory server implementation to another.

For example, with Netscape Directory Server it is important to avoid substring filters that use fewer than three characters (for example, `cn=*hi*`). You should also ensure that at least one ANDed component of each search filter references an indexed attribute type. Unindexed searches processed by Netscape Directory Server are flagged with `notes=U` in the server access log. You can also examine the access log for searches whose elapsed time (`etime`) is longer than a few seconds.

- **Not taking advantage of LDAP client-side size and time limits.** To prevent searches from taking too long and monopolizing the directory service, use reasonable size and time limits in your application. Using these limits also protects your application and your users from being frustrated by a slow, unresponsive service.
- **Treating multiple values stored in one attribute as an ordered list.** In LDAP, the attribute values associated with each attribute type are treated as an unordered set of values. Although many directory servers return the values for a given attribute in the same order they're stored, it is dangerous for an application to rely on this behavior. If you need a way to distinguish between two or more values for the same attribute, include something in the values themselves (such as a sequence number or other identifying text) to allow your application to tell the values apart.
- **Storing extremely large entries in the directory service.** Most directory server and SDK implementations are designed with relatively small entries in mind. There is no hard-and-fast rule for how big is too big, but it is unusual for an entry to be larger than 100K.

- **Improperly handling zero-length passwords.** If your application uses LDAP simple bind operations to test for valid passwords, make sure you check for zero-length or NULL passwords before passing them on to the directory server. LDAP servers return success responses for simple bind operations that contain a zero-length password and a syntactically valid DN, even though the server does not grant the privileges associated with the bind DN to the connection. In other words, LDAP simple bind operations with zero-length passwords always appear to the application to succeed when, in fact, no credentials were checked and no privileges were granted.
- **Improperly supporting LDAP referrals.** The most common mistake users of the LDAP C API make with respect to referral support is forgetting to install a rebinding function when automatic referral chasing is enabled. A *rebind function* is a callback function that is invoked by the SDK library to obtain credentials to use when connecting to another server. Without a rebinding function in place, all referred connections are anonymous, causing errors or confusion when the directory is being modified or attributes protected by access control are being accessed.

Another common referral-related mistake is not providing a way to disable automatic chasing of referrals. The ability to disable referrals is useful when you're troubleshooting directory problems or trying to improve performance in the face of unreachable servers.

- **Reading an entry immediately after it was modified and expecting to see recent changes.** A short delay before changes made on a writable directory server are visible on a read-only replica is common. See [Chapter 11](#), Replication Design, for a discussion of the loosely consistent replication strategy employed by most directory products. If your application is connected to a read-only directory replica, searches sometimes return stale data. Try to design your application so that it doesn't need to immediately read information that it just changed and, if appropriate, so that it warns users to expect a short delay before directory changes are visible. If it is essential for your application to immediately see the changes it makes, design it so that it can locate a writable directory server. The writable server can then be used to process reads as well as writes once a change has been made by your application (of course, requiring use of a writable server for reads diminishes the value of replication somewhat). Using a client-side "write-through" cache to hide replication propagation delay is also an option, but if the contents of the cache are flushed (when a user quits and reloads your application, for example), old directory data may reappear and confuse the user.

Example 1: setpwd, a Password-Resetting Utility

Because end users often forget their passwords, one of the most common tasks for Help Desk personnel is to reset or assign new passwords. In this section, a command-line utility named setpwd is presented that resets the password in a user's directory entry (either to a randomly chosen value, or to a password value supplied on the setpwd command line).

Directory Use

The setpwd utility uses a directory service in a simple way. The LDAP service is always accessed by Help Desk personnel who have permission to modify the `userPassword` attribute in all person entries. Given a user ID that is supplied on its command line, the setpwd application locates a person's directory entry using an LDAP search operation. If exactly one entry is found, the existing `userPassword` values in the entry are replaced with a new password value.

Because this is a simple directory application, there are no significant application architecture issues to consider. The setpwd application relies on standard schemas and avoids reading unnecessary attributes from the user entries. In addition, setpwd uses only one LDAP connection even when it is asked to reset several users' passwords.

The Help Desk Staff's Experience

Because the setpwd application is meant for use by Help Desk staff, it is designed to be efficient and easy to use. This section provides some usage examples. [Listing 21.3](#) shows one sample run of the setpwd application. In this example, Sam Carter's password is reset to the random password "marsh!sightseer8."

Listing 21.3 kvaughan Uses setpwd to Reset scarter's Password

```
setpwd scarter
LDAP password for kvaughan: secret
Contacting LDAP server...
Finding scarter's entry...
Modifying scarter's entry...
Password reset. scarter's new password is: marsh!sightseer8.
Don't forget to remind scarter to reset his/her password right away!
```

[Listing 21.4](#) shows another sample run of the setpwd utility. In this instance, seven people's passwords are reset with a single run of the utility. The `-q` (quiet) option is used to reduce the amount of output, the `-h` option is used to specify the LDAP server (`ldap2.example.com`), and the `-n` option is used to set the password to a specific value (`Wel,come43`).

Listing 21.4 kvaughan Uses setpwd to Reset Several Users' Passwords

```
setpwd -q -h ldap2.example.com -n "Wel,come43" scarter tmorris abergin dmiller
gfarmer
```

↳ **kwinters trigden**

```
LDAP password for kvaughan: secret
scarter's new password is: Wel,come43.
tmorris's new password is: Wel,come43.
abergin's new password is: Wel,come43.
dmiller's new password is: Wel,come43.
gfarmer's new password is: Wel,come43.
kwinters's new password is: Wel,come43.
trigden's new password is: Wel,come43.
```

The Source Code

The setpwd utility is written in C, and it uses the Netscape/Mozilla Directory SDK for C (which provides a standard LDAPv3 C API). Nearly all the functions that are part of the LDAP C API begin with the prefix "ldap_". The code was compiled and tested under the Sun Solaris 8 Unix operating system, as well as Microsoft Windows 2000, although it should be easy to port to other systems and implementations of the LDAP C API. The source code for setpwd consists of one file that is named, logically enough, **setpwd.c**. The code is presented here in pieces to aid understanding.

[Listing 21.5](#) shows the beginning of the **setpwd.c** file. Notice the inclusion of the standard LDAP header file on line 10.

Listing 21.5 The `setpwd.c` Prelude

```
1. /* -*- Mode: C; tab-width: 4; c-basic-offset: 4 -*- */
2. /*
3.  * A simple password resetting program for use by a help desk.
4. */
5. #include <stdio.h>
6. #include <string.h>
7. #include <stdlib.h>
8. #include <ctype.h>
9. #include <time.h>
10. #include <ldap.h>
11. #ifdef _WINDOWS
12. #include <WINUSER.H>
13. #include <LMCONS.H>
14. #include "getopt.h"
```

```
15. #include "getpass.h"
16. #else
17. #include <unistd.h>
18. #include <pwd.h>
19. #endif
20.
21.
22. /*
23. * Macros:
24. */
25. #define DEFAULT_SEARCHBASE "ou=people,dc=example,dc=com"
26. #define DEFAULT_LDAPHOST "directory.example.com"
27. #define DEFAULT_LDAPPORT LDAP_PORT
28. #define DEFAULT_WORDFILE "/usr/dict/words"
29.
30. #define LONGEST_WORD 100
31.
32. #ifdef sun
33. #define GETPASS getpassphrase /* accepts a longer string */
34. #else
35. #define GETPASS getpass /* the old standby */
36. #endif
37.
38. #ifdef _WINDOWS
39. #define SRANDOM( seed ) srand( seed )
40. #define RANDOM() rand()
41. #else
42. #define SRANDOM( seed ) srandom( seed )
43. #define RANDOM() random()
44. #endif
45.
46.
47. /*
```

```

48. * Global variables:
49. */
50. static int quiet = 0; /* use the -q flag for quiet mode */
51. char *wordfile = DEFAULT_WORDFILE;
52. char *binddn = NULL;
53. char *bindpwd = NULL;
54.

```

A series of LDAP-related macros are defined on lines 25 to 27. The default LDAP host, port, and search base are compiled into this application, which is convenient for the Help Desk personnel. The `DEFAULT_WORDFILE` macro defined on line 28 has nothing to do with LDAP; it is used by `setpwd`'s random password generation code (described later, in [Listing 21.12](#)). The remainder of the code in [Listing 21.5](#) consists of some macros to hide platform differences and a few global variables.

[Listing 21.6](#) includes prototypes for static functions, as well as the `usage()` function. The `usage()` function is straightforward. To display the usage, invoke `setpwd` without any command-line parameters or with the `-H` parameter.

Listing 21.6 Function Prototypes and `usage()`

```

55.
56. /*
57. * Function prototypes:
58. */
59. static int resetpwd( LDAP *ld, const char *base,
60.                      const char *userid, const char *newpwd );
61. static char *userid2dn( LDAP *ld, const char *base,
62.                        const char *userid );
63. static void print_ldap_error( LDAP *ld, int lderr, const char *s );
64. static char *randompwd( void );
65. static char *randomword( void );
66. static int LDAP_CALL LDAP_CALLBACK get_rebind_credentials( LDAP *ld,
67.                           char **dnp, char **passwdp, int *authmethodp,
68.                           int freeit, void *arg );
69.
70. /*
71. */

```

```

72. * Functions:
73. */
74. static void
75. usage( const char *progname )
76. {
77.     fprintf( stderr, "usage: %s [options] userid...\n", progname );
78.     fprintf( stderr, "where the options are:\n" );
79.     fprintf( stderr, "\t-H           help (display usage)\n" );
80.     fprintf( stderr, "\t-R           do not follow referrals\n" );
81.     fprintf( stderr, "\t-q           operate quietly\n" );
82.     fprintf( stderr, "\t-h host      LDAP server name\n" );
83.     fprintf( stderr, "\t-p port      LDAP server port\n" );
84.     fprintf( stderr, "\t-b basedn    base DN for searches\n" );
85.     fprintf( stderr, "\t-u bindid    user id to bind as\n" );
86.     fprintf( stderr, "\t-D binddn    DN to bind as\n" );
87.     fprintf( stderr, "\t-w passwd    bind password\n" );
88.     fprintf( stderr, "\t-n newpasswd  password to set\n" );
89.     fprintf( stderr, "\t-d file      words file, e.g. "
90.             DEFAULT_WORDFILE "\n" );
91.     exit( 2 );
92. }
93.

```

[Listing 21.7](#) shows the first part of the `setpwd main()` function. The code on lines 108 to 121 retrieves the current user name from the operating system to use as the default user ID for authentication. The remaining code in [Listing 21.7](#) uses the `getopt()` library function to process the command-line arguments (on Microsoft Windows, `getopt()` is not a standard library function, so in that case an open-source `getopt()` implementation is used).

Listing 21.7 The `setpwd main()` Function (Part 1 of 2)

```

94. int
95. main( int argc, char **argv )
96. {
97.     LDAP          *ld;

```

```
98.     int          i, c, errflg, rc;
99.     int          ldapv3 = LDAP_VERSION3;
100.    void         *follow_referrals = LDAP_OPT_ON;
101.    const char   *host = DEFAULT_LDAPHOST;
102.    int          port = DEFAULT_LDAPPRT;
103.    const char   *base = DEFAULT_SEARCHBASE;
104.    const char   *binduserid = NULL;
105.    const char   *newpwd = NULL;
106.    const char   *bindidstring = NULL;
107.
108. #ifdef _WINDOWS
109.     char         useridbuf[ UNLEN + 1 ];
110.     DWORD        useridlen = sizeof(useridbuf);
111.
112.     if ( GetUserName( useridbuf, &useridlen ) ) {
113.         binduserid = useridbuf;
114.     }
115. #else
116.     struct passwd *pwdinfo;
117.
118.     if ( NULL != ( pwdinfo = getpwuid( getuid() ) ) ) {
119.         binduserid = strdup( pwdinfo->pw_name );
120.     }
121. #endif
122.
123.     /*
124.      * Process command line arguments.
125.      */
126.     errflg = 0;
127.     while ( ( c = getopt( argc, argv, "HRqh:p:b:u:D:w:n:d:" ) ) != EOF ) {
128.         switch ( c ) {
129.             case 'H': /* help */
130.                 errflg = 1;
```

```
131.         break;
132.     case 'R': /* do not follow referrals */
133.         follow_referrals = LDAP_OPT_OFF;
134.         break;
135.     case 'q': /* quiet */
136.         quiet = 1;
137.         break;
138.     case 'h': /* host */
139.         host = optarg;
140.         break;
141.     case 'p': /* port */
142.         port = atoi( optarg );
143.         break;
144.     case 'b': /* port */
145.         base = optarg;
146.         break;
147.     case 'u': /* bind user id */
148.         binduserid = optarg;
149.         break;
150.     case 'D': /* bind DN */
151.         binddn = optarg;
152.         break;
153.     case 'w': /* bind password */
154.         bindpwd = optarg;
155.         break;
156.     case 'n': /* new password */
157.         newpwd = optarg;
158.         break;
159.     case 'd': /* words file */
160.         wordfile = optarg;
161.         break;
162.     default:
```

```

163.         errflg = 1;
164.     }
165. }
166. if ( errflg || optind >= argc ||
167.         ( binduserid == NULL && binddn == NULL ) ) {
168.     usage( argv[ 0 ] );
169. }
170.
171. if ( binddn != NULL ) {
172.     bindidstring = binddn;
173. } else {
174.     bindidstring = binduserid;
175. }

```

Tip

Although the `setpwd` program includes default values for all its LDAP-related settings, it also supports an extensive set of command-line options. These options ensure that the program is convenient for Help Desk staff to use, but also flexible enough to be used in other situations. Consider similar factors when writing your own LDAP applications.

[Listing 21.8](#) shows the remainder of `main()`. The code on lines 177 to 195 prompts the user for a password if none was provided on the command line. Finally, the first LDAP API call is made on line 202. The `ldap_init()` call creates an LDAP session handle for our LDAP server host and port. Note that the `ldap_init()` function call does not open a connection to the LDAP server. A connection to the LDAP server is opened the first time an LDAP request is made.

Listing 21.8 The `setpwd main()` Function (Part 2 of 2)

```

176.
177. /*
178. * Prompt for password if not provided on command line.
179. */
180. if ( NULL == bindpwd || '\0' == *bindpwd ) {
181.     const char *p = "LDAP password for";
182.     char *prompt;
183.
184.     prompt = malloc( strlen( p ) + strlen( bindidstring ) + 3 );

```

```
185.         if ( prompt == NULL ) {
186.             perror( "malloc" );
187.             return 1;
188.         }
189.         sprintf( prompt, "%s %s: ", p, bindidstring );
190.         bindpwd = GETPASS( prompt );
191.         free( prompt );
192.         if ( NULL == bindpwd || '\0' == *bindpwd ) {
193.             return 0;
194.         }
195.     }
196.
197.     /*
198.      * Connect and bind to the directory server, looking up the bind DN
199.      * if a bind user ID was provided.
200.     */
201.     if ( !quiet ) puts( "Contacting LDAP server..." );
202.     if (( ld = ldap_init( host, port ) ) == NULL ) {
203.         perror( host );
204.         return 1;
205.     }
206.
207.     if ( ldap_set_option( ld, LDAP_OPT_PROTOCOL_VERSION,
208.                           &ldapv3 ) != 0 ) {
209.         print_ldap_error( ld, ldap_get_lderrno( ld, NULL, NULL ),
210.                           "set protocol version" );
211.         ldap_unbind( ld );
212.         return 1;
213.     }
214.
215.     if ( ldap_set_option( ld, LDAP_OPT_REFERRALS,
216.                           follow_referrals ) != 0 ) {
```

```

217.     print_ldap_error( ld, ldap_get_lderrno( ld, NULL, NULL),
218.                         "set referral following option" );
219.     ldap_unbind( ld );
220.     return 1;
221. }
222.
223. if ( follow_referrals == LDAP_OPT_ON ) {
224.     if ( ldap_set_option( ld, LDAP_OPT_REBIND_FN,
225.                           (void *)get_rebind_credentials ) != 0 ) {
226.         print_ldap_error( ld, ldap_get_lderrno( ld, NULL, NULL),
227.                         "set referral rebind function" );
228.         ldap_unbind( ld );
229.         return 1;
230.     }
231. }
232.
233. if ( binduserid != NULL ) {
234.     binddn = userid2dn( ld, base, binduserid );
235.     if ( binddn == NULL ) {
236.         return 1;
237.     }
238. }
239.
240. rc = ldap_simple_bind_s( ld, binddn, bindpwd );
241. if ( rc != LDAP_SUCCESS ) {
242.     print_ldap_error( ld, rc, bindidstring );
243.     ldap_unbind( ld );
244.     return 1;
245. }
246.
247.
248. /*
249. * Process user ID arguments, setting random passwords

```

```

250.     * for each one.
251.     */
252.     rc = 0;
253.     SRANDOM( time( NULL ) );
254.     for ( i = optind; i < argc; ++i ) {
255.         if ( !quiet && i > optind ) {
256.             putchar( '\n' );
257.         }
258.         rc |= resetpwd( ld, base, argv[ i ], newpwd );
259.     }
260.
261.     /*
262.      * Close LDAP server connection and exit.
263.     */
264.     ldap_unbind( ld );
265.     return rc;
266. }
267.

```

The code on lines 207 to 231 calls the `ldap_set_option()` LDAP API function three times to set various options that affect the LDAP session:

1. The `LDAP_OPT_PROTOCOL_VERSION` value is set to ensure that LDAPv3 is used.
2. The `LDAP_OPT_REFERRALS` option is set to the value in the `follow_referrals` local variable, which is either `LDAP_OPT_ON` (the default) or `LDAP_OPT_OFF` (if the `-R` command-line option is used). This LDAP session option controls whether LDAP referrals and references are automatically followed by the underlying LDAP API implementation.
3. If referral following is enabled, the `LDAP_OPT_REBIND_FN` option is set to the address of a function named `get_rebind_credentials()`. The LDAP API implementation calls this rebinding function when it needs to obtain authentication credentials while following a referral to another LDAP server. The code for the `get_rebind_credentials()` function is shown later (in [Listing 21.11](#)).

If a bind DN was not provided, the code on lines 233 to 238 uses the `userid2dn()` function (described later, in [Listing 21.10](#)) to search the LDAP server for a person entry matching the user ID. The `ldap_simple_bind_s()` call on line 240 connects to the directory server, issues an LDAP bind request, and waits for the server response. A return code of `LDAP_SUCCESS` indicates that the DN and password were accepted and that the authentication was successful.

The code on lines 248 to 259 moves through the arguments provided on the command line and passes each one to the `resetpwd()` function, which will be described next, in [Listing 21.9](#). The `ldap_unbind()` call included in the cleanup code on lines 261 to 265 does two things: It closes the LDAP connection, and it disposes of the memory and any other resources consumed by the LDAP session handle.

Tip

The code in [Listing 21.8](#) demonstrates proper handling of zero-length passwords (requiring that the user enter a password that is not empty, which ensures that a zero-length password is never sent to the LDAP server) and proper installation of a rebind function. Make sure that the LDAP application code you write addresses these issues correctly.

[Listing 21.9](#) shows the code for the `resetpwd()` function, which is responsible for resetting one user's password. The first task performed by this code is to find the user's directory entry (we need to know the DN to be able to modify the user's entry). The code on lines 283 to 289 calls the `userid2dn()` utility function (described next, in [Listing 21.10](#)) to find and return the user's DN, given her user ID. The code on lines 291 to 295 optionally calls another utility function, named `randompwd()` (described later, in [Listing 21.12](#)), which generates a new password for the user. The `randompwd()` function is not used if a password was provided on the command line through use of the `-n` option.

Listing 21.9 The `setpwd resetpwd()` Function

```
268.  
269. /*  
270. * resetpwd(): set an entry's password.  
271. * If newpwd is NULL, a random password is used.  
272. *  
273. * Returns 0 on success and 1 on failure.  
274. */  
275. static int  
276. resetpwd( LDAP *ld, const char *base,  
277.             const char *userid, const char *newpwd )  
278. {  
279.     char      *dn, *pwd, *vals[2];  
280.     int       rc;  
281.     LDAPMod  mod, *mods[2];  
282.  
283.     /*  
284.      * Find the entry.
```

```
285.     */
286.     if ( !quiet ) printf( "Finding %s's entry...\n", userid );
287.     if (( dn = userid2dn( ld, base, userid ) ) == NULL ) {
288.         return 1;
289.     }
290.
291.     if ( newpwd != NULL ) {
292.         pwd = (char *)newpwd;
293.     } else if (( pwd = randompwd() ) == NULL ) {
294.         return 1;
295.     }
296.
297.     /*
298.      * Replace the userPassword attribute.
299.     */
300.     if ( !quiet ) printf( "Modifying %s's entry...\n", userid );
301.     vals[0] = pwd;
302.     vals[1] = NULL;
303.     mod.mod_values = vals;
304.     mod.mod_type = "userPassword";
305.     mod.mod_op = LDAP_MOD_REPLACE;
306.     mods[0] = &mod;
307.     mods[1] = NULL;
308.
309.     if (( rc = ldap_modify_s( ld, dn, mods ) ) == LDAP_SUCCESS ) {
310.         rc = 0;
311.         if ( quiet ) {
312.             printf( "%s's new password is: %s.\n", userid, pwd );
313.         } else {
314.             printf( "Password reset.  %s's new password is: %s.\n",
315.                     userid, pwd );
316.             printf( "Don't forget to remind %s to reset "
317.                     "his/her password right away!\n", userid );
```

```

318.         }
319.     } else {
320.         print_ldap_error( ld, rc, "modify" );
321.         rc = 1;
322.     }
323.
324.     if ( newpwd == NULL ) {
325.         free( pwd );
326.     }
327.     ldap_memfree( dn );
328.     return rc;
329. }
330.

```

The remainder of the `resetpwd()` code (lines 297–329) replaces the `userPassword` attribute in the user's entry with the newly generated password. After setting up a single-element array of modifications, the `ldap_modify_s()` LDAP SDK function (line 309) is called to direct the server to make the password change.

[Listing 21.10](#) shows the source code for the `userid2dn()` utility function, which is called from `main()` and from `setpwd()`. The `userid2dn()` function uses an LDAP subtree search to find a user's directory entry, given her user ID.

Listing 21.10 The `setpwd` `userid2dn()` Function

```

331.
332. /*
333. * userid2dn(): Given an entry's user ID, return its DN.
334. *
335. * The DN returned should be freed by being passed to
336. * ldap_memfree(). If no entry is found, NULL is returned.
337. */
338. static char *
339. userid2dn( LDAP *ld, const char *base, const char *userid )
340. {
341.     int          rc;
342.     const char  *filterpattern = "(&(objectClass=person)(uid=%s))";

```

```

343.     char      *filter, *dn = NULL;
344.     char      *attrs[] = { LDAP_NO_ATTRS, NULL };
345.     LDAPMessage *e, *res = NULL;
346.
347.     if (( filter = (char *)malloc( strlen( userid ) +
348.                           + strlen( filterpattern ) + 1 ) ) == NULL ) {
349.         perror( "malloc" );
350.         return NULL;
351.     }
352.
353.     sprintf( filter, filterpattern, userid );
354.     rc = ldap_search_s( ld, base, LDAP_SCOPE_SUBTREE,
355.                          filter, attrs, 0, &res );
356.     free( filter );
357.
358.     if ( rc != LDAP_SUCCESS ) {
359.         print_ldap_error( ld, rc, "search" );
360.     } else if (( e = ldap_first_entry( ld, res ) ) == NULL ) {
361.         fprintf( stderr, "No match for user id %s.\n", userid );
362.     } else if ( ldap_next_entry( ld, e ) != NULL ) {
363.         fprintf( stderr, "%d matches for %s.\n",
364.                  ldap_count_entries( ld, res ), userid );
365.     } else {
366.         dn = ldap_get_dn( ld, e ); /* found just one! */
367.     }
368.
369.     ldap_msgfree( res );
370.     return dn;
371. }
372.

```

The code on lines 347 to 353 creates an LDAP search filter that looks like `(&(objectClass= person)(uid=ID))`, where **ID** is the user ID provided on the setpwd command line. This search filter will

match all entries that have the `person` object class and have the correct user ID. The actual search is done by the `ldap_search_s()` LDAP SDK call that appears on lines 354 and 355. Notice that the `attrs` parameter, which is an array of attribute types to be returned with each entry found during the search, contains one type named `LDAP_NO_ATTRS` (`attrs` is initialized on line 344).

The special `LDAP_NO_ATTRS` macro, defined by the LDAP API, is used to indicate that no attribute types should be returned. Using `LDAP_NO_ATTRS` is desirable because it is wasteful to ask for attributes we do not need, and in this instance we are interested only in obtaining the DN of the entry. If we were to pass `NULL` for the `attrs` parameter, every attribute in the entry would be returned, which would be less efficient. The code on lines 358 to 370 checks if exactly one entry was found; if so, it returns the DN of the entry. If more than one entry was found, or if no entries were found, an error is reported.

[Listing 21.11](#) shows the code for the `get_rebind_credentials()` rebinding function and the `print_ldap_error()` utility function.

Listing 21.11 The setpwd `get_rebind_credentials` and `print_ldap_error()` Functions

```
373.  
374. /*  
375. * Callback function for LDAP bind credentials; used when  
376. * rebinding during referral and reference following.  
377. */  
378. static int LDAP_CALL LDAP_CALLBACK  
379. get_rebind_credentials( LDAP *ld, char **dnp, char **passwdp,  
380.                         int *authmethodp, int freeit, void *arg )  
381. {  
382.     if ( !freeit ) {  
383.         *dnp = binddn;  
384.         *passwdp = bindpwd;  
385.         *authmethodp = LDAP_AUTH_SIMPLE;  
386.     }  
387.  
388.     return LDAP_SUCCESS;  
389. }  
390.  
391.  
392. /*  
393. * Display an LDAP error message.
```

```

394. */
395. void
396. print_ldap_error( LDAP *ld, int lderr, const char *s )
397. {
398.     fprintf( stderr, "%s: %s\n", s, ldap_err2string( lderr ) );
399. }
400.

```

Recall that the `get_rebind_credentials()` function is called by the LDAP C API implementation when it needs to obtain bind credentials while following a referral. The code simply sets the `dnp`, `passwdp`, and `authmethodp` return parameters to appropriate values (`binddn` and `bindpwd` are global variables that are set in the `setpwd main()` function). This rebind function is actually called twice by the LDAP API implementation each time bind credentials are needed—once with the `freeit` parameter set to zero (to request the credentials), and a second time with `freeit` set to a nonzero value (to give the application an opportunity to dispose of any memory allocated during the first call).

The `print_ldap_error()` function is a simple utility function that is called from several places in `setpwd.c`. It reports LDAP-related errors to the user in a consistent way.

[Listing 21.12](#) shows the `randompwd()` function, which the `resetpwd()` function that we already examined calls if a password must be generated. This code does not perform any LDAP-related functions. It uses a file that contains a series of words (one per line) and the standard `random()` number generator function to select a new password. The algorithm could easily be altered to match a specific organization's security policy or the preferences of Help Desk technical staff.

Listing 21.12 The `setpwd randompwd()` Function

```

401.
402. /*
403. * randompwd(): generate a reasonably good password using some
404. * random words from a file. The password should be easy to
405. * remember and able to be passed on verbally to a user.
406. *
407. * The algorithm we use is this:
408. *   a) select two random words from the words file (w1 and w2).
409. *   b) pick a random punctuation character (c).
410. *   c) pick a random digit 0-9 (d).
411. *   d) construct a candidate password as: w1 c w2 d.
412. *   e) change the case of one of the letters at random.
413. */

```

```
414. static char *
415. randompwd( void )
416. {
417.     static char *punctuation = "!@#$%&*-+=,.?";
418.     char     *pwd, *w1, *w2, *p;
419.
420.     if (( w1 = randomword() ) == NULL
421.         || ( w2 = randomword() ) == NULL ) {
422.         if ( w1 != NULL ) {
423.             free( w1 );
424.         }
425.         return NULL;
426.     }
427.
428.     if (( pwd = (char *)malloc( LONGEST_WORD * 2 + 3 ) ) != NULL ) {
429.         sprintf( pwd, "%s%c%s%d", w1,
430.                  punctuation[ RANDOM() % strlen( punctuation ) ], w2,
431.                  RANDOM() % 10 );
432.         p = pwd + ( RANDOM() % strlen( pwd ) );
433.         if ( isalpha( *p ) ) {
434.             if ( isupper( *p ) ) {
435.                 *p = tolower( *p );
436.             } else {
437.                 *p = toupper( *p );
438.             }
439.         }
440.     }
441.
442.     free( w1 );
443.     free( w2 );
444.     return pwd;
445. }
```

446.

The `randompwd()` function uses a utility function named `randomword()` to pick the two words that it uses to construct the password value. Listing 21.13 shows the first part of the `randomword()` function, which consists mainly of initialization tasks. The `randomword()` function reads the random words from a text file that contains one word per line. By default, the standard Unix `/usr/dict/words` file is used, but any file is acceptable.

Listing 21.13 The `setpwd` `randomword()` Function (Part 1 of 2)

447.

```
448. /*
449. * randomword(): return a random, lowercase word.
450. *
451. * Returns a malloc'd string if successful.
452. * Returns NULL on error, e.g., if unable to access
453. * the words file.
454. *
455. * This function assumes that all words in the file are
456. * less than LONGEST_WORD characters long.
457. */
458. static char *
459. randomword( void )
460. {
461.     static FILE *fp = NULL;
462.     static long filesize = 0L;
463.     char     *word, *p;
464.     int      len;
465.
466.     if ( fp == NULL ) {
467.         /*
468.             * initialize: open the words file and
469.             * determine its size
470.         */
471.         if (( fp = fopen( wordfile, "r" ) ) == NULL
472.             || fseek( fp, 0L, SEEK_END ) == -1
```

```

473.             || ( filesize = ftell( fp ) ) == -1 ) {
474.                 perror( wordfile );
475.             if ( fp != NULL ) {
476.                 fclose( fp );
477.             }
478.             return NULL;
479.         }
480.
481.         if ( filesize < LONGEST_WORD ) {
482.             fprintf( stderr, "%s: the words file must be at"
483.                     " least %d bytes long.\n",
484.                     wordfile, LONGEST_WORD );
485.             if ( fp != NULL ) {
486.                 fclose( fp );
487.             }
488.             return NULL;
489.         }
490.
491.         filesize -= LONGEST_WORD;
492.     }
493.

```

[Listing 21.14](#) shows the remainder of the `randomword()` function, which seeks to a random location within the words file and returns one word.

Listing 21.14 The `setpwd` `randomword()` Function (Part 2 of 2)

```

494.     /*
495.      * Seek to a random location in the file.
496.     */
497.     if ( fseek( fp, RANDOM() % filesize, SEEK_SET ) == -1 ) {
498.         perror( wordfile );
499.         return NULL;
500.     }

```

```

501.

502.     if (( word = (char *)malloc( LONGEST_WORD ) ) == NULL ) {

503.         perror( "malloc" );
504.
505.     }
506.
507.     /*
508.      * Read the tail (remainder) of one word and then
509.      * the entire next word, which is what we will return.
510.      */
511.     fgets( word, LONGEST_WORD - 1, fp );
512.
513.     if ( fgets( word, LONGEST_WORD - 1, fp ) == NULL ) {
514.
515.         perror( wordfile );
516.         free( word );
517.         return NULL;
518.     }
519.     len = strlen( word ) - 1;
520.     if ( word[ len ] == '\n' ) {
521.         word[ len ] = '\0';
522.     }
523.     for ( p = word; *p != '\0'; ++p ) {
524.         if ( isupper( *p ) ) {
525.             *p = tolower( *p );
526.         }
527.     }
528.
529.     return( word );
530. }
```

Ideas for Improvement

The setpwd application could be improved in many ways. Here are a few ideas:

- Increase security by using a different form of authentication (such as Kerberos or public key certificates) and by protecting the LDAP traffic using SSL or TLS.
- Increase security by taking advantage of your directory service's password policy features (if available) to force users to choose a new password the next time they bind to the directory after their password is reset using setpwd.
- Expand the application's role by adding features to disable a user's account and change other information in user entries, such as e-mail addresses.
- Create reusable, shared libraries that contain some of the LDAP-related code included in setpwd. Good candidates include the `userid2dn()` and `print_ldap_error()` functions.

Example 2: SimpleSite, a Web Site with User Profile Storage

Many Web sites allow people to register with the site to enter profile information and to support authenticated services. This section describes an application that implements a simple Web site where people can create personal profiles (including passwords for authentication), update their profiles, and search for other registered users. Although this sample application has a limited feature set, it shows one way to create an authenticated Web site with profile storage.

Directory Use

Our application is named SimpleSite. It uses a directory service for the following tasks:

- **User profile creation.** When a new Web site visitor becomes a member of the SimpleSite Web site by registering his personal profile, a new entry is created in the directory.
- **User profile updates.** When an authenticated member updates his personal profile, a directory entry is modified.
- **Password-based authentication.** Members must log in to the Web application to access its features. The LDAP bind operation is used to verify each user's password.
- **Searching for other registered users.** The SimpleSite application provides a **Find** page that has an LDAP directory search back end.

The SimpleSite application uses the `inetOrgPerson` object class from RFC 2798 as the structural class for user profile entries. [Listing 21.15](#) shows the additional schema used. Although the listing shows placeholders for the OIDs (object identifiers), real OIDs should be used.

Listing 21.15 The SimpleSite Custom Schema

```
# Schema for SimpleSite LDAP Application Example

#
# attribute types:
#
dn: cn=schema

attributeTypes: ( simpleEmailFormat-oid
    NAME 'simpleEmailFormat'
    DESC 'preferred format for received messages, text or HTML'
    EQUALITY caseIgnoreMatch
    SYNTAX 1.3.6.1.4.1.1466.115.121.1.15
)
```

```

attributeTypes: ( simpleExpertise-oid
    NAME 'simpleExpertise'
    DESC 'areas of expertise'
    EQUALITY caseIgnoreMatch
    SYNTAX 1.3.6.1.4.1.1466.115.121.1.15
)
#
# object classes:
#
objectClasses: ( simpleSiteObject-oid
    NAME 'simpleSiteObject'
    DESC 'simple site information'
    SUP top
    AUXILIARY
    MAY ( co $ simpleEmailFormat $ simpleExpertise )
)

```

One auxiliary object class called `simpleSiteObject` is defined that includes two optional attribute types. This auxiliary class is added to all user entries. The `co` attribute holds the user's country. The `simpleEmailFormat` attribute holds a MIME type that specifies the user's preferred format for e-mail messages he receives, either `text/plain` (text format) or `text/html` (HTML format). This information could be used by the SimpleSite administrators when sending monthly newsletters or personalized e-mail messages.

The `simpleExpertise` attribute holds text that indicates the areas in which a registered user claims to have some expertise. For example, it could hold values such as `Java programming` or `sewing`. [Listing 21.16](#) shows a sample user profile entry.

Listing 21.16 A SimpleSite User Profile Entry

```

dn: mail=mcs@netscape.com,ou=Members,o=SimpleSite
objectClass: top
objectClass: person
objectClass: organizationalPerson
objectClass: inetOrgPerson
objectClass: simpleSiteObject

```

```
mail: mcs@netscape.com
userPassword: secret
cn: Mark Smith
sn: Smith
street: 100 S. Main Street
l: Ann Arbor
st: MI
postalCode: 48104
co: US
simpleEmailFormat: text/html
simpleExpertise: LDAP, cooking
```

The SimpleSite application uses a very simple DIT structure in which all registered users are placed under a configurable branch within a configurable top-level subtree. Entries are named by the `mail` attribute.

Tip

Although the SimpleSite application is a self-contained application and could use all custom schemas, it uses standard schema elements and one custom auxiliary class. This is a good example of how to extend LDAP schemas to support a specific application while maintaining compatibility with as many LDAP clients and servers as possible.

The User Experience

Users who interact with the SimpleSite application must first register a personal profile. [Figure 21.13](#) shows the initial page that visitors see.

Figure 21.13. The SimpleSite Login Page



Clicking on the **Register** button brings up a **Create New Profile** form. Most of the fields are optional, but an e-mail address and password are required. [Figure 21.14](#) shows a completed profile form that is about to be submitted.

Figure 21.14. The SimpleSite Create New Profile Page



Once a new visitor has registered or a returning member has logged in, he is given three options:

1. To edit his profile
2. To find other people
3. To log out

The logout function simply discards all identity information and returns the user to the **Login** page. [Figure 21.15](#) shows an example of the **Edit Profile** page. This page is straightforward in function.

Figure 21.15. The SimpleSite Edit Profile Page

Edit Profile for mvinfo@mountvernon.org - Netscape 6

File Edit View Search Go Bookmarks Tasks Help

http://simple/site/SimpleSiteServlet/editprofile

Search N

Edit Profile for mvinfo@mountvernon.org

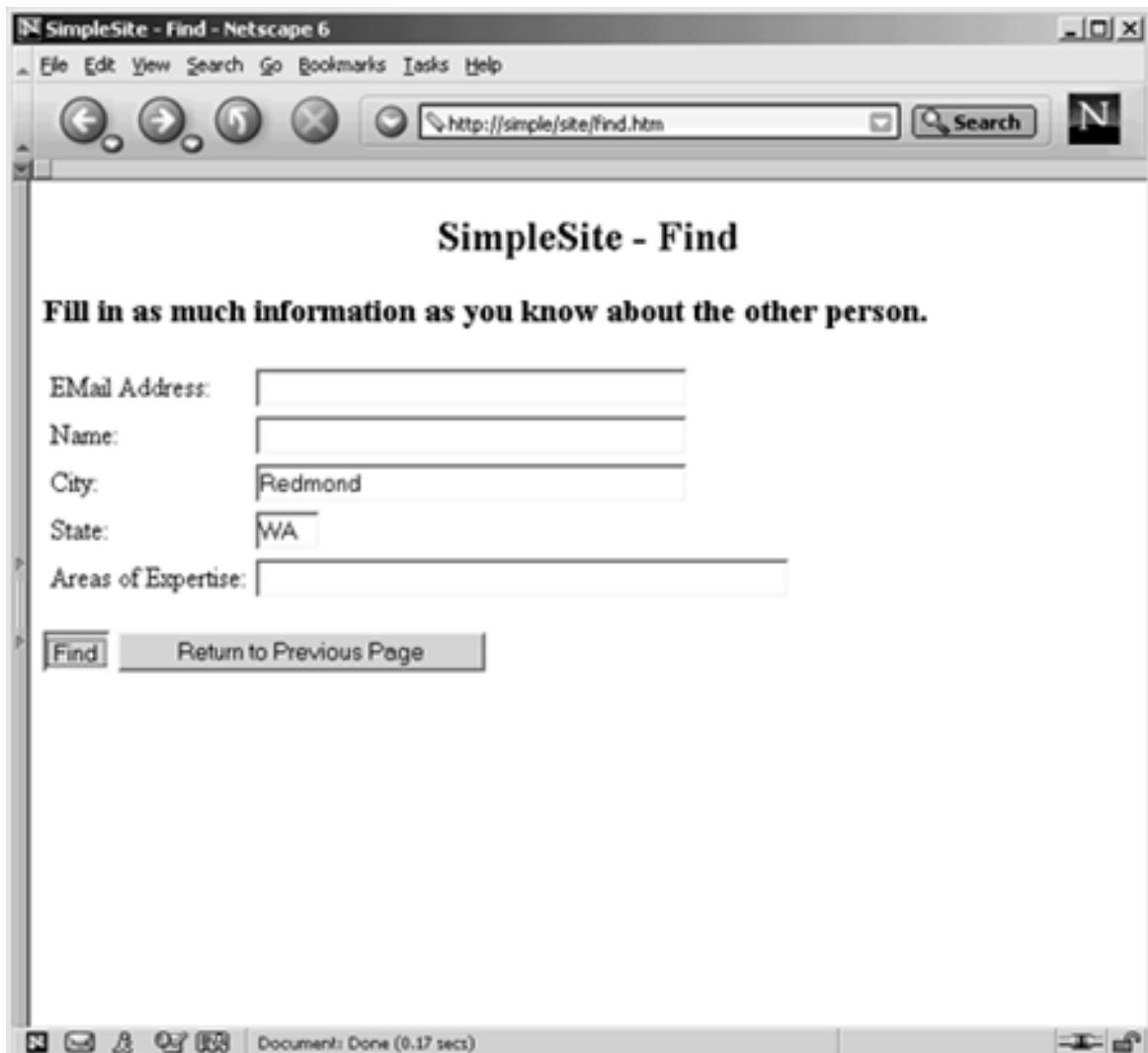
Name:	George Washington
Street Address:	Mt. Vernon Memorial Hwy
City:	Mount Vernon
State:	VA
Zip Code:	22121
Country:	US
Areas of Expertise:	farming,fathering countries
Preferred Email Format:	<input checked="" type="radio"/> Text <input type="radio"/> HTML

[Save](#) [Return to Previous Page](#)

Document: Done (0.2 secs)

The **Find** function is more interesting. A page containing a search form is presented, and the user is asked to fill in as many fields as he can. [Figure 21.16](#) shows a SimpleSite **Find** page in which the user has filled in a city ("Redmond") and a state code ("WA").

Figure 21.16. The SimpleSite Find Page



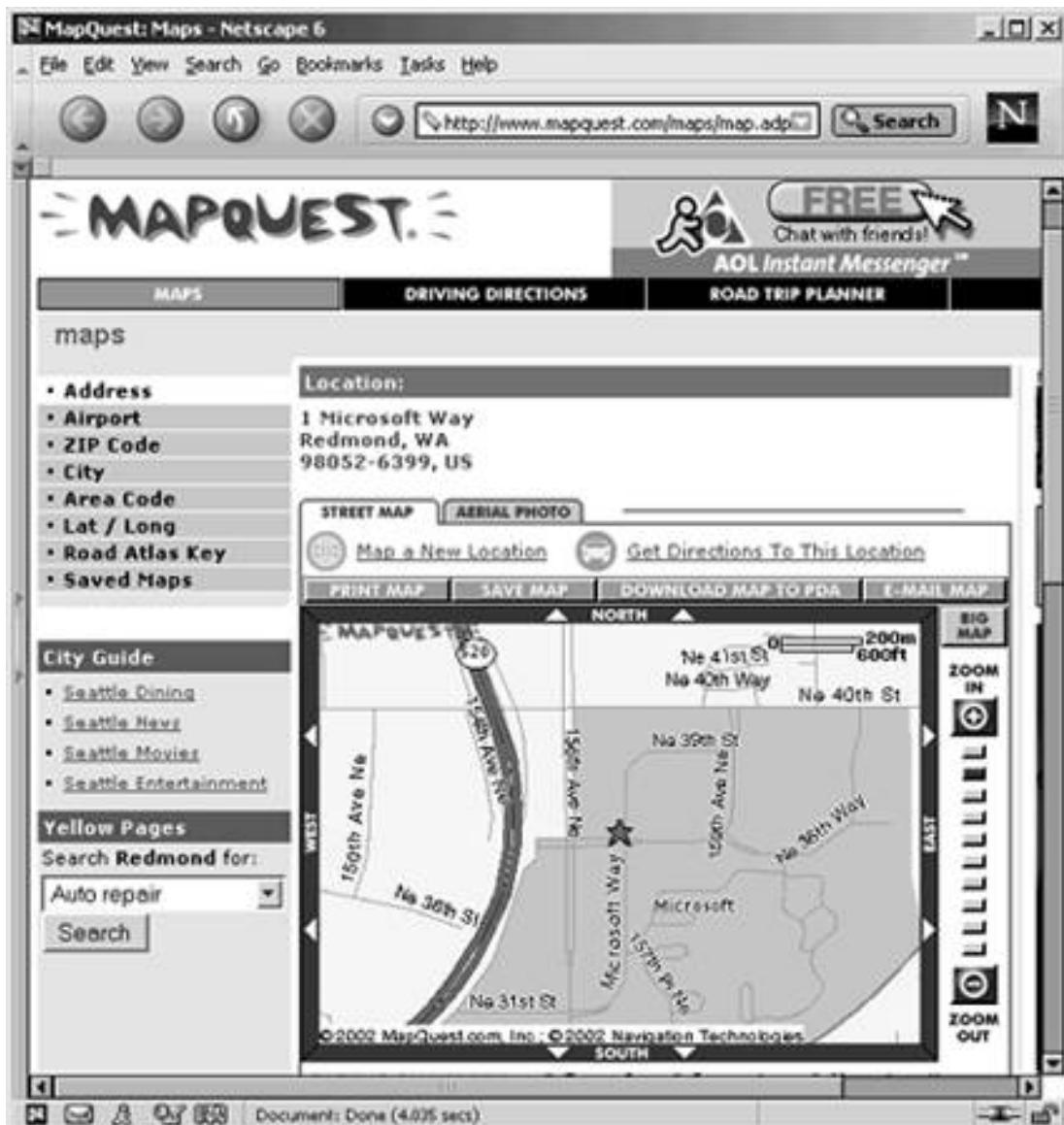
When the user clicks on the **Find** button, a directory search is performed and all matching entries are returned. An exact match is used for all fields except **Areas of Expertise**, where a substring match is used. [Figure 21.17](#) shows a sample **Search Results** page.

Figure 21.17. A SimpleSite Search Results Page



Below the information for the person or persons found, **Send Email** and **Display Map** links are provided. The **Send Email** link is a "mailto:" URL that typically causes a new e-mail window to be opened with the **To:** field already filled in. The **Display Map** link is an external "http:" link that points to the MapQuest map generation service. [Figure 21.18](#) shows the result of clicking on Mr. Gates's **Display Map** link.

Figure 21.18. The Result of Clicking on a SimpleSite Display Map Link



The location information stored in the user's profile is used to create the link to MapQuest.

The Source Code

The SimpleSite application is written as a Java servlet, and it uses JNDI to access an LDAP directory service. The code was compiled and tested within the Web application container on Netscape Enterprise Server 6 running on the Sun Solaris 8 Unix operating system. The code should run on any Web server that provides a standard servlet container. The source code for the SimpleSite application consists of two HTML files and one Java file. No JSP tags are used; all HTML is generated directly by the Java code or through static HTML files. The code is presented here in pieces to aid understanding.

[Listing 21.17](#) shows the `login.htm` file, which is the source for the page that is presented when someone first connects to the SimpleSite application (as shown in [Figure 21.13](#)). There are two HTML forms on this page: a "Login" form and a "Register" form. Both have an action that targets `SimpleSiteServlet` with extra path information used to specify a subcommand (`login` or `newprofile`). `SimpleSiteServlet` is the name of the servlet implemented by the Java code that will be described later.

Listing 21.17 The SimpleSite login.htm File

<html>

```
<head><title>SimpleSite - Login</title></head>

<body>

<div align="Center"><h2>SimpleSite - Login</h2></div>

<h3>Members, please log in.</h3>

<form action="SimpleSiteServlet/login" method="POST">

<table cellpadding="2" cellspacing="2" border="0"><tbody>

<tr>
    <td>EMail Address:</td>
    <td><input type="text" name="email" size="30"></td>
</tr>

<tr>
    <td>Password:</td>
    <td><input type="password" name="pwd" size="30"></td>
</tr>

</tbody></table>

<p>
    <input type="submit" value="Login">
</p>

<hr>

<h3>New visitors, please click the Register button.</h3>

<form action="SimpleSiteServlet/newprofile" method="GET">

<input type="submit" value="Register">
</form>

</body>

</html>
```

[Listing 21.18](#) shows the `find.htm` file, which provides the initial user interface for the SimpleSite **Find** page. Several input fields are provided (for example, **City**). The input field names are LDAP attribute names. The form action again points to `SimpleSiteServlet`, this time with a subcommand of `find`.

Listing 21.18 The SimpleSite `find.htm` File

```
<html>

<head><title>SimpleSite - Find</title></head>

<body>

<div align="Center"><h2>SimpleSite - Find</h2></div>

<h3>Fill in as much information as you know about the other person.</h3>

<form action="SimpleSiteServlet/find" method="POST">

<table cellpadding="2" cellspacing="2" border="0"><tbody>

<tr>
    <td>EMail Address:</td>
    <td><input type="text" name="mail" size="40"></td>
</tr>

<tr>
    <td>Name:</td>
    <td><input type="text" name="cn" size="40"></td>
</tr>

<tr>
    <td>City:</td>
    <td><input type="text" name="L" size="40"></td>
</tr>

<tr>
    <td>State:</td>
    <td><input type="text" name="st" size="4"></td>
</tr>
```

```

<tr>
    <td>Areas of Expertise:</td>
    <td><input type="text" name="simpleExpertise" size="50"></td>
</tr>

</tbody></table>

<p>
<input type="submit" value="Find">
<input type="button" value="Return to Previous Page"
onClick="window.history.back()">
</form>

</body>
</html>

```

The servlet source code is in a file named `SimpleSiteServlet.java`. Listing 21.19 shows the beginning of this file.

Listing 21.19 The First Part of `SimpleSiteServlet.java`

```

1. /*
2. * SimpleSite Servlet LDAP Application Example.
3. *
4. * From the 2nd Edition of the book:
5. *      "Understanding and Deploying LDAP Directory Services"
6. *      by Timothy A. Howes, Mark C. Smith, and Gordon S. Good.
7. */
8. import java.io.*;
9. import java.util.*;
10. import javax.servlet.*;
11. import javax.servlet.http.*;
12. import javax.naming.*;
13. import javax.naming.directory.*;

```

```

14.

15. public class SimpleSiteServlet extends HttpServlet

16. {

17. /*
18. * Data structures.
19. */

20. private class fieldmap {

21.     boolean      mNewProfileOnly;
22.     String       mHtmlFieldName;
23.     int          mHtmlFieldSize;
24.     String       mLdapAttrName;
25.     String       mMapQuestFieldName;

26.

27.     // fieldmap constructor

28.     private fieldmap( boolean newProfileOnly, String htmlFieldName,
29.                         int htmlFieldSize, String ldapAttrName,
30.                         String mapQuestFieldName ) {
31.
32.         mNewProfileOnly = newProfileOnly;
33.
34.         mHtmlFieldName = htmlFieldName;
35.
36.         mHtmlFieldSize = htmlFieldSize;
37.         mLdapAttrName = ldapAttrName;
38.         mMapQuestFieldName = mapQuestFieldName;
39.     }
40. }

41. }

42.
```

The Java packages used are imported by the code on lines 8 to 13. The packages that begin with "javax.naming" are part of JNDI. The `SimpleSiteServlet` class is declared on line 15; it is derived from the standard Java `HttpServlet` class. The remainder of the code (lines 20–37) defines a private class named `fieldmap` that is able to store information for one user-visible data field. The `fieldmap` class is a simple data structure (it has only one method, a constructor). The `fieldmap` member variables are

- **mNewProfileOnly**. A Boolean that indicates whether this data field should be displayed only on the **New Profile** page.

- **mHtmlFieldName**. A human-readable name for the field that is used to label HTML form elements.
- **mHtmlFieldSize**. The HTML field size that is used to construct the `size=` attribute of an HTML input form element.
- **mLdapAttrName**. The LDAP attribute type for the data field.
- **mMapQuestFieldName**. The label used for this field within a MapQuest **Show Map** URL. If `null`, the field is not used when constructing a **Show Map** URL.

[Listing 21.20](#) shows the second portion of the Java source file. The code on lines 42 to 48 defines some `SimpleSiteServlet` member variables, most of which are set from context initialization properties that appear in an external Web application file (the code that does that is shown later, in [Listing 21.21](#)).

Listing 21.20 The Second Part of `SimpleSiteServlet.java`

```

39. /*
40. * Private data.
41. */
42. private String mApplName      = "SimpleSite";
43. private String mMQURLPrefix = "http://www.mapquest.com/maps/map.adp?";
44. private String mLdapURL;      // URL for the LDAP server
45. private String mBaseDN;       // from "ldapBase" property
46. private String mAddLocation; // from "ldapAddLocation" property
47. private String mAdder;        // from "ldapAdder" property
48. private String mAdderPwd;     // from "ldapAdderPwd" property
49.
50. private fieldmap[] mProfileFields = {
51.   new fieldmap( true,   "Email Address",  50, "mail",           null ),
52.   new fieldmap( true,   "Password",       30, "userPassword",    null ),
53.   new fieldmap( false,  "Name",          30, "cn",            null ),
54.   new fieldmap( false,  "Street Address", 40, "street",         "address" ),
55.   new fieldmap( false,  "City",           40, "L",              "city" ),
56.   new fieldmap( false,  "State",          4,  "st",             "state" ),
57.   new fieldmap( false,  "Zip Code",       12, "postalCode",     "zipcode" ),

```

```

58.     new fieldmap( false, "Country",           4, "co",          "country" ) ,
59.     new fieldmap( false, "Areas of Expertise",      50, "simpleExpertise", null ),
60.                                         0, "simpleEmailFormat",null )
61.     new fieldmap( false, "Preferred Email Format",
62.                               0, "simpleEmailFormat",null )
63.   );
64.
65. private String[] mLdapAttrsToRetrieve;
66.
67. private String[] mObjectClassValues = {
68.     "inetOrgPerson",
69.     "simpleSiteObject"
70. };
71.

```

The code on lines 50 to 63 creates a member variable named `mProfileFields` and initializes it with an array of `fieldmap` objects. For example, on line 55 a `fieldmap` object is created that has an HTML input field name of `City`, an HTML field size of `40` characters, an LDAP attribute name of `L` (locality), and a MapQuest URL field name of `city`.

The code on lines 65 to 70 defines two additional member variables. The `mLdapAttrsToRetrieve` variable is an array of LDAP attributes to retrieve when searching for entries (the array is created by code we will look at later). The `mObjectClassValues` variable is a statically initialized array that includes the two LDAP object class values that must be included in all new entries (`inetOrgPerson` and `simpleSiteObject`).

The `SimpleSiteServlet` class has only three public methods:

1. `init()`. Called once when the servlet is loaded.
2. `doGet()`. Called when an HTTP client (typically a Web browser) submits an HTTP `GET` request that targets the servlet.
3. `doPost()`. Called when an HTTP client submits an HTTP `POST` request that targets the servlet.

The Java servlet container (typically part of a Web server or an application server) ensures that these public methods are called. [Listing 21.21](#) shows the `init()` method.

Listing 21.21 The `SimpleSiteServlet init()` Method

```

72.
73. /*
74.  * Public servlet methods.
75. */
76. /*
77.  * init(): Perform essential initialization
78. */
79. public void init(
80.         ServletConfig config )
81.         throws ServletException {
82.     super.init( config );
83.
84.     // initialize our base DN, member RDN, and LDAP URL
85.     mBaseDN = getServletContext().getInitParameter( "ldapBase" );
86.     mAddLocation = getServletContext().getInitParameter(
87.             "ldapAddLocation" );
88.     mLdapURL = "ldap://" +
89.             + getServletContext().getInitParameter( "ldapServer" )
90.             + "/" + mBaseDN;
91.
92.     // initialize the DN and password we use to create new entries
93.     mAdder = getServletContext().getInitParameter( "ldapAdder" );
94.     mAdderPwd = getServletContext().getInitParameter( "ldapAdderPwd" );
95.
96.     // create a list of LDAP attributes we will retrieve
97.     mLdapAttrsToRetrieve = new String[ mProfileFields.length ];
98.     for ( int i = 0; i < mProfileFields.length; ++i ) {
99.         mLdapAttrsToRetrieve[i] = mProfileFields[i].mLdapAttrName;
100.    }
101. }
```

The `init()` code calls its parent's `init()` method on line 82, as required by the servlet

specification. The code on lines 84 to 94 retrieves a set of context initialization properties from the servlet container and saves them in member variables. For example, line 85 retrieves the `ldapBase` property and stores it in a member variable named `mBaseDN`. Later this value is used as the base for LDAP searches, and the `ldapAddLocation` property is prepended to the `mBaseDN` value to determine the parent DN for new entries. The code on lines 96 to 100 initializes the `mLdapAttrsToRetrieve` array by copying the LDAP attribute names from the `mProfileFields` `fieldmap` array.

Tip

The SimpleSite application retrieves a series of LDAP configuration parameters from the servlet container. The LDAP server, port, base DN for searches, DN and password used in the addition of entries, and the location where new entries are added can all be changed to match a specific directory deployment or test bed. Typically, servlet parameters like these are stored in an XML configuration file, which means that the parameters can be modified without the SimpleSite code having to be recompiled. You should use similar techniques in your own LDAP applications to avoid hard-coding knowledge of a particular directory deployment (or server location within a deployment) in your code.

[Listing 21.22](#) shows the `doGet()` method, which is called by the servlet container in response to an HTTP `GET` request. The `doGet()` method examines the extra path information from the request URL and calls one of these three methods to handle the request:

1. `doNewProfile()`. Displays a **Create New Profile** form, as shown in [Figure 21.14](#).
2. `doEditProfile()`. Displays the **Edit Profile** form with the current user's profile data filled in. [Figure 21.15](#) shows an example of such a form.
3. `doLogout()`. Clears all authentication and identity information and redirects the browser to the **Login** page.

The source code for each of these methods is shown later, in [Listings 21.28](#) and [21.32](#).

Listing 21.22 The `SimpleSiteServlet` `doGet()` Method

```
102.  
103.      /*  
104.      * doGet(): handle an HTTP GET request  
105.      */  
106.      public void doGet(  
107.                  HttpServletRequest request,  
108.                  HttpServletResponse response )  
109.                  throws IOException, ServletException {  
110.
```

```

111.     // prepare for response generation and dispatch to handler method
112.     response.setContentType( "text/html" );
113.     PrintWriter writer = response.getWriter();
114.
115.     String operation = request.getPathInfo();
116.     if ( operation == null ) {
117.         operation = "";
118.     } else {
119.         operation = operation.substring( 1 );
120.     }
121.
122.     if ( operation.equals( "editprofile" ) ) {
123.         doEditProfile( request, response, writer );
124.     } else if ( operation.equals( "newprofile" ) ) {
125.         doNewProfile( request, response, writer );
126.     } else if ( operation.equals( "logout" ) ) {
127.         doLogout( request, response, writer );
128.     } else {
129.         unknownRequest( operation, "HTTP GET", writer );
130.     }
131. }
```

[Listing 21.23](#) shows the `doPost()` method, which is called by the servlet container in response to an HTTP `POST` request.

Listing 21.23 The `SimpleSiteServlet doPost()` Method

```

132.
133.     /*
134.      * doPost(): handle an HTTP POST request
135.     */
136.     public void doPost(
137.         HttpServletRequest request,
```

```

138.             HttpServletRequest response )
139.         throws IOException, ServletException {
140.
141.         // prepare for response generation and dispatch to handler method
142.         response.setContentType( "text/html" );
143.         PrintWriter writer = response.getWriter();
144.
145.         String operation = request.getPathInfo();
146.         if ( operation == null ) {
147.             operation = "";
148.         } else {
149.             operation = operation.substring( 1 );
150.         }
151.
152.         if ( operation.equals( "login" ) ) {
153.             doLogin( request, response, writer );
154.         } else if ( operation.equals( "find" ) ) {
155.             doFind( request, response, writer );
156.         } else if ( operation.equals( "saveprofile" ) ) {
157.             doSaveProfile( request, response, writer );
158.         } else {
159.             unknownRequest( operation, "HTTP POST", writer );
160.         }
161.     }

```

Similar to how `doGet()` works, the `doPost()` method examines the extra path information from the request URL and calls one of these three methods to handle the request:

1. `doLogin()`. Processes the **Login** form and authenticates the user against the directory service.
2. `doFind()`. Processes the **Find** form, searches the directory, and displays results like those shown in [Figure 21.17](#).
3. `doSaveProfile()`. Processes input from the **Create New Profile** and **Edit Profile**

forms, performing an LDAP add or modify operation as appropriate.

The source code for the `doFind()` and `doSaveProfile()` methods is shown later, in [Listings 21.30](#), [21.34](#), and [21.35](#). [Listing 21.24](#) shows the `doLogin()` method.

Listing 21.24 The `SimpleSiteServlet` `doLogin()` Method

```
162.  
163. /*  
164. * Operation handlers.  
165. */  
166. /*  
167. * doLogin(): handle a "login" HTTP POST sub-request  
168. */  
169. private void doLogin(  
170.           HttpServletRequest request,  
171.           HttpServletResponse response,  
172.           PrintWriter writer) {  
173.  
174.     // output the page header  
175.     writePageHeader(mApplName + " - Login", writer);  
176.  
177.     // retrieve the HTTP session  
178.     HttpSession httpsession = request.getSession();  
179.  
180.     // retrieve the login form variables  
181.     String email = request.getParameter("email");  
182.     if (email == null || email.length() == 0) {  
183.         reportError("Login: please enter your email address",  
184.                     null, writer);  
185.         return;  
186.     }  
187.     String pwd = request.getParameter("pwd");  
188.     if (pwd == null || pwd.length() == 0) {
```

```
189.         reportError( "Login: please enter your password", null, writer );
190.
191.     }
192.
193.     clearSessionValues( httpsession );
194.
195.     try {
196.
197.         DirContext ctx = createLDAPContext( null );
198.
199.         // map the e-mail address to a DN
200.         String partialDN = email2LDAPDN( ctx, email, writer );
201.
202.         ctx.close();
203.
204.         if ( partialDN != null ) {
205.
206.             // try to authenticate
207.             ctx = createLDAPContext( partialDN, pwd );
208.
209.             // store user information in the HTTP session
210.             httpsession.setAttribute( "id", email );
211.             httpsession.setAttribute( "partialDN", partialDN );
212.             httpsession.setAttribute( "pwd", pwd );
213.
214.             // finish the page and close the LDAP connection
215.             writePageFooter( "<h3>Authentication succeeded.</h3>" ,
216.                             true, writer );
217.
218.             ctx.close();
219.         }
220.     } catch ( Exception e ) {
221.         reportError( "Login:", e, writer );
222.     }
223. }
```

Line 175 calls the `writePageHeader()` utility method to generate the beginning of an HTML page, including a title (the code for `writePageHeader()` is shown later, in [Listing 21.36](#)). The code on line 178 retrieves the current HTTP session, creating one if necessary. The standard servlet infrastructure provides a shared HTTP session that can be used to store arbitrary session attributes. The HTTP session is typically implemented through HTTP cookies or through URL rewriting, although servlet writers do not need to worry about the details.

The `SimpleSiteServlet()` method uses three session attributes:

1. **id**. The e-mail address of the logged-in user—for example, `mcs@netscape.com`.
2. **partialDN**. The partial LDAP DN (relative to the `mBaseDN` variable) of the logged-in user—for example, `mail=mcs@netscape.com,ou=Members`.
3. **pwd**. The LDAP password of the logged-in user—for example, `secret`.

The `doLogin()` method sets the session attributes (lines 206–209). Other methods, such as `doSaveProfile()`, retrieve the session attributes and use them to identify the user and to authenticate to the directory service.

The code on lines 195 to 218 handles the LDAP-based authentication. Line 196 includes a call to a `createLDAPContext()` utility method, whose job is to create a JNDI LDAP directory context (the code for `createLDAPContext()` is shown next, in [Listing 21.25](#)). Line 199 calls a utility method named `email2LDAPDN()` to find the partial LDAP DN (again, relative to the `mBaseDN` variable) that matches the e-mail address entered on the **Login** form. The code for `email2LDAPDN()` is shown later. Line 204 tries to create an authenticated JNDI directory context, and if it is successful, the session attributes are set and a success message is sent to the user. Errors are displayed via the `reportError()` utility method (shown later, in [Listing 21.37](#)).

[Listing 21.25](#) shows two `createLDAPContext()` utility methods (this is an example of Java method overloading, in which two methods have the same name but different parameter lists). These methods demonstrate how to create a JNDI LDAP directory context, optionally with simple (password-based) authentication. [Listing 21.25](#) shows methods that are located near the end of the `SimpleSiteServlet.java` source file; that's why the line numbers do not begin where [Listing 21.24](#) left off (the `SimpleSiteServlet` methods are presented out of order to make it easier to follow the flow between methods).

Listing 21.25 The `SimpleSiteServlet` `createLDAPContext()` Methods

```
585.  
586. /*  
587. * LDAP utility methods.  
588. */  
589. /*  
590. * createLDAPContext(): create an initial directory
```

```
591.      *      context (open the LDAP connection). If partialDN is
592.      *      not null, simple authentication is done.
593.      */
594.  private DirContext createLDAPContext(
595.      String      partialDN, // can be null
596.      String      pwd )
597.      throws NamingException {
598.
599.      // initialize basic environment for JNDI's LDAP provider
600.      Hashtable env = new Hashtable();
601.      env.put( Context.INITIAL_CONTEXT_FACTORY,
602.                  "com.sun.jndi.ldap.LdapCtxFactory" );
603.      env.put( Context.PROVIDER_URL, mLdapURL );
604.      env.put( Context.REFERRAL, "follow" );
605.
606.      // handle optional simple bind
607.      if ( partialDN != null && partialDN.length() > 0 ) {
608.          String dn = partialDN + "," + mBaseDN;
609.          env.put( Context.SECURITY_AUTHENTICATION, "simple" );
610.          env.put( Context.SECURITY_PRINCIPAL, dn );
611.          env.put( Context.SECURITY_CREDENTIALS, pwd );
612.      }
613.      return new InitialDirContext( env );
614.  }
615.
616.  /*
617.   * createLDAPContext(): create an initial directory context based
618.   * on information found in an HTTP session.
619.   */
620.  private DirContext createLDAPContext(
621.      HttpSession httpsession ) // can be null
622.      throws NamingException {
```

```

623.

624.         String dn = null, pwd = null;

625.         if ( httpSession != null ) {

626.             dn = (String)httpSession.getAttribute( "partialDN" );
627.             pwd = (String)httpSession.getAttribute( "pwd" );
628.         }

629.         return createLDAPContext( dn, pwd );
630.     }

```

The first method takes two `String` parameters: `partialDN` and `pwd`. The second method takes one parameter: `httpSession`. Both methods create a JNDI LDAP directory context, which is essentially a connection to an LDAP server, along with some associated state information that is used by JNDI. The second method extracts the partial DN and password from the HTTP session and simply calls the first method.

JNDI uses a hash table to pass directory-specific parameters into the `InitialDirContext` constructor. The code on line 600 creates a hash table named `env` (short for environment). Sun's LDAP provider is selected by lines 601 and 602 (JNDI allows other LDAP and non-LDAP providers to be used; for example, Netscape includes an LDAP provider in its LDAP Java SDK). The code on line 603 adds to the hash table an LDAP URL containing the server, port, and base DN, and line 604 enables automatic following of LDAPv3 referrals.

If the `partialDN` and password (`pwd`) are not `null`, an authenticated LDAP connection is requested. The code on lines 606 to 612 adds the necessary information to the `env` hash table. Finally, the code on line 613 calls the `InitialDirContext` constructor to establish the LDAP connection and perform authentication (if requested).

[Listing 21.26](#) shows the `email2LDAPDN()` utility method that is called from `doLogin()` to return a partial LDAP DN, given an e-mail address. The `email2LDAPDN()` code demonstrates how to perform an LDAP search using JNDI. An LDAP directory context (`DirContext`) must be passed to `email2LDAPDN()`, along with an e-mail address `String` and a `PrintWriter` that is associated with the HTML response (used for reporting errors).

Listing 21.26 The `SimpleSiteServlet` `email2LDAPDN()` Method

```

631.

632.         /*
633.          * email2LDAPDN(): Find a person entry based on an e-mail address.
634.          *      Return a partial DN that is context-relative.
635.         */
636.         private String email2LDAPDN(

```

```
637.         DirContext ctx,
638.
639.         String email,
640.
641.         PrintWriter writer ) {
642.
643.             String partialDN = null;
644.
645.             // construct the search filter and search controls
646.             String filter = "(&(objectClass=person)(mail=" +
647.                           escapedValue( email ) + " ))";
648.             String[] attrlist = { "1.1" }; // all we need is the DN
649.             SearchControls ctrls = new SearchControls(
650.                 SearchControls.SUBTREE_SCOPE,
651.                 100, // size limit
652.                 1000 * 15, // 15s time limit
653.                 attrlist, // attrs to retrieve
654.                 false, // return entire object
655.                 false // dereference aliases
656.             );
657.
658.             // do the search
659.             try {
660.                 NamingEnumeration answer = ctx.search( "", filter, ctrls );
661.
662.                 int count = 0;
663.
664.                 while ( answer.hasMore() ) {
665.                     SearchResult sr = (SearchResult)answer.next();
666.
667.                     if ( count == 0 ) {
668.                         partialDN = sr.getName();
669.                     }
670.                     ++count;
671.                 }
672.             }
```

```

669.         if ( count == 0 ) {
670.             reportError( "No matches for " + email, null, writer );
671.             partialDN = null;
672.         } else if ( count > 1 ) {
673.             reportError( count + " matches for " + email, null, writer );
674.             partialDN = null;
675.         }
676.     } catch ( Exception e ) {
677.         reportError( "Email lookup :", e, writer );
678.     }
679.
680.     return partialDN;
681. }

```

An LDAP search filter of the form `(&(objectClass=person)(mail=ESCAPED-VALUE))` is created by the code on lines 644 and 645. The `escapedValue()` utility method (shown next, in [Listing 21.27](#)) is used to escape all special LDAP filter characters within the search string. The code on lines 646–654 creates a JNDI `SearchControls` object that includes information such as LDAP size and time limits and a list of attributes to retrieve. Because only the DN is needed by the `email2LDAPDN()` method, the special string `"1.1"` is the only element in the `attrlist` (indicating that no attributes should be returned).

[Listing 21.27](#) shows the `escapedValue()` utility method. This method works by making a copy of the value that is passed in, escaping the appropriate characters by replacing each one with a backslash (\) followed by the two-digit hexadecimal representation of the character. The `specialChars` local variable that is defined on line 689 contains the characters that are escaped —namely, *, (,), and \. This set of characters comes from Section 4 of RFC 2254.

Listing 21.27 The `SimpleSiteServlet` `escapedValue()` Method

```

682.
683.     /*
684.      * escapedValue(): produce a copy of a string value, but with
685.      *     special LDAP filter characters escaped as \HH (hex value).
686.      */
687.     private String escapedValue( String value ) {
688.         StringBuffer escapedBuf = new StringBuffer();

```

```

689.         String specialChars = "*()\\\";
690.         char c;
691.
692.         for ( int i = 0; i < value.length(); ++i ) {
693.             c = value.charAt( i );
694.             if ( specialChars.indexOf( c ) >= 0 ) { // escape it
695.                 escapedBuf.append( '\\\\' );
696.                 String hexString = Integer.toHexString( c );
697.                 if ( hexString.length() < 2 ) {
698.                     escapedBuf.append( '0' );
699.                 }
700.                 escapedBuf.append( hexString );
701.             } else {
702.                 escapedBuf.append( c );
703.             }
704.         }
705.         return escapedBuf.toString();
706.     }

```

Now that some of the LDAP-related utility methods have been shown, we return to our discussion of the methods that process HTTP `GET` and `POST` requests. [Listing 21.28](#) shows the `doLogout()` method. This method is very simple. The HTTP session information is cleared by the call on line 231. Then a utility method named `getIDWithRedirect()` is called to send an HTTP redirect to the browser to return the user to the SimpleSite login page. The `getIDWithRedirect()` method attempts to retrieve the `id` session attribute, and sends a redirect if `id` can't be retrieved. Because the `doLogout()` method clears all of the session attributes before calling `getIDWithRedirect()`, a redirect is always issued in this case.

Listing 21.28 The `SimpleSiteServlet doLogout()` Method

```

220.
221.     /*
222.      * doLogout(): handle a "logout" HTTP GET subrequest
223.      */
224.     private void doLogout(
225.         HttpServletRequest request,

```

```

226.         HttpServletResponse response,
227.         PrintWriter writer ) {
228.
229.         // retrieve the HTTP session and clear all values
230.         HttpSession httpsession = request.getSession();
231.         clearSessionValues( httpsession );
232.
233.         // since the session info is gone, this will always redirect
234.         String id = getIDWithRedirect( httpsession, response, writer );
235.     }

```

[Listing 21.29](#) shows the `getIDWithRedirect()` and `clearSessionValues()` methods. These two methods are straightforward, using the `HttpSession.getAttribute()` method (line 765), the `setAttribute()` method (lines 784–786), and the `HttpServletResponse.sendRedirect()` method (line 769).

Listing 21.29 The `SimpleSiteServlet` `getIDWithRedirect()` and `clearSessionValues()` Methods

```

752.
753. /*
754. * General utility methods.
755. */
756. /*
757. * getIDWithRedirect(): Get the user's id from the HTTP session,
758. *      redirecting to the login page if not present.
759. */
760. private String getIDWithRedirect(
761.         HttpSession httpsession,
762.         HttpServletResponse response,
763.         PrintWriter writer ) {
764.
765.         String id = (String)httpsession.getAttribute( "id" );
766.

```

```

767.         if ( id == null || id.length() == 0 ) {
768.             try {
769.                 response.sendRedirect( "login.htm" );
770.             } catch ( Exception e ) {
771.                 reportError( "Redirect to login:", e, writer );
772.             }
773.             id = null;
774.         }
775.
776.         return id;
777.     }
778.
779.     /*
780.      * clearSessionValues(): clear old information that may be
781.      *       in the HTTP session.
782.      */
783.     private void clearSessionValues( HttpSession httpsession ) {
784.         httpsession.setAttribute( "id", "" );
785.         httpsession.setAttribute( "partialDN", "" );
786.         httpsession.setAttribute( "pwd", "" );
787.     }

```

Next we examine the `doFind()` method that is called when an HTTP `POST` request is received with extra path information of `find`. Listing 21.30 shows the `doFind()` implementation.

Listing 21.30 The `SimpleSiteServlet` `doFind()` Method

```

236.
237.     /*
238.      * doFind(): handle a "find" HTTP POST subrequest
239.      */
240.     private void doFind( // HTTP POST method
241.                         HttpServletRequest request,

```

```
242.         HttpServletResponse response,
243.         PrintWriter writer ) {
244.
245.         // retrieve the HTTP session
246.         HttpSession httpsession = request.getSession();
247.
248.         // generate start of page
249.         String id = getIDWithRedirect( httpsession, response, writer );
250.         if ( id == null ) return;
251.         writePageHeader( mApplName + " - Search Results", writer );
252.
253.         // Retrieve form variables and create an ANDed search filter.
254.         // For each value, we create an equality filter component,
255.         // except for "Areas of Expertise" where a substring
256.         // component is used if 3 characters or more were entered.
257.         StringBuffer filter = new StringBuffer();
258.         Enumeration params = request.getParameterNames();
259.         while ( params.hasMoreElements() ) {
260.             String paramName = (String)params.nextElement();
261.             String paramVal = request.getParameter( paramName );
262.             if ( paramVal != null && paramVal.length() > 0 ) {
263.                 if ( paramName.equalsIgnoreCase( "simpleExpertise" )
264.                     && paramVal.length() >= 3 ) {
265.                     filter.append( '(' + paramName + "=*" +
266.                         escapedValue( paramVal ) + "*)" );
267.                 } else {
268.                     filter.append( '(' + paramName + '=' +
269.                         escapedValue( paramVal ) + ')' );
270.                 }
271.             }
272.         }
```

```
273.     if ( filter.length() == 0 ) {
274.         reportError( "Find: please provide some information",
275.                     null, writer );
276.         return;
277.     }
278.
279.     // restrict our search to person entries only
280.     filter.insert( 0, "(objectClass=person)" );
281.     filter.append( ')' );
282.
283.     // search
284.     try {
285.         DirContext ctx = createLDAPContext( httpsession );
286.
287.         SearchControls ctrls = new SearchControls(
288.             SearchControls.SUBTREE_SCOPE,
289.             10,                                // size limit
290.             1000 * 15,                          // 15s time limit
291.             mLdapAttrsToRetrieve,               // attr list
292.             false,                             // return entire object
293.             false                               // dereference aliases
294.         );
295.
296.         NamingEnumeration answer = ctx.search( "", filter.toString(), ctrls );
297.
298.         int count = 0;
299.         SearchResult sr = null;
300.
301.         while ( answer.hasMore() ) {
302.             sr = (SearchResult)answer.next();
303.             displayOneEntry( sr.getAttributes(), writer );
304.             ++count;
305.         }
306.     } catch ( Exception e ) {
307.         reportError( "Find: " + e.getMessage(),
308.                     null, writer );
309.     }
310. 
```

```

305.

306.         if ( count == 0 ) {

307.             reportError( "No matches", null, writer );

308.         } else if ( count > 1 ) {

309.             writer.println( "<h3>" + count + " people found.</h3>" );

310.         }

311.

312.         writePageFooter( null, true, writer );

313.

314.     } catch ( Exception e ) {

315.         reportError( "Find:", e, writer );

316.     }

317. }

```

On line 249, the `getIDWithRedirect()` method is called to ensure that the user is authenticated (if not, no `id` attribute is present in the HTTP session and the user is redirected to the login page). The code on lines 253 to 281 constructs an LDAP search filter based on the form variables that were part of the HTTP `POST` request. The search filter consists of a series of ANDed components, where each component is a simple equality filter. The one exception is that an inner substring filter component of the form (`simpleExpertise= *ESCAPED-VALUE*`) is used for the `simpleExpertise` LDAP attribute.

This approach for constructing the search filter relies on the fact that the HTML input field names are LDAP attribute names, as [Listing 21.18](#) showed. The complete filter is of the following form: `(&(objectClass=person)(ATTR1=ESCAPED-VALUE1)(ATTR2= ESCAPED-VALUE2)...)`. For example, a search for people who live in the state of Washington and who claim to have expertise in software would look like this: `(&(objectClass=person)(st=WA)(simpleExpertise=*software*))`.

Tip

The SimpleSiteServlet `doFind()` method avoids creating LDAP search filters that include short substring filter components, and it avoids approximate filter components entirely. Why? Because search filters that contain short substrings and approximate components tend to put a greater load on directory servers and cause slow searches. The best search filter to use depends on what kind of application you're creating and the preferences of the users of the application, but use simple, efficient filters whenever possible.

The remaining code in the `doFind()` method issues an LDAP search operation and processes the results. This code is very similar to that found in the `email2LDAPDN()` method examined earlier

(see [Listing 21.26](#)), except that all of the attributes used by the SimpleSite application are requested (line 291), and each entry returned is displayed by a call to a utility method called `displayOneEntry()`. JNDI returns search results as a `NamingEnumeration`, and the code on lines 298 to 304 steps through the enumeration, using the `getAttributes()` method to retrieve the LDAP attributes, which are then passed to `displayOneEntry()`.

[Listing 21.31](#) shows the `displayOneEntry()` method, which generates an HTML table with the human-readable field names in the first column and the user profile values in the second column. This method demonstrates how to retrieve a specific LDAP attribute from a JNDI `Attributes` object. The heart of the `displayOneEntry()` method is the `for` loop that starts on line 718. It loops over the elements of the `mProfileFields` fieldmap array, retrieving LDAP attribute values and emitting an HTML table row for each element. The code on lines 719 to 723 uses the `Attributes get()` method to retrieve an attribute by name, and the code on line 724 retrieves a single `String` value.

Listing 21.31 The SimpleSiteServlet displayOneEntry() Method

```

728.                         " : </b></td>\n<td>" + val + "</td></tr>" );
729.
730.         if ( mProfileFields[i].mMapQuestFieldName != null ) {
731.             mqURL = mqURL + "&" +
732.                     mProfileFields[i].mMapQuestFieldName +
733.                     "= " + val;
734.     }
735.
736.     if ( mProfileFields[i].mLdapAttrName.equalsIgnoreCase(
737.             "mail" ) ) {
738.         email = val;
739.     }
740. }
741. }
742. writer.println( "</tbody></table>\n" );
743.
744. if ( email != null ) {
745.     writer.println( "<a href=\"mailto:" + email +
746.                     "\">Send Email</a>&nbsp;&nbsp;&nbsp;" );
747. }
748. writer.println( "<a href=\"" + mqURL + "\" target=_new>" +
749.                     "Display Map</a><p>" );
750. }
751.

```

After the `for` loop is complete, two HTML links are added to the page being generated: a **Send Email** "mailto:" link based on the `mail` attribute and a **Display Map** "http:" link to MapQuest's site. The MapQuest link is constructed from the attributes that have a non-`null` `mMapQuestFieldName` member variable in their `fieldmap` object.

[Listing 21.32](#) shows the code for the two remaining HTTP `GET` request handler methods: `doNewProfile()` and `doEditProfile()`. Both of these methods call a utility method named `emitProfileForm()` (shown in [Listing 21.33](#)) that writes out an HTML form that has input fields for the user's profile information.

Listing 21.32 The SimpleSiteServlet doNewProfile() and doEditProfile() Methods

```
318.  
319.      /*  
320.      * doNewProfile(): handle a "newprofile" HTTP GET subrequest  
321.      */  
322.      private void doNewProfile(  
323.          HttpServletRequest request,  
324.          HttpServletResponse response,  
325.          PrintWriter writer ) {  
326.  
327.          clearSessionValues( request.getSession() );  
328.          writePageHeader( mApplName + " - Create New Profile", writer );  
329.          try {  
330.              emitProfileForm( null, writer );  
331.          } catch ( Exception e ) {  
332.              reportError( "New profile:", e, writer );  
333.          }  
334.      }  
335.  
336.      /*  
337.      * doEditProfile(): handle an "editprofile" HTTP GET subrequest  
338.      */  
339.      private void doEditProfile(  
340.          HttpServletRequest request,  
341.          HttpServletResponse response,  
342.          PrintWriter writer ) {  
343.  
344.          // retrieve the HTTP session  
345.          HttpSession httpsession = request.getSession();  
346.
```

```

347.         // retrieve the ID plus DN and generate start of page
348.         String id = getIDWithRedirect( httpsession, response, writer );
349.         if ( id == null ) return;
350.         String partialDN = (String)httpsession.getAttribute( "partialDN" );
351.         writePageHeader( "Edit Profile for " + id, writer );
352.
353.         try {
354.             // retrieve the user's profile information from the DS
355.             DirContext ctx = createLDAPContext( httpsession );
356.
357.             Attributes attrs = ctx.getAttributes( partialDN,
358.                                                 mLdapAttrsToRetrieve );
359.             emitProfileForm( attrs, writer );
360.             ctx.close();
361.         } catch ( Exception e ) {
362.             reportError( "Edit Profile:", e, writer );
363.         }
364.     }
365.

```

The `doNewProfile()` code is straightforward: It clears the HTTP session information and calls `emitProfileForm()` with a `null` first parameter (to indicate that no existing data is to be used on the HTML form).

The `doEditProfile()` code is only a little more complex. The code on lines 348 to 350 retrieves the session information, including the e-mail address (`id`) and the partial DN that identifies the authenticated user. That partial DN is passed to the JNDI `getAttributes()` method of the `DirContext` object. The `getAttributes()` method reads one entry by issuing an LDAP search with a base scope. Most JNDI methods expect names to be relative to the directory context, which is why a partial DN is typically used rather than a full DN.

[Listing 21.33](#) shows the `emitProfileForm()` utility method that is called by `doNewProfile()` and `doEditProfile()`. Like the `displayOneEntry()` method examined earlier (see [Listing 21.31](#)), `emitProfileForm()` uses a `for` loop to step through each element of the `mProfileFields fieldmap` array.

Listing 21.33 The `SimpleSiteServlet emitProfileForm()` Method

```
366.  
367.     /*  
368.      * emitProfileForm(): common code for "editprofile" and "newprofile"  
369.      *      HTTP GET subrequests.  
370.     */  
371.     private void emitProfileForm(  
372.             Attributes attrs,    // if null, create a "new profile" form  
373.             PrintWriter writer )  
374.         throws NamingException {  
375.  
376.         boolean newProfile = ( attrs == null );  
377.  
378.         writer.println( "<form action=\"saveprofile\" method=\"POST\">" );  
379.         writer.println( "<table border=\"0\">" );
380.         for ( int i = 0; i < mProfileFields.length; ++i ) {
381.             if ( !newProfile && mProfileFields[i].mNewProfileOnly ) {
382.                 continue;
383.             }
384.  
385.             // retrieve the attribute value, if present
386.             Attribute attr = null;
387.             String val = "";
388.  
389.             if ( attrs != null ) {
390.                 attr = attrs.get( mProfileFields[i].mLdapAttrName );
391.                 if ( attr != null ) {
392.                     val = (String)attr.get();
393.                 }
394.             }
395.  
396.             if ( mProfileFields[i].mLdapAttrName.equalsIgnoreCase(
397.                         "simpleEmailFormat" ) ) {
```

```

398.         // use radio buttons for preferred e-mail format
399.         String  textChecked, htmlChecked;
400.
401.         if ( val.length() == 0
402.             || val.equalsIgnoreCase( "text/plain" ) ) {
403.             textChecked = " CHECKED";
404.             htmlChecked = "";
405.         } else {
406.             textChecked = "";
407.             htmlChecked = " CHECKED";
408.         }
409.
410.         writer.println( "<tr><td>" +
411.                         + mProfileFields[i].mHtmlFieldName
412.                         + ":</td>\n<td>" );
413.         writer.println( "<input type=\"radio\" name=\""
414.                         + mProfileFields[i].mLdapAttrName
415.                         + "\" value=\"text/plain\""
416.                         + textChecked + ">&nbsp;Text<br>" );
417.         writer.println( "<input type=\"radio\" name=\""
418.                         + mProfileFields[i].mLdapAttrName
419.                         + "\" value=\"text/html\""
420.                         + htmlChecked + ">&nbsp;HTML</td></tr>" );
421.     } else {
422.         String fieldType = "text";
423.
424.         if ( mProfileFields[i].mLdapAttrName.equalsIgnoreCase(
425.             "userPassword" ) ) {
426.             // use an HTML password form field for userPassword
427.             fieldType = "password";
428.         }

```

```

429.

430.        writer.println( "<tr><td>" +
431.                                + mProfileFields[i].mHtmlFieldName
432.                                + ":</td>" );
433.        writer.print( "<td><input type=\"\" " + fieldType +
434.                                "\ name=\"\" " +
435.                                mProfileFields[i].mLdapAttrName + "\" " );
436.        if ( mProfileFields[i].mHtmlFieldSize > 0 ) {
437.            writer.print( " size=\""
438.                                + mProfileFields[i].mHtmlFieldSize
439.                                + "\" " );
440.        }
441.        writer.println( " value=\"\" " + val + "\"></td></tr>" );
442.    }
443. }
444.
445. writer.println( "</tbody></table>" );
446. if ( newProfile ) {
447.     writer.println( "<input type=\"hidden\" " +
448.                     " name=\"_NewProfile\" value=\"TRUE\">" );
449. }
450. writer.println( "<input type=\"submit\" value=\"Save\">" );
451. writer.println( "<input type=\"button\" value=\"Return to\" +
452.                     \" Previous Page\" onClick=\"window.history.back()\">" );
453. writer.println( "</form>" );
454. writePageFooter( null, false, writer );
455. }
456.

```

The code on lines 385 to 394 uses JNDI methods to retrieve the values of the existing attributes. The remainder of the `emitProfileForm()` code outputs an HTML form by creating a table that contains a series of text labels in the first column and HTML input fields in the second column. The field for the `simpleEmailFormat` attribute is represented by two radio buttons (one for `text/plain` and one for `text/html`). The field for the `userPassword` attribute is of type

`password` (to ensure that the characters typed are not echoed to the user's screen), and the other fields are simple text input fields. If an attribute has a value, the value is written out. If the `attrs` parameter is `null` (signifying that this is a new profile form) or if no values are retrieved, the input field is written with an empty value. The code on lines 445 to 454 completes the table, includes a hidden HTML form field named `_NewProfile` if this is a new profile form, and creates the **Save** and **Return to Previous Page** buttons.

The only remaining HTTP request handler method is `doSaveProfile()`, which is called when the user submits a new or modified profile. This method demonstrates how to create a new LDAP entry or modify an existing one using JNDI. [Listing 21.34](#) shows the first part of the `doSaveProfile()` method.

Listing 21.34 The `SimpleSiteServlet` `doSaveProfile()` Method (Part 1 of 2)

```
457.      /*
458.      * doSaveProfile(): handle a "saveprofile" HTTP POST subrequest
459.      */
460.      private void doSaveProfile( // HTTP POST method
461.          HttpServletRequest request,
462.          HttpServletResponse response,
463.          PrintWriter writer ) {
464.
465.          // retrieve the HTTP session
466.          HttpSession httpsession = request.getSession();
467.
468.          // determine whether we are processing a new profile
469.          boolean newProfile =
470.              ( request.getParameter( "_NewProfile" ) != null );
471.
472.          try {
473.              String id, partialDN, newPwd = null;
474.              DirContext ctx;
475.
476.              if ( newProfile ) {
477.                  // generate the ID plus DN and generate start of page
478.                  id = request.getParameter( "mail" );
479.                  if ( id == null || id.length() == 0 ) {
```

```
480.             reportError( "New profile: email address required",
481.                         null, writer );
482.
483.         }
484.
485.         partialDN = "mail=" + id + "," + mAddLocation;
486.
487.         // establish an authenticated LDAP connection
488.
489.         ctx = createLDAPContext( mAdder, mAdderPwd );
490.
491.     } else {
492.
493.         // retrieve the ID plus DN and generate start of page
494.         id = getIDWithRedirect( httpsession, response, writer );
495.
496.         if ( id == null ) return;
497.
498.         partialDN = (String)httpsession.getAttribute( "partialDN" );
499.
500.         writePageHeader( "Saving Profile for " + id, writer );
501.
502.         ctx = createLDAPContext( httpsession ); // authenticate
503.
504.     }
505.
506.     // count the values
507.
508.     int attrCount = 0;
509.
510.     for ( int i = 0; i < mProfileFields.length; ++i ) {
511.
512.         String ldapAttr = mProfileFields[i].mLdapAttrName;
513.
514.         String val = request.getParameter( ldapAttr );
515.
516.         if ( val != null ) {
517.
518.             ++attrCount;
519.
520.             if ( ldapAttr.equalsIgnoreCase( "userPassword" ) ) {
521.
522.                 newPwd = val;
523.
524.             }
525.
526.         }
527.
528.     }
529.
```

```
511. }
```

The code on lines 476 to 497 creates an authenticated JNDI LDAP directory context by calling the `createLDAPContext()` utility method we examined earlier. There are two cases:

1. **Creating a new profile.** A new partial DN is constructed from the e-mail address provided by the user with the `mAddLocation` value appended. For example, if the `mAddLocation` value is `ou=Members`, and the e-mail address provided by the user is `bjensen@example.com`, then the resulting partial DN will be `mail=bjensen@example.com, ou=Members`. The directory context is authenticated by the `mAdder` partial DN and the `mAdderPwd` password value. The `mAdder` and `mAdderPwd` values are configurable values that identify an administrative LDAP entry that has permission to add new entries.
2. **Modifying an existing profile.** The directory context is created from the identity information present in the HTTP session.

The code on lines 499 to 511 loops through the elements of the `mProfileFields` fieldmap array to determine how many attributes were included in the HTTP `POST` data. The count is used later to create arrays of the right size.

[Listing 21.35](#) shows part two of the `doSaveProfile()` method. This code adds a new LDAP entry or modifies an existing one.

Listing 21.35 The `SimpleSiteServlet doSaveProfile()` Method (Part 2 of 2)

```
512.  
513.     ModificationItem[ ] mods = null;  
514.     BasicAttributes attrs = null;  
515.  
516.     if ( newProfile ) {  
517.         // create a collection of attributes  
518.         attrs = new BasicAttributes();  
519.         // include the objectClass values in the set  
520.         BasicAttribute ocattr = new BasicAttribute( "objectClass" );  
521.         for ( int i = 0; i < mObjectClassValues.length; ++i ) {  
522.             ocattr.add( mObjectClassValues[i] );  
523.         }  
524.         attrs.put( ocattr );  
525.     } else {  
526.         // create an array of modification items
```

```
527.         mods = new ModificationItem[ attrCount ];
528.     }
529.
530.     // populate the attrs collection or the mod item array
531.     attrCount = 0;
532.     for ( int i = 0; i < mProfileFields.length; ++i ) {
533.         String ldapAttr = mProfileFields[i].mLdapAttrName;
534.         String val = request.getParameter( ldapAttr );
535.
536.         if ( val != null ) {
537.             BasicAttribute attr;
538.
539.             if ( newProfile && val.length() > 0 ) {
540.                 attr = new BasicAttribute( ldapAttr, val );
541.                 attrs.put( attr );
542.                 if ( ldapAttr.equalsIgnoreCase( "cn" ) ) {
543.                     // construct a surname from the cn (simplistic)
544.                     String snVal;
545.                     int idx = val.indexOf( " " );
546.                     if ( idx >= 0 ) {
547.                         snVal = val.substring( idx + 1 );
548.                     } else {
549.                         snVal = val;
550.                     }
551.                     attrs.put( new BasicAttribute( "sn", snVal ) );
552.                 }
553.             } else if ( !newProfile ) {
554.                 if ( val.length() > 0 ) {
555.                     attr = new BasicAttribute( ldapAttr, val );
556.                 } else { // remove all values
557.                     attr = new BasicAttribute( ldapAttr );
558.                 }
559.             }
560.             mods[ attrCount ] = attr;
561.             attrCount++;
562.         }
563.     }
564. 
```

```

559.                         mods[ attrCount ] = new ModificationItem(
560.                                         ctx.REPLACE_ATTRIBUTE, attr );
561.                         }
562.                         ++attrCount;
563.                     }
564.                 }
565.
566.             // make the change via LDAP
567.             if ( newProfile ) {
568.                 ctx.createSubcontext( partialDN, attrs );
569.                 // store user information in the HTTP session
570.                 httpsession.setAttribute( "id", id );
571.                 httpsession.setAttribute( "partialDN", partialDN );
572.                 httpsession.setAttribute( "pwd", newPwd );
573.             } else {
574.                 ctx.modifyAttributes( partialDN, mods );
575.             }
576.
577.             // finish the page and close the LDAP connection
578.             writePageFooter( "<h3>Profile successfully saved.</h3>" ,
579.                             true, writer );
580.             ctx.close();
581.         } catch ( Exception e ) {
582.             reportError( "Edit Profile:", e, writer );
583.         }
584.     }

```

The code on lines 517 to 524 is executed when a new profile is submitted. It creates a new JNDI `BasicAttributes` object named `attrs` and adds the object class values required for new entries. The `BasicAttributes` object is an implementation of the JNDI `Attributes` interface, which holds a collection of directory attributes.

The code on line 527 is executed when an existing profile is modified. It creates an array of JNDI `ModificationItem` objects named `mods`; each element represents a modification to one

LDAP attribute. The `for` loop that encompasses lines 532 to 564 uses the posted HTML form data to populate the `attrs` object or the `mods` array, as appropriate.

For new profiles, a `BasicAttribute` object is created and added to the `attrs` collection for each field (lines 540 and 541). The code on lines 542 to 551 creates a surname (`sn`) attribute based on the common name (`cn`) attribute (the `sn` attribute is never exposed to SimpleSite users). The LDAP add operation itself is performed by the `createSubcontext()` call on line 568. The terminology used by JNDI does not always match that used by LDAP practitioners; in JNDI a subcontext is simply an LDAP entry that is located beneath the base DN that was used to create the initial directory context. The base DN used to create the initial directory context is stored in the `mBaseDN` `SimpleSiteServlet` member variable and is also part of the `mLdapURL` value. For example, if the `partialDN` value is `mail=mcs@netscape.com, ou=Members` and the `mBaseDN` value is `dc=simple,dc=example,dc=com`, then the entry created will have a full DN of `mail=mcs@netscape.com,ou=Members, dc=simple,dc=example,dc=com`.

For updates to existing profiles, an LDAP modification type that specifies a replace operation is always used within each `mods` array element (lines 559 and 560). The LDAP modify operation is performed by the JNDI `modifyAttributes()` call on line 574; this time, the purpose of the method is clear from its name.

[Listing 21.36](#) shows three utility methods that create HTML output. These functions do not perform any LDAP-related work; therefore, they are not described in detail.

Listing 21.36 The `SimpleSiteServlet` `writePageHeader()`, `writePageFooter()`, and `writeHREFButton()` Methods

```
788.  
789.      /*  
790.      * writePageHeader(): Output an HTML page header with title.  
791.      */  
792.      private void writePageHeader()  
793.          String title,  
794.          PrintWriter writer ) {  
795.  
796.          if ( title == null ) {  
797.              title = mApplName;  
798.          }  
799.          writer.println( "<html>" );  
800.          writer.println( "<head><title>" + title + "</title></head>" );  
801.          writer.println( "<body>\n<center><h2>" + title + "</h2></center>" );  
802.      }
```

```
803.  
804.     /*  
805.      * writePageFooter(): Output an HTML page footer.  
806.     */  
807.     private void writePageFooter(  
808.             String statusMessage,  
809.             boolean displayActions,  
810.             PrintWriter writer ) {  
811.  
812.     if ( statusMessage != null ) {  
813.         writer.println( "<p>" + statusMessage );  
814.     }  
815.     if ( displayActions ) {  
816.         writer.println( "<form>" );  
817.         writeHREFButton( "Find Person", "../find.htm", writer );  
818.         writeHREFButton( "Edit Profile", "editprofile", writer );  
819.         writeHREFButton( "Log Out ", "logout", writer );  
820.         writer.println( "</form>" );  
821.     }  
822.  
823.     writer.println( "</body>\n</html>" );  
824. }  
825.  
826. /*  
827.      * writeHREFButton(): Output a button that is a link.  
828.     */  
829.     private void writeHREFButton(  
830.             String label,  
831.             String url,  
832.             PrintWriter writer ) {  
833.  
834.     writer.println( "<input type=\"button\" value=\"" );
```

```

835.           + label + "\" onClick=\"window.location.href='"
836.           + url + "'\\>&nbsp;&nbsp;" );
837.       }
838.

```

[Listing 21.37](#) shows the last two methods that make up the `SimpleSiteServlet` class: `unknownRequest()` and `reportError()`. These two methods are used to report errors to the user in a consistent way.

Listing 21.37 The `SimpleSiteServlet` `unknownRequest()` and `reportError()` Methods

```

839.
840.      /*
841.      * unknownRequest(): Output an "unknownRequest" HTML error
842.      *     page (entire page).
843.      */
844.      private void unknownRequest(
845.          String msg,
846.          String type, // get, post, ...
847.          PrintWriter writer ) {
848.
849.          writePageHeader( mApplName + " - unknown request", writer );
850.          reportError( "Unknown " + type + " request: (" + msg + ")",
851.                      null, writer );
852.      }
853.
854.      /*
855.      * reportError(): Output an HTML error page (all but page header).
856.      */
857.      private void reportError(
858.          String msg,
859.          Exception e,

```

```

860.             PrintWriter writer ) {
861.
862.             writer.println( "<p><b>" + msg + "</b><br>" );
863.             if ( e != null ) {
864.                 writer.println( "Error: " + e + "<br>" );
865.             }
866.             writer.println( "<form><input type=\"button\" "
867.                             + "value=\"Try Again\" "
868.                             + "onClick=\"window.history.back()\"></form>" );
869.             writePageFooter( null, false, writer );
870.         }
871.     }

```

Although the `SimpleSiteServlet.java` file includes fewer than 900 lines of code, it does a lot: It provides a complete, authenticated Web site with user profile storage and a search capability.

Ideas for Improvement

The SimpleSite application could be improved in many ways. Here are a few ideas:

- Separate the HTML page content from the programming logic. Although it is simpler for beginners to generate HTML code directly from Java methods, use of JSP or another template-based approach is highly recommended. That way, Web designers can improve the appearance of the application without requiring Java source code changes.
- Improve the error reporting and handling. The code makes very little effort to present JNDI exceptions in a user-friendly way.
- Add a **Confirm Password** field to the **Create New Profile** form to protect users against simple typographic errors when entering their passwords.
- Add a separate field for surname (rather than programmatically extracting an `sn` value from the name provided by the user).
- Reduce the update load on the directory server by detecting and omitting the profile values that have no changes from the list of modifications sent to the server.
- Support more than one value within some of the user profile fields. For example, the `simpleExpertise` attribute typically has more than one value from the user's perspective, but it is stored as a single free-form string value.
- Add additional search capabilities. Either/or searching, as well as substring and approximate matching for fields such as **Name**, could be supported.

Developing New Applications Checklist

The following checklist summarizes the issues you should consider when developing new directory-enabled applications:

- Identify one or more needs that can be filled by development of a new directory-enabled application.
- Check that it makes sense to use an LDAP directory for the application.
- Choose the best LDAP development tool for the task at hand.
- Design your application to fit in well with other applications and the directory service itself.
- Document how the application will be integrated with the directory service. Share that information with the people who maintain the directory service.
- Consider performance and scalability during application design and implementation.
- Develop a prototype, and conduct pilot tests.
- Leverage existing code if possible.
- Avoid common LDAP application development mistakes.

Further Reading

Active Directory Services Interface (ADSI). Available on the World Wide Web at <http://www.microsoft.com/msdn/sdk>.

DSML Tools. Available on the World Wide Web at <http://www.dsmltools.org>.

Implementing LDAP. M. Wilcox, Wrox Press, 1999.

The Java LDAP Application Program Interface (Internet Draft). R. Weltman, C. Tomlinson, M. Kekic, S. Sonntag, J. Sermersheim, T. Howes, and M. Smith, 2001. Available on the World Wide Web at <http://www.ietf.org>.

Java Naming and Directory Interface (JNDI). Information is available on the World Wide Web at <http://java.sun.com/products/jndi>.

The Java Programming Language (3rd edition). K. Arnold, J. Gosling, and D. Holmes. Addison-Wesley, 2000.

Java Servlet Technology. Information is available on the World Wide Web at <http://java.sun.com/products/servlet>.

JNDI API Tutorial and Reference: Building Directory-Enabled Java Applications. R. Lee and S. Seligman, Addison-Wesley, 2000.

Jtag LDAP Library. Virtuas. Available on the World Wide Web at <http://www.virtuascommunity.com/2k2/servlet/VSController?transaction=listdnlds&packageid=7&free=N>.

LDAP ASP Component. /n software inc. Information is available on the World Wide Web at <http://www.nsoftware.com/showdesc.asp?ctl=LDAP>.

LDAP Client API for Python. Available on the World Wide Web at <http://python-ldap.sourceforge.net>.

LDAP JSP Tag Library. Simya Consultancy. Available on the World Wide Web at <http://www.simya.net/products.html>.

LDAP Programming with Java. R. Weltman and T. Dahbura, Addison-Wesley, 2000.

mozilla.org Directory SDK source code releases for C, Java, and Perl. Available on the World Wide Web at <http://www.mozilla.org/directory>.

The Net::LDAP Perl-LDAP modules. G. Barr and contributors. Information is available on the World Wide Web at <http://perl-ldap.sourceforge.net>.

Netscape Directory SDKs for C, Java, and PerLDAP. Available on the AOL Strategic Business Solutions site at <http://enterprise.netscape.com>.

OASIS Directory Services Technical Committee (DSML). Information is available on the World Wide Web at <http://www.oasis-open.org/committees/dsml>.

Programming Directory-Enabled Applications with Lightweight Directory Access Protocol. T. Howes and M. Smith, Macmillan Technical Publishing, 1997.

Ruby/LDAP. Available on the World Wide Web at <http://ruby-ldap.sourceforge.net>.

[SourceForge.Net](#) Open Source development site. Available on the World Wide Web at <http://sourceforge.net>.

X.509 Internet Public Key Infrastructure Online Certificate Status Protocol—OCSP (RFC 2560). M. Myers, R. Ankney, A. Malpani, S. Galperin, and C. Adams, 1999. Available on the World Wide Web at <http://www.ietf.org/rfc/rfc2560.txt>.

Team LiB

◀ PREVIOUS

NEXT ▶

Looking Ahead

By using LDAP within an application, you can enable exciting features and lower the cost of maintenance. In [Chapter 22](#), Directory-Enabling Existing Applications, we will continue our discussion of how to leverage your deployed directory service effectively by looking at how to add LDAP support to existing applications.

Chapter 22. Directory-Enabling Existing Applications

- Reasons to Directory-Enable Existing Applications
- Advice for Directory-Enabling Existing Applications
- Example 1: A Directory-Enabled finger Service
- Example 2: Adding LDAP Address Lookup to an E-Mail Client
- Directory-Enabling Existing Applications Checklist
- Further Reading
- Looking Ahead

[Chapter 21](#), Developing New Applications, described the process and benefits of developing new applications that use your directory service. The basic idea is to leverage your deployed LDAP directory and bring its benefits to your users.

But what about existing applications? They can also benefit from being directory-enabled. Often it is easier to persuade people to use a new release of an application they already use than it is to get them to try a new application. Therefore, it is often advantageous to directory-enable an existing application instead of writing a new one. If the application to be modified has a split client/server, or *thin client*, architecture, you may be able to LDAP-enable it by making changes only on the server side and thus avoid the need to deploy new software to end users. Most Web (HTML- and XML-based) applications can be LDAP-enabled in this way.

The usual motivation for directory-enabling an existing application is to add new features that use LDAP, or to change an existing feature to use LDAP instead of an application-specific database. By leveraging a deployed LDAP directory service, you may be able to improve the end-user experience and reduce system administration costs at the same time.

The first section of this chapter gives some reasons why it makes sense to directory-enable an existing application. Next some advice is provided to help you easily and successfully directory-enable applications. The last portion of the chapter includes two concrete examples that show how to add LDAP support to an application. For general information on the software development kits (SDKs) and tools that can be used to integrate your applications with LDAP, refer to [Chapter 21](#).

Reasons to Directory-Enable Existing Applications

By integrating existing applications with your directory service, you can do any or all of the following:

- **Enable new features in the applications.** For example, users of an e-mail client application can access shared address books in addition to local, private address books.
- **Lower your data management costs.** LDAP-enabling some existing applications may allow you to eliminate redundant copies of data and to decommission private directories and databases.
- **Simplify life for end users by leveraging your deployed directory service.** For example, using a centralized directory for authentication reduces the number of distinct passwords people need to remember.
- **Bring the directory service to your end users.** By LDAP-enabling an existing application instead of forcing people to switch to a new one, you make it easier for end users to adopt your directory service.

However, sometimes it does not make sense to integrate an application with a directory service. For example, the data the application uses may violate your directory data policy. [Chapter 21](#), Developing New Applications, provides more information on when it does *not* make sense to directory-enable an application.

Each of the benefits of integrating an application with a directory service is discussed further in the following sections.

Enabling New Features in Applications

New features can increase the value of the application and expose users to your directory service in a new way. For example, suppose you have a widely used e-mail application with only local lookup capabilities for its address book; you could add LDAP lookup capabilities so that users could access other address books shared across the organization. They would thus access your directory service more because it would be closely tied to a task they perform often: addressing and sending e-mail.

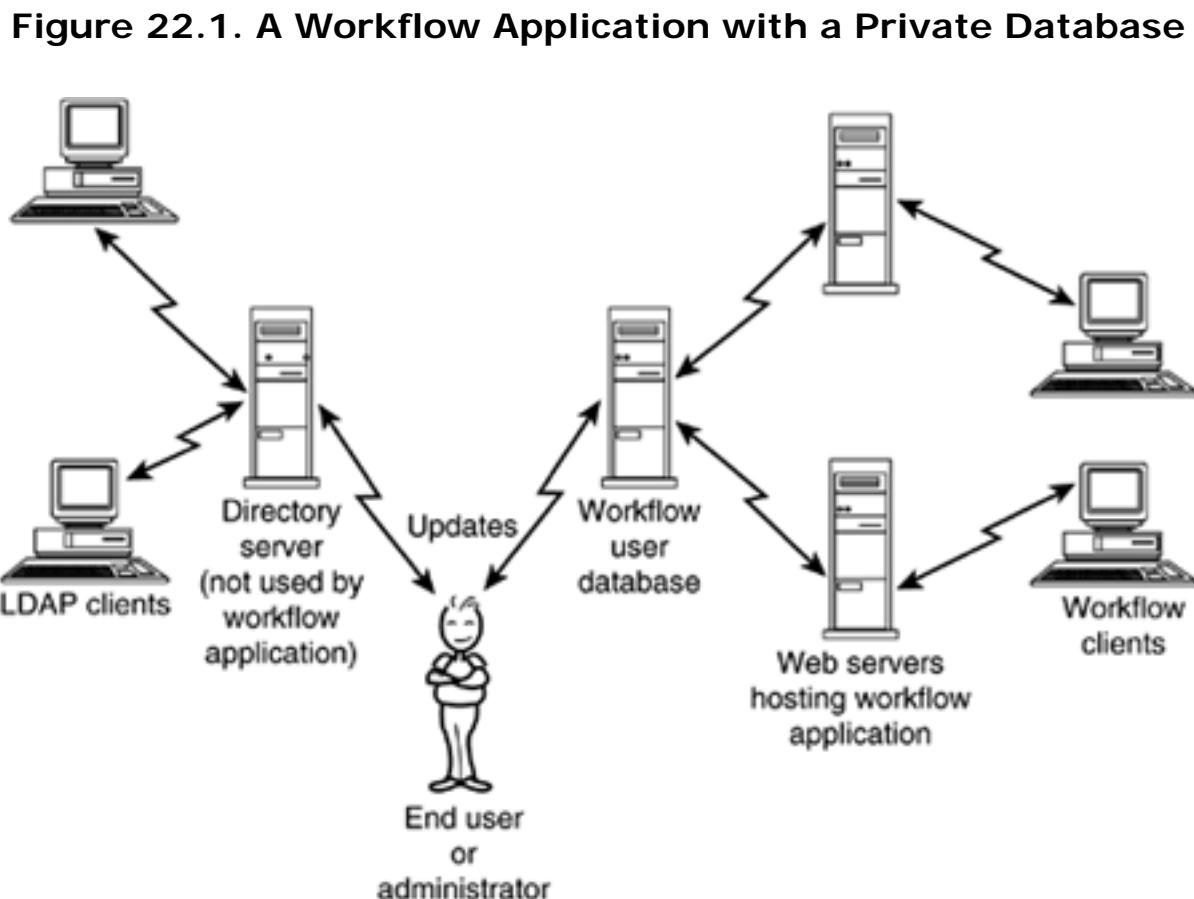
Features such as a local-to-global address book upgrade are extensions of existing features, but you might add a completely new feature too. For example, modifying a server application to store its configuration information in an LDAP directory allows the configuration to be shared among a set of similar servers, thereby reducing deployment and management costs.

Lowering Data Management Costs

Directory-enabling an application that has a private, application-specific data store can reduce the cost of data management. The private data store can be taken out of service, and all the resources used to run and maintain it can be eliminated. Old, application-specific directories and databases are often more expensive to maintain than a centralized, LDAP-based directory service; expertise is harder to find, management tools are weaker, and platform choices are more limited than with an LDAP directory.

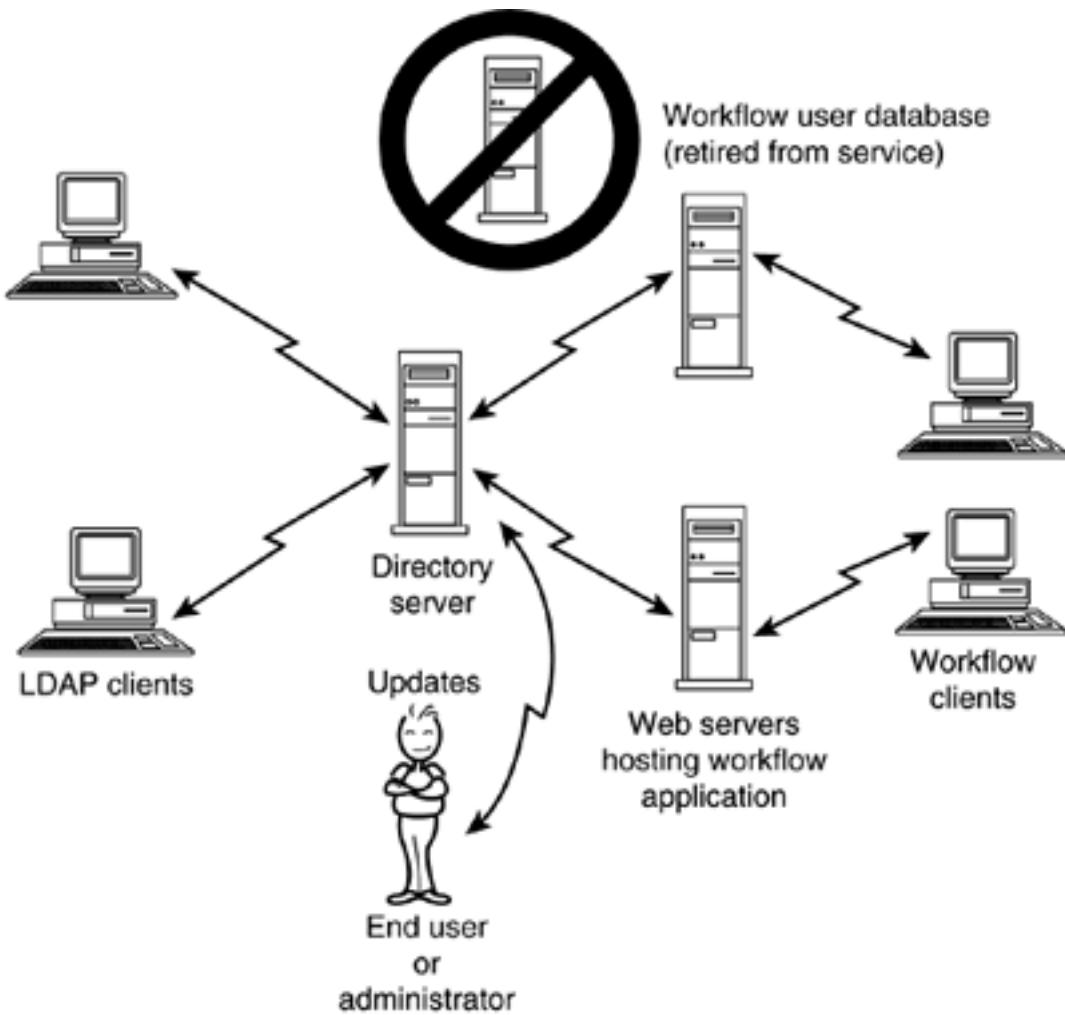
Another way data management costs can be reduced is by eliminating redundant copies of data elements; the need for synchronization between a data source and your centralized directory is eliminated with the redundant data. If some of the data used by the application is personal data, your end users will appreciate the fact that data about them is stored in one less place, therefore making changes to the data easier.

For example, [Figure 22.1](#) shows an existing workflow system with its own database that stores information about users of the system. End users and administrators must update two separate information stores.



By consolidating into a directory service the information held in the private database, you eliminate redundant data and simplify the system as a whole. [Figure 22.2](#) shows the revised, directory-enabled application.

Figure 22.2. The Directory-Enabled Workflow Application



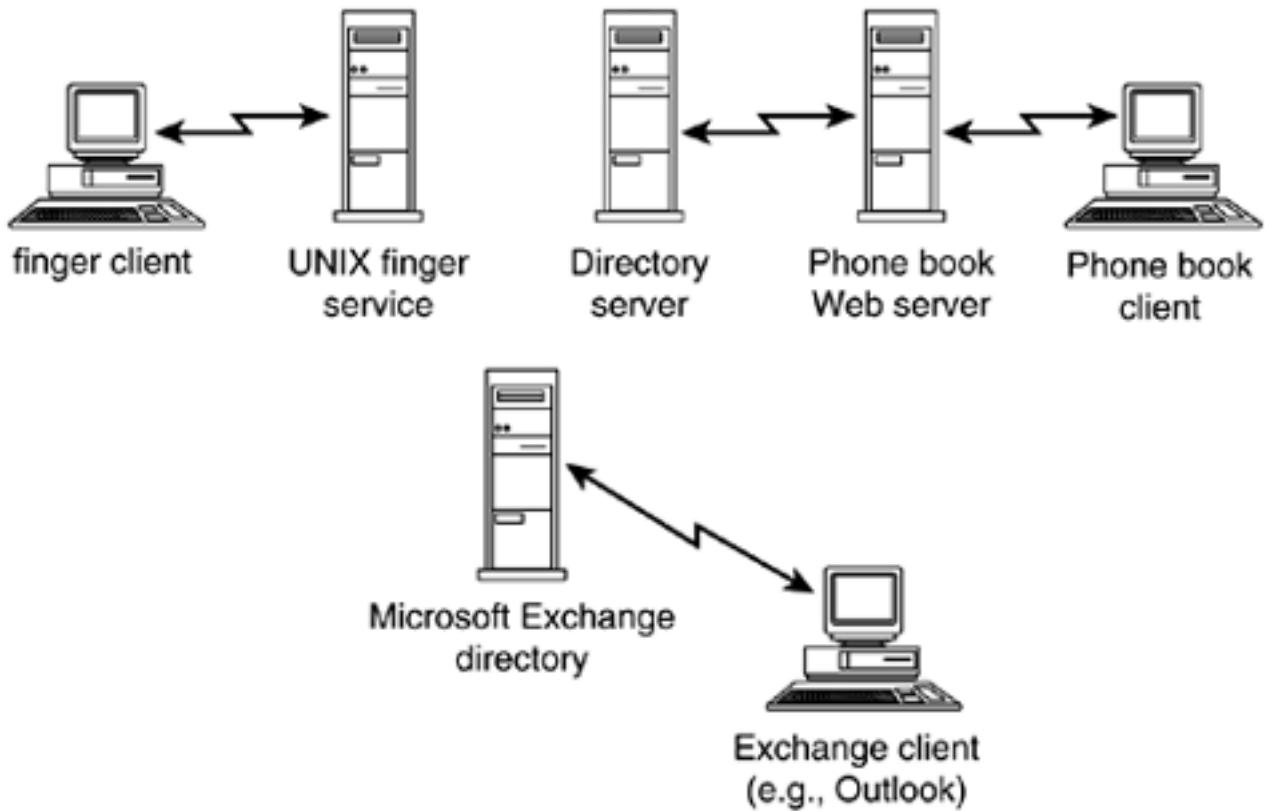
Simplifying Life for End Users

Another reason to leverage your deployed directory service in existing applications is to simplify life for end users. Users will thank you every time you eliminate another redundant, application-specific data store. For example, if all your applications that send postal mail use a common directory service to determine a person's mailing address, employees need to change their addresses in only one place when they move.

Another way to simplify life for end users is to create a consistent view of information from within a variety of applications. All applications that use the same LDAP directory service access the same information using the same protocol. Use of a common data store and access method leads to greater consistency in the look and feel of different applications and in the terminology they use. This makes it easier for end users to carry over their knowledge of one application to another. For example, a public Web site that uses one central directory service for all of its Web-based applications may be easier for end users to use than a site that uses many different databases.

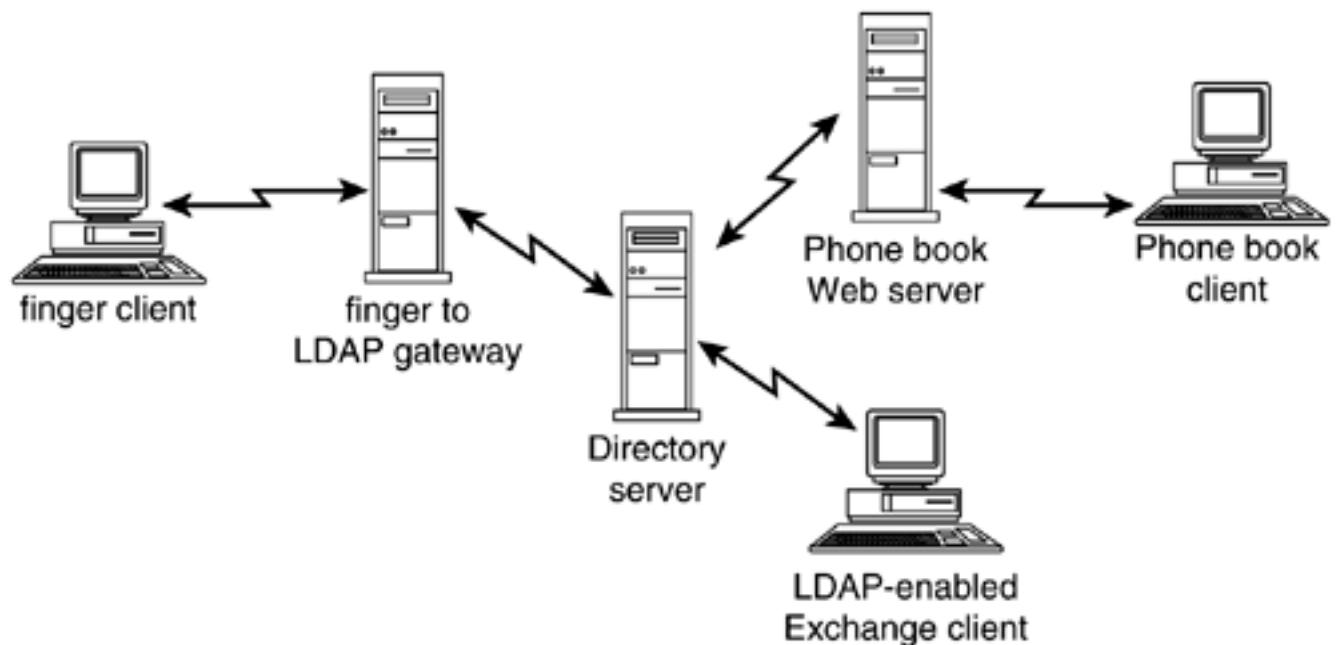
A related motivating factor is that over time, people will expect the directory service to be used when they access certain kinds of information. For example, novice users who discover your Web-based directory phone book application may assume that all contact information about them is stored in one directory service. They will be disappointed if, as shown in [Figure 22.3](#), the same information they find in the phone book isn't returned when they execute the `finger` command on their Unix workstation.

Figure 22.3. Contact Information Lookup Applications That Are Not Integrated



Directory-enabling the finger service can reduce end-user confusion and frustration. [Figure 22.4](#) shows the improved situation.

Figure 22.4. Contact Information Lookup Applications That Are Integrated and Directory-Enabled



Bringing the Directory Service to Your End Users

If one of your goals is to increase the use and visibility of your directory service, adding LDAP support to applications can be somewhat self-serving. Directory-enabled applications bring the directory service closer to your end users, making it more likely that they will use the service. By making your service an essential part of users' everyday life, you ensure future financial and political support for the directory. Data quality and data consistency

across data stores also improve as more people use a common directory service.

People are more likely to try a new version of a software application they already use than they are to try a completely new application. Therefore, it is often better to add a little bit of LDAP support to an existing application than it is to invest in the development and deployment of a brand-new, richly featured directory-enabled application. The key is to add features that are valuable to your users so that they will want to use your directory service.

For example, you might have a Web-based conference room locator service running on one of your internal Web servers. Users of the application can search for a conference room on any of your campuses and view a map that shows the location of room. This application could be enhanced to locate people as well. Real value to end users can be provided by the addition of the capability to search the directory, retrieve a person's building and office number, and map the location.

Team LiB

◀ PREVIOUS

NEXT ▶

Advice for Directory-Enabling Existing Applications

Although adding LDAP support to an existing application is often better than creating a brand-new LDAP application, it can also be more difficult. The degree of difficulty depends on the nature of the application you're modifying, the scope of the changes you need to make, and the level of LDAP expertise you expect the application's users to have. The following techniques should ease the pain of development and increase the payoff when you are done. Note that some choices conflict with others, so not all of these techniques will apply in all situations:

- To reduce barriers to deployment, hide the directory integration from users and systems that depend on the application.
- Promote your new features by making the new directory capabilities visible to the users of the application.
- Use a protocol gateway to achieve integration and reduce barriers to deployment.
- Avoid problematic architectural choices that adversely affect performance, involve the duplication of data, and so on.
- Consider how the directory service will be affected, and inform those who run the service if additional capacity or any other modifications are required.
- Plan for a smooth transition from the existing application to the new directory-enabled version.
- Be creative, and consider all your options.

Each idea is discussed in greater detail in the following sections.

Hide the Directory Integration

Depending on the kind of directory-enabling you do, it may make sense to try to hide the changes from the users of the application as much as possible. This can be especially important if the application is used widely or is used by other systems that are difficult and expensive to modify.

For example, suppose you have an e-mail delivery system that uses five servers, each with its own local database of information about users and mailing lists. You want to reduce your system management costs by directory-enabling the e-mail servers so that they all share a common directory of information about users and mailing lists. You may be able to hide this change from users of the e-mail service by doing all the directory integration on the back end (that is, in the e-mail servers themselves). This way, you avoid deploying new e-mail clients, and your users do not need to learn anything new.

Another example of hiding directory integration is to modify a set of Web-based applications that each use a proprietary database for access control and authorization so that instead they use an LDAP directory—without changing the user interface of any of the applications. You can eliminate the proprietary databases without requiring that visitors to your Web site learn a new interface.

Hiding directory integration from the people or systems that use an application provides the benefits of a shared directory service while minimizing the associated deployment costs.

Make the New Directory Capabilities Visible

The alternative to hiding directory integration is, logically enough, to make the new directory capabilities visible to the application users. If you are truly adding new features—or if you

simply want to promote your directory service—exposing the new features is helpful.

For example, suppose you have a Web-based workflow application that requires users to know a person's user ID to include that person in a flow. Adding a feature that allows directory lookups of people based on name, telephone number, and other criteria may be a great benefit to workflow users. To ensure that people notice the new feature, you could, for example, add a new **Directory Lookup** button to the workflow application's interface. As a side benefit, users who appreciate the new feature will know that it is made possible by a directory service, which will help promote the service and ensure future funding.

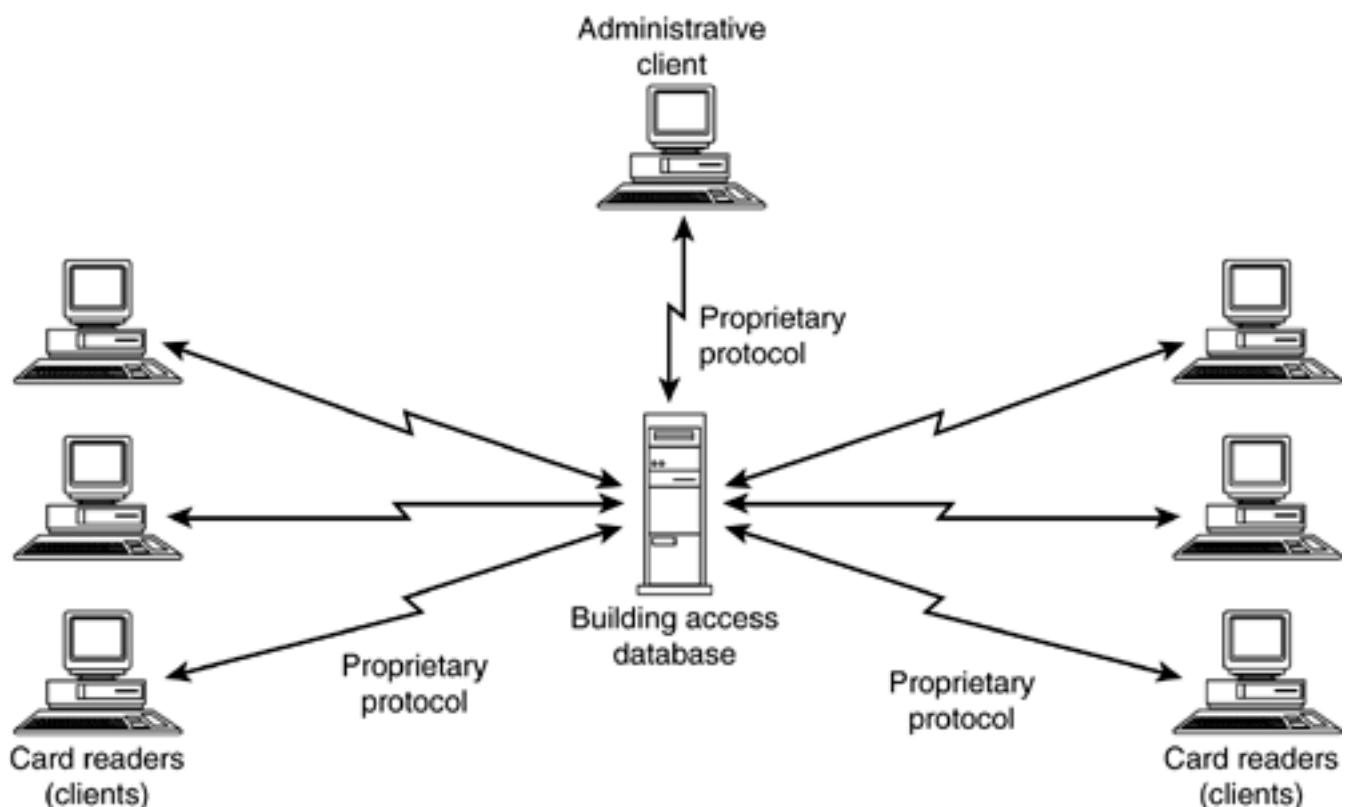
Another promotional technique is to modify the application to let people know they are using a directory service. For example, you might add a banner that reads "Directory-Enabled finger Service" to the output returned by an LDAP-aware Unix finger server.

Use a Protocol Gateway to Achieve Integration

Developing a gateway that translates between an existing protocol and LDAP is often an effective way to directory-enable an existing, widely used application. A *gateway* is a server application that accepts a request using one application protocol, translates it to another protocol such as LDAP, and passes it to another server. A similar translation procedure is performed in the reverse direction on information returned from the directory server.

For example, [Figure 22.5](#) shows a building access system that consists of dozens of electronic card readers that communicate with a centralized building access database through a proprietary HTTP- and XML-based protocol.

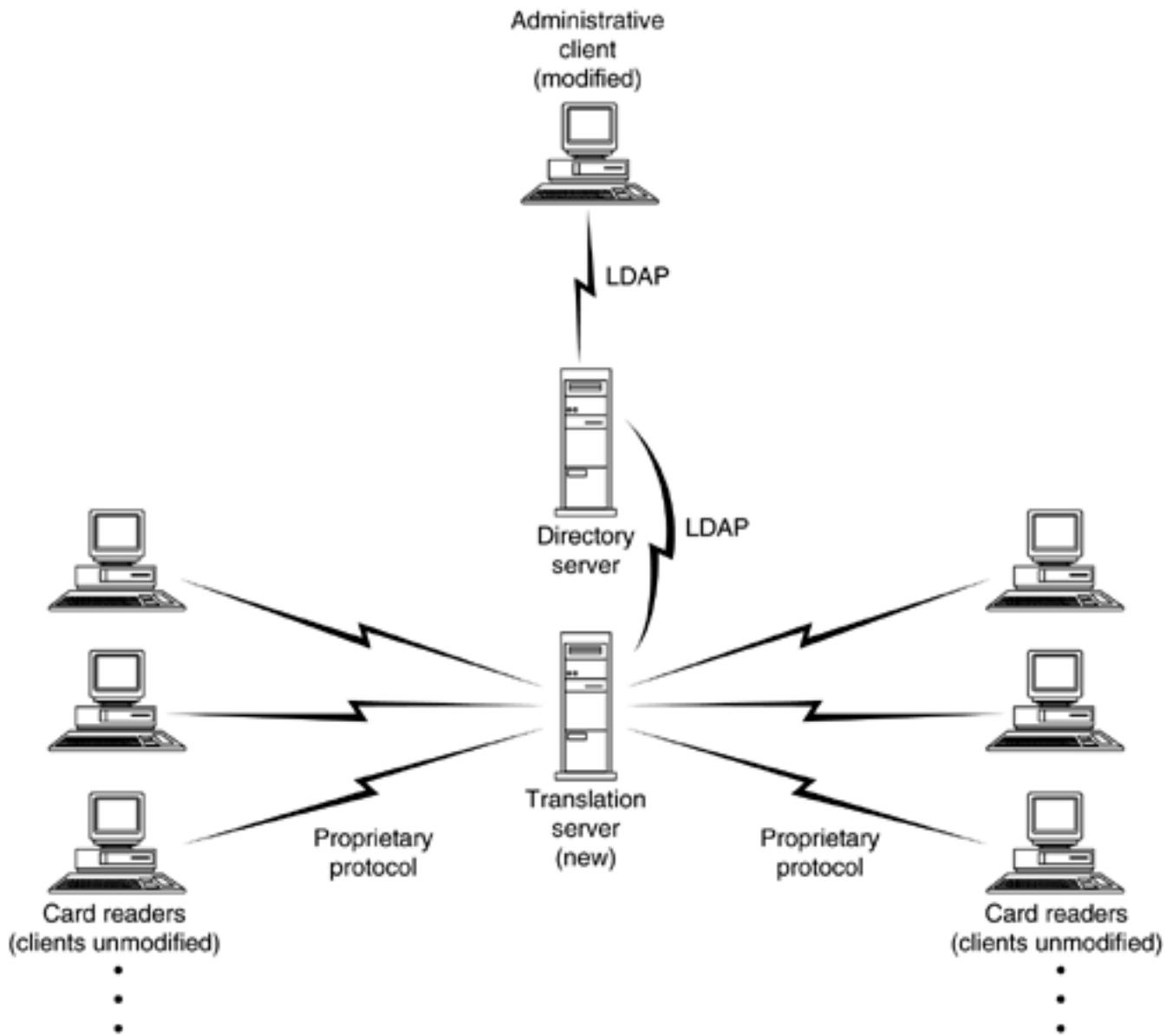
Figure 22.5. A Building Access System before Directory-Enabling



Let's assume that the building access database contains redundant information about people, so you decide to replace it with your directory service. One approach would be to upgrade to card readers that speak LDAP directly. Because of the high cost and lack of availability of such devices, however, that may not be a cost-effective option. A cheaper and

simpler approach might be to hide the directory integration from the card readers by building an intermediate server that acts as a gateway between the card readers' proprietary protocol and LDAP. [Figure 22.6](#) shows the new deployment architecture.

Figure 22.6. A Directory-Enabled Building Access System



The success of an approach that uses a protocol gateway depends greatly on how easy it is to bridge the gap between the existing protocol and your directory service, and also on whether the existing protocol is well specified. Assuming you can pull it off, using an application protocol gateway to bring the directory service to your users allows you to deploy the new directory-integrated solution more quickly and with less expense.

Avoid Problematic Architectural Choices

For all their good points, gateways have some problematic characteristics as well. Using a gateway sometimes leads to unacceptable performance, introducing delay and consuming additional system resources to translate from one protocol to another. Using gateways also causes management headaches: The gateway is one more piece of software to monitor and keep running.

In addition, gateways sometimes produce an imperfect directory experience for users.

Typically, compromises must be made when any two protocols are being fitted together, and if too much is lost in the translation, users may complain. For example, advanced features of LDAP such as the Virtual List View (VLV) control are not supported by older protocols such as the Unix finger protocol. VLV allows users to browse tens of thousands of directory entries efficiently; however, people who access your directory service via a finger gateway will not be able to use VLV to do so.

Integration schemes that make copies of directory data are also troublesome. Accessing live directory data is generally preferred over accessing data that has been duplicated and placed in another data store. Synchronization between data stores is often difficult, resource intensive, and fragile. Consider the trade-off between high-performance access to data, which you might achieve best by copying the data, and timely access to changing data. More information on integrating other data sources with your directory service can be found in [Chapter 23](#), Directory Coexistence.

Consider How the Directory Service Will Be Affected

When you are directory-enabling an application, always consider how the application uses the directory service. The people who run the directory need to know in advance what kind of demands the new directory-enabled version of the application will place on their service. The application may generate additional load in the form of LDAP read and update operations. Changes to the directory schema may be required to accommodate the application. Sometimes an application requires an LDAP protocol feature that is not supported by the directory server software, so an upgrade may be necessary, or you may need to pursue a different strategy to accomplish the directory integration.

For example, the directory-enabled version of the building access system described earlier would increase the load on the directory service. The number of card key accesses performed weekly with the existing, non-LDAP building access database can be used to easily calculate the amount of the increase. Suppose this number is 70,000. The new directory-enabled system might use two LDAP operations when deciding whether to grant access: one to find the user associated with a card (a subtree search) and one to check whether the user is in a group that has access to a given building (a compare operation to check group or role membership). The total load imposed on the directory service by the new system would thus be 140,000 operations per week.

In addition, each operation needs to be completed fairly quickly because people will not tolerate long delays while waiting for a door to be unlocked (in fact, they may become impatient and try again, therefore generating even more load). You should also take into account peak load: An application may be used more often at certain times of the day or on certain days of the week. For example, most card key accesses will probably occur when people arrive for work, which might be between 8:00 and 10:30 A.M. on weekdays.

If you provide detailed information about how your directory-enabled application will affect the directory service, those responsible for the service can make plans to accommodate the application. More advice on communicating the needs of your application to those who maintain a directory service can be found in [Chapter 21](#), Developing New Applications.

Plan for a Smooth Transition

You should plan an orderly transition from the existing application to the new directory-enabled version. By introducing the new directory capabilities in a controlled way, you reduce the risk associated with the change. To transition smoothly, you must continue to support the old application for a period of time while users and dependent systems switch and adjust to the new version.

Try to adopt an incremental approach in which you pilot the new version of the application before fully rolling it out. For most applications, you can do this by announcing the new directory-enabled version to a small number of users, extracting and incorporating their feedback, and then rolling it out to your entire user base in phases. If the application you modify is used primarily by other systems, you may be able to limit how many systems you initially switch over to use the directory-enabled version of the application. Another approach is to test the modified application with the dependent systems during off-hours, so that your business is not affected as severely if problems occur.

Be Creative, and Consider All Your Options

As you consider how best to directory-enable an existing application, be creative. The most obvious and straightforward approach is not always the best one. Consider all the possible techniques presented in this chapter to determine which approach is best for your application.

When considering the alternatives, think about what you want to accomplish by directory-enabling the application. Do you want to provide completely new capabilities to end users? Are you trying to save system maintenance costs by decommissioning a redundant data store? Do you want to introduce the directory-enabled application quietly, or do you want to make a big splash? Your approach should be influenced by the answers to these and many other questions.

For example, when adding LDAP support to a widely used application such as an e-mail client, adopting an address book metaphor for the interface to the new LDAP features may increase the rate of adoption and reduce the learning curve for your users. On the other hand, introducing an entirely new user interface paradigm, such as a **Search LDAP** screen, allows more flexibility in exposing LDAP-specific features such as compound Boolean search filters or advanced features such as VLV. With some creative thinking, you can design your application to provide power users with access to advanced LDAP capabilities but preserve a familiar address book interface for novices. A possible approach could be to present a search field that accepts simple strings (for novices) as well as raw LDAP search filters (for advanced users).

As another example, consider a legacy application with a proprietary data store that you want to LDAP-enable. If performance is a key requirement, it may be appropriate to modify the application to speak LDAP natively. But suppose the primary goal of your LDAP integration effort is to eliminate the database system used by the application and reduce your data store maintenance costs. Then the best approach for integration might involve a protocol or API-level gateway that looks to the application like its old data store but accesses data from an LDAP directory. The gateway approach may be easier to develop and deploy, thereby allowing you to achieve your primary goal of decommissioning the legacy data store more quickly.

Example 1: A Directory-Enabled finger Service

A finger service is a simple client/server account lookup mechanism supported by all Unix operating systems. A finger client exists for most desktop operating systems, including all flavors of Unix, Microsoft Windows, and Apple MacOS. A finger server, included with all Unix systems, can return information about local accounts to all finger clients on the network (finger servers have been developed for non-Unix systems as well). The finger clients and servers communicate using a simple text-based protocol that was originally developed by the BSD Unix developers in the 1980s.

Assuming that many end users have access to a finger client and know how to use it, it's sensible to make some of the people information stored in a directory service available via finger. This section presents source code and usage examples for a directory-enabled finger server written in Perl. The directory-enabled finger service searches an LDAP directory and returns information about people.

The Integration Approach

The goal is to leverage the knowledge and client software that users already have, so a gateway is used to integrate finger with LDAP. On most Unix systems, an executable called `in.fingerd` is executed in response to incoming finger protocol requests. Replacing that executable with a script that understands the finger protocol but uses LDAP to retrieve information establishes compatibility with existing clients. This is an example of bringing the directory service to your users; no new client software needs to be installed for people to access the directory via the finger gateway.

The LDAP finger gateway is a standalone executable; it does not execute within the confines of an existing application. Therefore, nearly any programming language that provides access to LDAP could be used. The gateway example is written in the Perl 5 language, and it uses the PerLDAP object-oriented LDAP access module available from the Mozilla Web site at <http://mozilla.org/directory/perldap>. This choice allows rapid prototyping, provides maximum portability for the code, and performs acceptably. The `lfingerd.pl` script replaces the standard `in.fingerd` binary program executed by Unix's `inetd` process, which is responsible for managing the execution of a variety of TCP/IP services.

Directory Use

The `lfingerd.pl` gateway uses a directory service in a simple way. The LDAP service is accessed anonymously; that is, without authentication. Given a query string that is sent by the finger client, a search for user entries is performed with a filter that includes an (`objectClass=person`) component. Each entry returned by the LDAP search is printed in a format that is familiar to finger users.

Along with contact information such as name and telephone number, the `lfingerd.pl` gateway retrieves status information for AOL Instant Messenger (AIM) users. This feature relies on Netscape Directory Server 6, which allows retrieval of instant messaging online/offline status over LDAP for person entries that include appropriate attributes. [Listing 22.1](#) shows a sample entry that includes an AIM ID.

Listing 22.1 An Entry That Includes an AIM ID

```
dn: uid=bjensen,ou=People,dc=example,dc=com
cn: Barbara Jensen
cn: Babs Jensen
sn: Jensen
givenName: Barbara
objectClass: top
objectClass: person
objectClass: organizationalPerson
objectClass: inetOrgPerson
objectClass: nsAIMPresence
ou: Product Development
ou: People
L: Cupertino
uid: bjensen
mail: bjensen@example.com
telephoneNumber: +1 408 555 1862
facsimileTelephoneNumber: +1 408 555 1992
roomNumber: 0209
userPassword: hifalutin
nsAIMID:Babs Jensen
```

The AIM-related attributes and values are shown in bold type. **nsAIMPresence** is an auxiliary object class that allows these three attribute types:

1. **nsAIMID**. An AIM identifier; also known as a *screen name* (syntax `DirectoryString`).
2. **nsAIMStatusText**. An enumerated text attribute that indicates the user's online or offline status. It has one of these values: `ONLINE`, `OFFLINE`, `ERROR` (syntax `DirectoryString`).
3. **nsAIMStatusGraphic**. A GIF image that reflects the user's status in a graphical way (syntax `OctetString`).

In Netscape Directory Server 6, the `nsAIMStatusText` and `nsAIMStatusGraphic` attributes are virtual, operational attributes. Recall from [Chapter 2](#), Introduction to LDAP, that operational attributes must be retrieved by name when an LDAP search operation is used to

retrieve them. The `nsAIMStatusText` and `nsAIMStatusGraphic` attributes are also *virtual*, which means the attribute values are not stored in the directory server's database on disk but instead are computed on demand. For these two attributes, the values are computed by a Presence plug-in that is bundled with Netscape Directory Server 6. The Presence plug-in connects to AOL's AIM service to compute the correct `nsAIMStatusText` and `nsAIMStatusGraphic` values for each entry returned by an LDAP search operation.

The `lfingerd.pl` gateway avoids reading unnecessary attributes from the user entries. The additional load placed on the directory service may be estimated directly on the basis of the expected use of the finger gateway. There are no other significant directory architecture issues to consider for this application.

The End-User Experience

[Listing 22.2](#) shows a `finger` command (in bold type), along with the output produced by a standard, non-LDAP-enabled Unix finger service.

Listing 22.2 A Non-LDAP-Enabled Unix `finger` Lookup for `dsmith`

```
finger dsmith@unix.example.com
```

```
Login name: dsmith           In real life: Daniel Smith
```

```
Directory: /u/dsmith          Shell: /bin/tcsh
```

```
On since Jul 14 09:05:10 on console from :0
```

```
8 minutes 42 seconds Idle Time
```

```
No unread mail
```

```
Project: Human Resources web site
```

```
Plan:
```

```
Send me e-mail at dsmith@example.com
```

[Listing 22.3](#) shows a lookup using the LDAP directory-enabled finger service. In this case, only one entry matches the query string.

[Listing 22.4](#) shows an LDAP-enabled finger example that returns an entry with more attributes. Babs's entry contains `postalAddress` and `labeledURI` attributes, as well as an `nsAIMID` attribute. Babs is currently logged in to AOL Instant Messenger.

Listing 22.3 An LDAP `finger` Lookup for `smith`

```
finger smith@example.com
```

```
[example.com]
```

```
Example.Com LDAP Finger Service
```

```
Login name: dsmith
```

```
In real life: Daniel Smith
```

```
E-Mail: dsmith@example.com
```

```
Phone: +1 408 555 9519
```

```
Department: Human Resources
```

```
Room: 0368
```

```
One entry matched 'smith'
```

Listing 22.4 An LDAP finger Lookup for bjensen

```
finger bjensen@example.com
```

```
[example.com]
```

```
Example.Com LDAP Finger Service
```

```
Login name: bjensen
```

```
In real life: Barbara Jensen
```

```
E-Mail: bjensen@example.com
```

```
Phone: +1 408 555 1862
```

```
Department: Product Development
```

```
Room: 0209
```

```
AIM ID: Babs Jensen (status: ONLINE)
```

```
Address: 1234 Main St. $ Mountain View, CA $ 94043
```

```
URL: http://homepages.example.com/bjensen My Home Page
```

```
One entry matched 'bjensen'
```

[Listing 22.5](#) shows one more example. Four entries match the query string "carter"; the finger LDAP gateway returns information about all of them.

Listing 22.5 An LDAP finger Lookup That Returns Multiple Entries

```
finger carter@example.com
```

[example.com]

Example.Com LDAP Finger Service

Login name:	scarter	In real life:	Sam Carter
E-Mail:	scarter@example.com	Phone:	+1 408 555 4798
Department:	Accounting	Room:	4612
AIM ID:	Sam Carter Ex (status: OFFLINE)		

Login name:	scarte2	In real life:	Stephen Carter
E-Mail:	scarte2@example.com	Phone:	+1 408 555 6022
Department:	Product Development	Room:	2013

Login name:	kcarter	In real life:	Karen Carter
E-Mail:	kcarter@example.com	Phone:	+1 408 555 4675
Department:	Human Resources	Room:	2320
AIM ID:	KarenCarter42 (status: OFFLINE)		

Login name:	mcarter	In real life:	Mike Carter
E-Mail:	mcarter@example.com	Phone:	+1 408 555 1846
Department:	Accounting	Room:	3819
AIM ID:	MikeCarterCPA (status: ONLINE)		

4 entries matched 'carter'

The Source Code

The `lfingerd.pl` source code consists of a single script, which is presented here in pieces to aid understanding. [Listing 22.6](#) shows the prelude and main module.

Listing 22.6 The `lfingerd.pl` Prelude and Main Module

```
1. #!/usr/local/bin/perl
```

```
2. #

3. # lfingerd -- a Perl 5 script that implements the server side

4. #   of the BSD finger protocol using an LDAP server as its

5. #   database of user information. Also returns AIM

6. #   online/offline status if Netscape Directory Server 6 or

7. #   later is used.

8. #

9. # From the 2nd Edition of the book:

10. # "Understanding and Deploying LDAP Directory Services"

11. # by Timothy A. Howes, Mark C. Smith, and Gordon S. Good.

12. #

13. # Requires: PerLDAP

14. #

15.

16. use Mozilla::LDAP::Conn;

17.

18. # Banner:

19. $banner = "Example.Com LDAP Finger Service";

20.

21. # LDAP server information:

22. $ldapbase = "dc=example,dc=com";

23. $ldaphost = "ldap.example.com";

24. $ldapport = "389";

25.

26. # Constants:

27. $unknown = "???";

28. $separator = "\n";

29. @attrlist = ( "cn", "uid", "mail", "departmentNumber",

30.   "telephoneNumber", "roomNumber", "postalAddress",

31.   "ou", "title", "displayName", "description",

32.   "labeledURI", "nsAIMID", "nsAIMStatusText" );
```

```
33.

34. # Start of main:

35. $| = 1;      # enable autoflush of output upon print

36. print "$banner\n\n";

37. $| = 0;      # disable autoflush

38.

39. # grab query string and chop off newline and return characters

40. $query = <STDIN>;

41. chop $query;

42. if ( $query =~ /\r$/ ) {

43.     chop $query;

44. }

45.

46. # form a filter using the query string

47. if ( $query =~ /.*=.*/ ) {

48.     $filter = "(&(objectClass=person)($query))";

49. } else {

50.     $filter = "(&(objectClass=person)"

51.             . "(|(sn=$query)(cn=$query)(uid=$query)))";

52. }

53.

54.

55. # open an anonymous connection to the LDAP server

56. $ldap = new Mozilla::LDAP::Conn( $ldaphost, $ldapport );

57. die "Unable to connect to server at "

58.     . "ldap://$ldaphost:$ldapport\n" unless $ldap;

59.

60. # search the directory

61. $entry = $ldap->search( $ldapbase, "subtree", $filter,

62.                         0, @attrlist );
```

```

63.

64. # display all the results

65. if ( $entry ) {

66.     $count = 0;

67.     do {

68.         displayEntry( $entry );

69.         $entry = $ldap->nextEntry;

70.         ++$count;

71.     } while ( $entry );

72.

73.     if ( $count > 1 ) {

74.         print $count, " entries matched '$query'\n";

75.     } else {

76.         print "One entry matched '$query'\n";

77.     }

78.

79. } else {

80.     print "No entries matched '$query'\n";

81. }

82.

83. # close LDAP connection and clean up

84. $ldap->close;

85.

86. # End of main.

87.

```

The following list summarizes the actions performed in [Listing 22.6](#):

1. The LDAP server information is specified on lines 21 to 24. Constants used elsewhere in the script are defined on lines 26 to 32.
2. When executed, the main module displays a banner string (defined on line 19 and printed by lines 35 to 37) that lets the user know he has connected to an LDAP finger

service.

3. The query string sent by the client is read and cleaned up. In the finger protocol, the client sends a text query string that is terminated with a newline character and a carriage return. The Perl code on lines 39 to 44 reads the query string from standard input and removes any end-of-line characters (inetd always arranges for protocol data sent by clients to be passed to standard input).
4. The query string is used to form an LDAP search filter (lines 46–52). If the query string contains an equal sign, it is assumed to be an LDAP filter component; otherwise a filter that specifies an exact match on `sn`, `cn`, or `uid` is created. In either case, an (`objectClass=person`) component is included to limit the search results to people entries.
5. The code on lines 55 to 84 uses PerLDAP calls to connect to the LDAP server, perform a search, and display all of the entries found. No authentication is done; all queries are anonymous.

[Listing 22.7](#) shows the `displayEntry` subroutine, which is used to produce a nicely formatted entry display. This routine is straightforward. It uses simple `printf` statements to produce a formatted display of an LDAP entry. The resulting text is sent to standard output, which the Unix inetd process has arranged to be sent back to the finger client.

Listing 22.7 The `lfingerd.pl` `displayEntry` Subroutine

```
88.  
89. # Start of displayEntry:  
90. sub  
91. displayEntry {  
92.     local( $entry ) = @_;  
93.     local( $value, @attrs );  
94.  
95.     @attrs = ( "displayName", "cn" );  
96.     printf( "Login name: %-8s" "  
97.             . " In real life: %s\n" ,  
98.             getSimpleValue( $entry, "uid" ),  
99.             getFirstValue( $entry, *attrs ));  
100.  
101.    printf( "E-Mail:      %-32s Phone: %s\n" ,  
102.              getSimpleValue( $entry, "mail" ),  
103.              getSimpleValue( $entry, "telephoneNumber" ));
```

```
104.  
105.    @attrs = ( "ou", "departmentNumber" );  
106.    printf( "Department: %-32s Room: %s\n",
107.              getFirstValue( $entry, *attrs ),
108.              getSimpleValue( $entry, "roomNumber" ) );  
109.  
110.    $value = getSimpleValue( $entry, "nsAIMID" );  
111.    if ( $value ne $unknown ) {  
112.        printf( "AIM ID:      %s (status: %s)\n", $value,
113.                  getSimpleValue( $entry, "nsAIMStatusText" ) );  
114.    }  
115.  
116.    $value = getSimpleValue( $entry, "postalAddress" );  
117.    if ( $value ne $unknown ) {  
118.        print "Address:      $value\n";
119.    }  
120.  
121.    $value = getSimpleValue( $entry, "title" );  
122.    if ( $value ne $unknown ) {  
123.        print "Title:      $value\n";
124.    }  
125.  
126.    $value = getSimpleValue( $entry, "description" );  
127.    if ( $value ne $unknown ) {  
128.        print "Description: $value\n";
129.    }  
130.  
131.    displayAllValues( $entry, "labeledURI", "URL:           " );
132.  
133.    print $separator;
```

```
134. }
135. # End of displayEntry.
136.
```

[Listing 22.8](#) shows the code for three subroutines that are called from `displayEntry`: `displayAllValues`, `getSimpleValue`, and `getFirstValue`. All of these display subroutines access values from an LDAP entry by using PerlLDAP's attribute hash array.

Listing 22.8 Additional `lfingerd.pl` Display Subroutines

```
137.
138. # Start of displayAllValues:
139. sub
140. displayAllValues {
141.     local( $entry, $attr, $prefix ) = @_;
142.     local( $value );
143.
144.     if ( $entry->{$attr} ) {
145.         foreach $value (@{$entry->{$attr}}) {
146.             print $prefix, $value, "\n";
147.         }
148.     }
149. }
150. # End of displayAllValues.
151.
152. # Start of getSimpleValue:
153. sub
154. getSimpleValue {
155.     local( $entry, $attr ) = @_;
156.     local( $value );
157.
158.     if ( $entry->{$attr} ) {
```

```

159.         $value = $entry->{$attr}[0];
160.     } else {
161.         $value = $unknown;
162.     }
163.
164.     $value;
165. }
166. # End of getSimpleValue.
167.
168. # Start of getFirstValue:
169. sub
170. getFirstValue {
171.     local( $entry, *attrs ) = @_;
172.     local( $a );
173.     local( $value );
174.
175.     $value = $unknown;
176.
177.     foreach $a (@attrs) {
178.         $value = getSimpleValue( $entry, $a );
179.         last if ( $value ne $unknown );
180.     }
181.
182.     $value;
183. }
184. # End of getFirstValue.

```

Finally, the system's `/etc/inetd.conf` configuration file must be modified to tell inetd to execute the `lfingerd.pl` script instead of the standard `in.fingerd` executable. [Listing 22.9](#) shows the altered portion of the `/etc/inetd.conf` file used on a Sun Solaris Unix system.

Listing 22.9 Changes to `/etc/inetd.conf` to Enable the `lfingerd.pl` Script

```
1. #finger stream  tcp  nowait  nobody  /usr/sbin/in.fingerd          in.fingerd  
2. finger  stream  tcp  nowait  nobody  /usr/local/etc/lfingerd.pl  lfingerd.pl
```

Line 1, which is commented out, is the line originally used to invoke the standard `in.fingerd` executable. Line 2 is the line that must be added to invoke the new `lfingerd.pl` Perl script; this line assumes the script is installed in the `/usr/local/etc` directory. Don't forget to restart the `inetd` process after making these changes.

Ideas for Improvement

Many things could be done to improve the LDAP finger gateway. Here are some ideas:

- Sort the entries before displaying them. Sorting would be helpful when more than a few entries are found.
- Improve the display of special attributes. For example, `postalAddress` attributes use the dollar sign (\$) character as a line separator, but the `lfingerd.pl` script does not take this into account.
- Improve the generation of search filters. As written, the script does not examine the query string at all; it just searches for all entries whose surname, full name, or user ID exactly matches the query string. A more intelligent script might examine the query string and generate different filters depending on its content. For example, a query string that consists only of numbers might trigger a search by telephone number.
- Enhance the `lfingerd.pl` script to log all finger client connections and queries to a file and consider whether additional measures should be taken to enhance security. Although all access to the directory through the gateway is unauthenticated, `lfingerd.pl` does open another channel through which unscrupulous hackers may attack your directory service.

Example 2: Adding LDAP Address Lookup to an E-Mail Client

Most e-mail clients include a private address book to store addresses of people with whom users correspond on a regular basis. A useful extension of a private address book is an LDAP address lookup feature that allows users to search a centralized directory service when addressing e-mail messages. This example shows how to directory-enable the ICEMail client by adding an LDAP lookup feature.

ICEMail is a pure Java Internet e-mail client that is developed and maintained by a group of open-source developers. Tim Endres (previously with ICE Engineering; now president of SecureByDesign.com) is the original author of ICEMail. Jeff Gay is now the chief organizer for the project. The source code for ICEMail is available under the terms of the GNU General Public License. The changes shown here are based on the ICEMail 3.5.1 release. For more information, including how to obtain the source code, visit the ICEMail Web site at <http://www.icemail.org>.

The Integration Approach

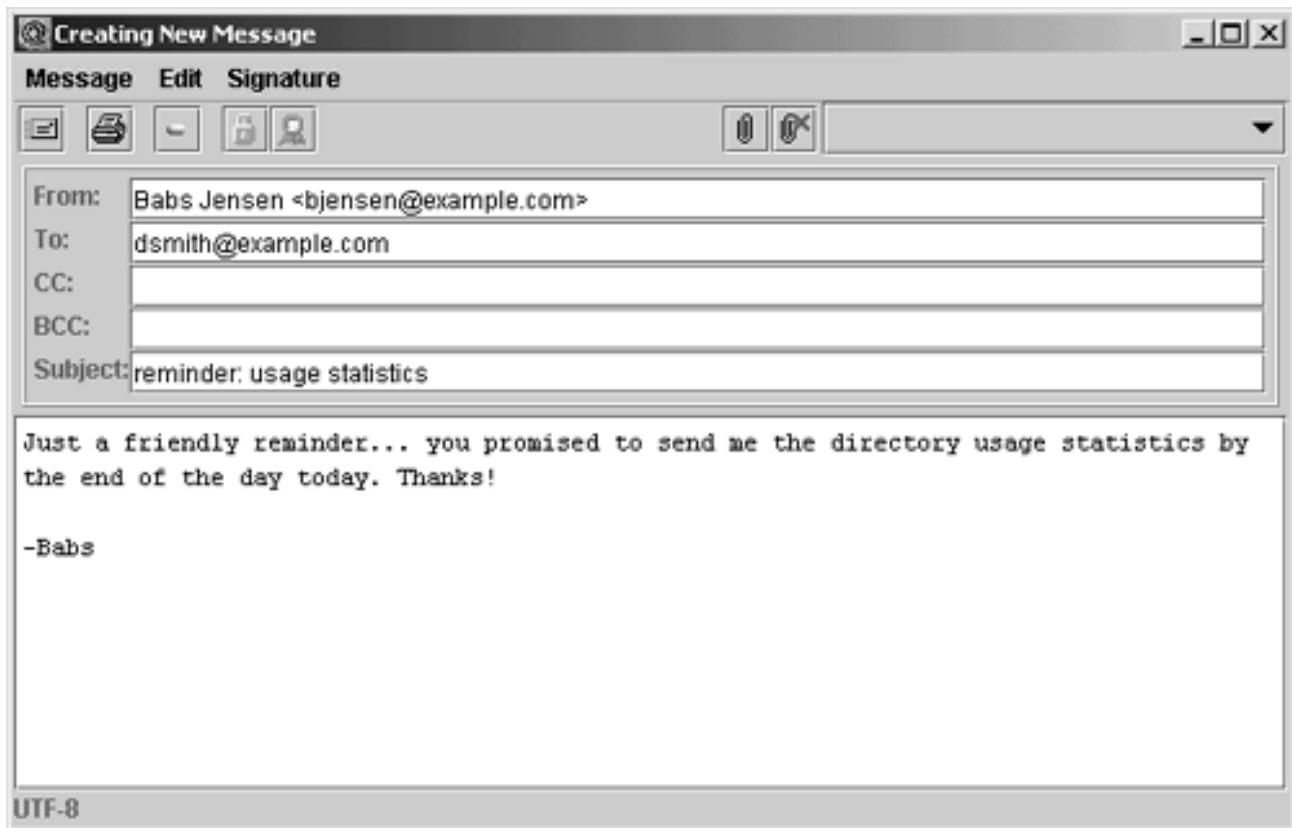
ICEMail has an address book, but it is very basic, so it has no LDAP features and it can't be used to address e-mail messages. To provide a convenient LDAP lookup feature, the decision was made to change the ICEMail message composition window to add a **Directory Lookup** command to the existing **Address** menu. The **Directory Lookup** command will grab the content of the active addressing field (**To**, **CC**, or **BCC**), look it up in the directory, and replace it with a person's e-mail address, if one was found. This is an example of enabling a new feature in an existing application using LDAP.

To accomplish this tight integration, the ICEMail client source code was modified directly. Because ICEMail is a pure Java application, the modifications were written in Java. ICEMail already uses the Netscape LDAP Java SDK to support LDAP Data Interchange Format (LDIF) storage of the ICEMail address book, so the Netscape SDK was the obvious choice for implementation of the new LDAP address lookup feature. The Netscape LDAP Java SDK closely tracks the emerging Internet Engineering Task Force (IETF) Java LDAP API standard.

The End-User Experience

The original ICEMail composition window is shown in [Figure 22.7](#). Note that a recipient's exact e-mail address must be typed into the **To**, **CC**, or **BCC** field when a message is being addressed.

Figure 22.7. The Original ICEMail Composition Window



There is no graphical user interface (GUI) for configuring the LDAP directory within the LDAP-enabled ICEMail application. Therefore, to enable the LDAP lookup feature, a set of lines like the ones shown in [Listing 22.10](#) should be added to a user's `icemailrc.txt` properties file.

Listing 22.10 A Sample Addition to `icemailrc.txt`

1. ICEMail.ldapDirectory=ldap://ldap.example.com:389/dc%3Dexample%2Cdc%3Dcom
2. ICEMail.ldapSizeLimit=25
3. ICEMail.ldapTimeLimit=30

On line 1, the portion to the right of the equal sign (=) is an LDAP URL that `ldap.example.com`, port `389`, and URL-encoded base DN `dc=example,dc=com`). Lines 2 and 3 are optional. They specify LDAP search size and time limits (in entries and seconds, respectively).

[Figure 22.8](#) shows the revised composition window, with the **Address** menu pulled down to show the new lookup command. Notice that the name of the recipient (Daniel Smith) has been typed into the **To** field, and the **Directory Lookup** menu command is about to be executed.

Figure 22.8. The LDAP-Enabled ICEMail Composition Window

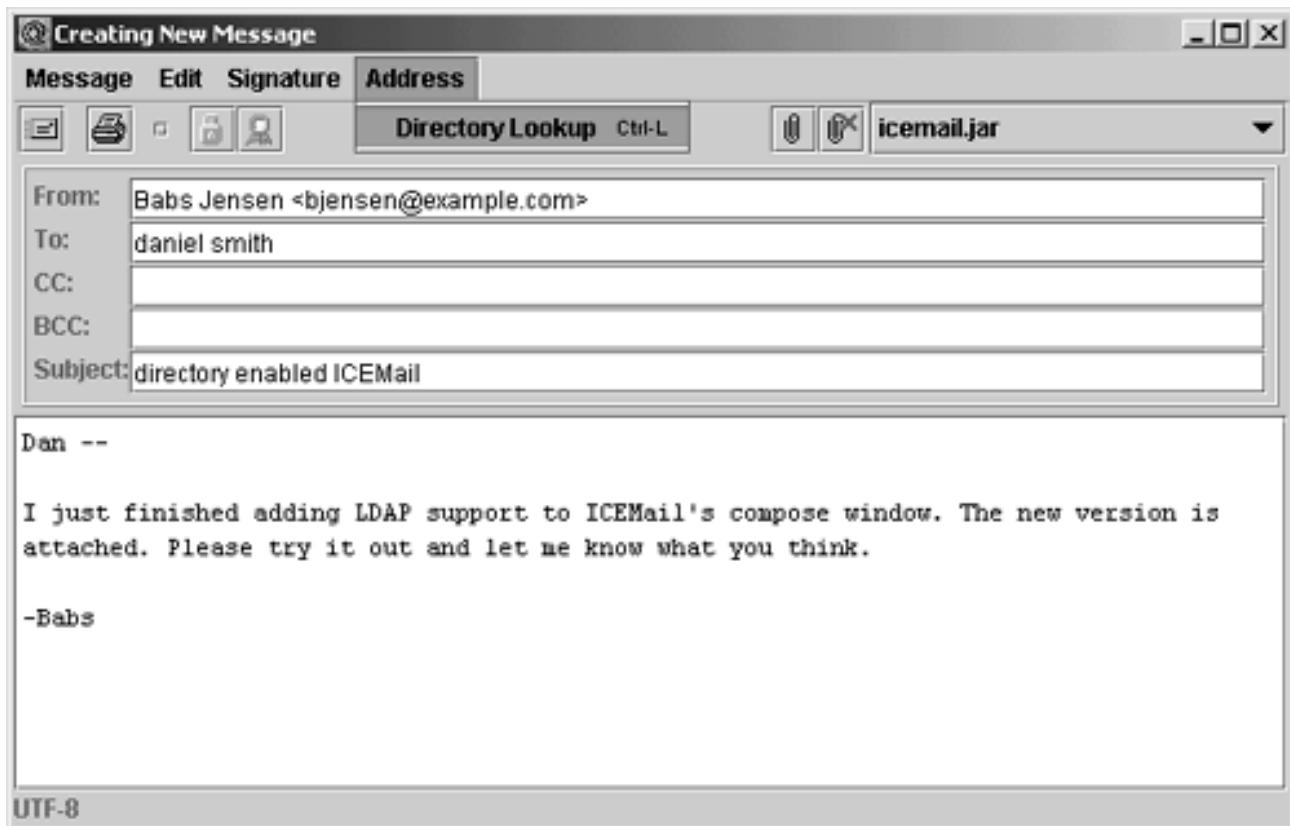
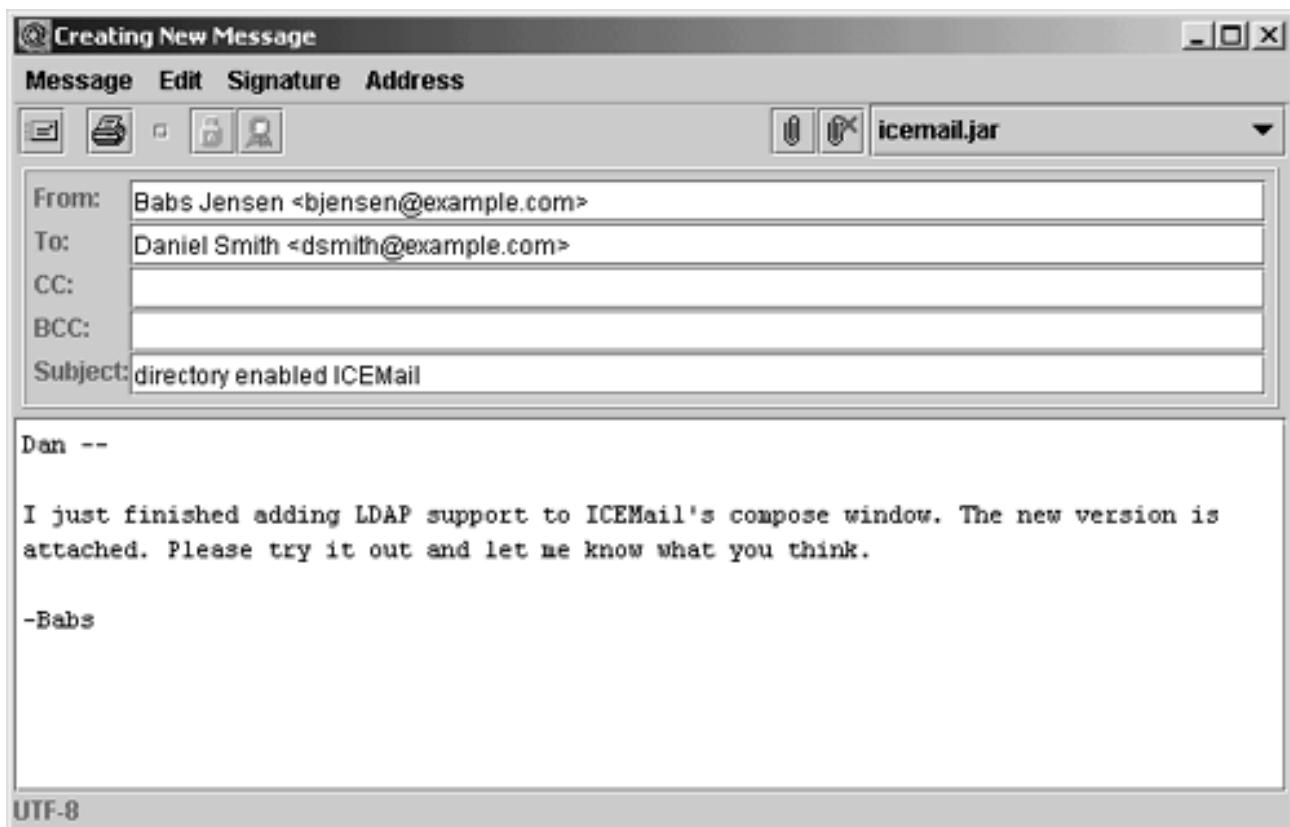


Figure 22.9 shows the result of the LDAP lookup: The name in the **To** field is replaced with Daniel Smith's e-mail address, as found in the `example.com` directory.

Figure 22.9. The ICEMail Composition Window after a Successful LDAP Lookup



ICEMail is a fairly large application with a friendly GUI built on Sun's Java Foundation Classes (JFC). ICEMail's existing `ComposeFrame` class was modified to add the LDAP lookup feature. This class is responsible for displaying the message composition (new message) window and interacting with the user. The source code for the `ComposeFrame` class is in the file `java/org/icemail/mail/ComposeFrame.java`.

In [Listings 22.11](#) through [22.14](#), lines with an ellipsis character (...) indicate a place where existing ICEMail code has been omitted to save space. Each listing includes the declaration of the class or method to which the new code belongs; this way, you can see exactly where the new code fits within the ICEMail source code. Existing code that surrounds the code changes and additions is included for context. In listings that show both existing and new code, the original ICEMail code is shown without line numbers, and the new code is shown in bold type.

[Listing 22.11](#) shows the first code change. The two new import statements were added to gain access to the Netscape LDAP classes.

Listing 22.11 Changes to the Import Section of ICEMail's `ComposeFrame` Class

```
...  
  
import org.icemail.util.RotateButton;  
  
import org.icemail.util.StringUtilities;  
  
import org.icemail.util.UserProperties;  
  
  
1. import netscape.ldap.*;  
  
2. import java.net.MalformedURLException;  
  
  
/**  
 * This class composes and edits messages,  
 * allowing messages to be sent, printed, and saved.  
...  

```

When an ICEMail `ComposeFrame` object is created, it reads some configuration information and stores it in member variables for later use by the class's methods. The new LDAP directory configuration information, which is stored in the `icemailrc.txt` properties file, is read and placed in a member variable so that it can be accessed conveniently.

[Listing 22.12](#) shows the definition of a new member variable, named `ldapDirectoryURL`. Because the directory configuration can easily be represented as an LDAP URL, a variable of type `LDAPUrl` is used to hold it (the `LDAPUrl` class is part of the Netscape LDAP Java API).

Listing 22.12 An Addition to `ComposeFrame`'s Private Member Variables

```

...
private IActionListener    actionListener_ = new IActionListener();
private IFocusListener     focusListener_ = new IFocusListener();
private JTextComponent      currentFocus_ = null;

1. private LDAPUrl         ldapDirectoryURL = null;

/*
 * Default constructor, creating a composition frame for a new message.
 */
public
ComposeFrame() {
...

```

As part of the initialization of a `ComposeFrame` object, the ICEMail code calls a method named `establishAddresses()` to create the **Address** menu. If an LDAP directory URL is present in a user's `icemailrc.txt` properties file, a **Directory Lookup** item must be added to the **Address** menu. [Listing 22.13](#) shows the required revisions to the `establishAddresses()` method.

Listing 22.13 Revisions to `ComposeFrame`'s `establishAddresses()` Method

```

...
private void
establishAddresses() {
    JMenuItem item;
    String addrListStr = UserProperties.getProperty( "addressList", null );
1.    /*
2.     * If an LDAP directory is configured, include a
3.     * "Directory Lookup" item in the Address menu.
4.     */
5.    String ldapURLStr = UserProperties.getProperty( "ldapDirectory", null );
6.    if ( ldapURLStr != null ) {
7.        try {

```

```
8.         ldapDirectoryURL = new LDAPUrl( ldapURLStr );
9.
10.    } catch ( MalformedURLException e ) {
11.
12.        JOptionPane.showMessageDialog( null,
13.                                         "Malformed LDAP URL: " + ldapURLStr );
14.
15.    }
16.
17.    String[] addrList = {};
18.
19.    if ( addrListStr != null )
20.
21.        addrList = StringUtilities.splitString( addrListStr, ":" );
22.
23.    if ( addrList.length < 1 && ldapDirectoryURL == null )
24.
25.        return;
26.
27.    JMenu menu = ComponentFactory.getMenu( ICEMail.getBundle(),
28.
29.                                         "Compose.Address" );
30.
31.    menuBar_.add( menu );
32.
33.    if ( ldapDirectoryURL != null ) {
34.
35.        JMenuItem item = new JMenuItem( "Directory Lookup" );
36.
37.        item.addActionListener( actionListener_ );
38.
39.        item.setActionCommand( "LDAP:LOOKUP" );
40.
41.        item.setAccelerator( KeyStroke.getKeyStroke( 'L',
42.
43.                                         Event.CTRL_MASK ) );
44.
45.        menu.add( item );
46.
47.        if ( addrList.length > 0 ) menu.addSeparator();
48.
49.    }
50.
51.    for ( int i = 0 ; i < addrList.length ; ++i ) {
52.
53.        String addrStr = UserProperties.getProperty( ("address." + addrList[i]),
54.
55.                                         null );
56.
57.        ...
58.    }
59.
```

The code on lines 5 to 13 retrieves the `ldapDirectory` URL from the user's properties file and initializes the `ldapDirectoryURL` member variable. The code on lines 15 to 24 is identical to that in the original ICEMail source code, except some additional checks were added to test whether `ldapDirectoryURL` is NULL (the **Address** menu is not created at all if no LDAP directory and no listed addresses are configured). The code on lines 26 to 34 is new code that adds the **Directory Lookup** menu item and associates the `LDAP:LOOKUP` action with the menu item.

The `actionPerformed()` method is invoked to handle actions that are triggered by menu selections or similar events. [Listing 22.14](#) shows the code that was added to support the new LDAP lookup feature.

Listing 22.14 An Addition to ComposeFrame's actionPerformed() Method

```
...
public void
actionPerformed( ActionEvent event ) {
...
} else if ( command.equals( "LOWPRIORITY" ) ) {
    headers_.setHeader( "Importance", "low" );
    headers_.setHeader( "X-Priority", "5" );
1. } else if ( command.equals( "LDAP:LOOKUP" ) ) {
2.     ldapLookup();
} else {
    System.err.println( "ComposeFrame.actionPerformed() - unknown command '" +
                        command + "' " );
}
...
...
```

The LDAP searches themselves are performed by the `ldapLookup()` method, which is the final addition to the `ComposeFrame` class. [Listing 22.15](#) shows the `ldapLookup()` source code.

Listing 22.15 The ComposeFrame ldapLookup() Method

```
1. /*
2. * ldapLookup: grab the contents of the current text entry field,
3. * look it up in the configured LDAP directory, and replace the
4. * original text with DISPLAYNAME <MAIL> or CN <MAIL>.
```

```
5.  */
6. private void
7. ldapLookup() {
8.     String ldAttrs[] = { "cn", "displayName", "mail" };
9.     LDAPConnection ld = null;
10.
11.    // Get text from the active text entry field
12.    if ( currentFocus_ == null ) {
13.        return;
14.    }
15.    String textToLookup = currentFocus_.getSelectedText();
16.    boolean useSelection = ( textToLookup != null );
17.    if ( !useSelection ) {
18.        textToLookup = currentFocus_.getText();
19.    }
20.    if ( textToLookup.length() <= 0 ) {
21.        return;
22.    }
23.
24.    // Create filter and execute an LDAP search
25.    String ldFilter = "(&(objectClass=person)";
26.    if ( textToLookup.indexOf( "=" ) != -1 ) {
27.        ldFilter += " (" + textToLookup + " ) ";
28.    } else {
29.        ldFilter += "( | (cn=" + textToLookup + " ) "
30.                    + "(uid=" + textToLookup + " ) "
31.                    + "(sn=" + textToLookup + " ) ) ";
32.    }
33.
34.    try {
35.        ld = new LDAPConnection();
```

```
36.     ld.connect( ldapDirectoryURL.getHost() ,
37.                   ldapDirectoryURL.getPort() ) ;
38.
39.     LDAPSearchConstraints cons = new LDAPSearchConstraints();
40.     int sizelimit = UserProperties.getProperty( "ldapSizeLimit", 100 );
41.     int timelimit = UserProperties.getProperty( "ldapTimeLimit", 60 );
42.     cons.setMaxResults( sizelimit );           // entries
43.     cons.setServerTimeLimit( timelimit );      // seconds
44.
45.     LDAPSearchResults res = ld.search(
46.             ldapDirectoryURL.getDN(),
47.             LDAPConnection.SCOPE_SUB, ldFilter, ldAttrs, false,
48.             cons );
49.
50.     // Process the search results
51.     int         count = 0;
52.     LDAPEntry   entry = null;
53.
54.     while ( res.hasMoreElements() ) {
55.         try {
56.             entry = res.next();
57.             ++count;
58.         } catch ( LDAPException e ) {
59.             JOptionPane.showMessageDialog( null,
60.                                         "LDAP res.next error: " + e.errorCodeToString());
61.         }
62.     }
63.
64.     if ( count <= 0 ) {
65.         JOptionPane.showMessageDialog( null,
66.                                     "No entries matched " + textToLookup );
67.     } else if ( count > 1 ) {
```

```
68.     JOptionPane.showMessageDialog( null,
69.             count + " entries matched " + textToLookup );
70. } else {
71.     /*
72.      * Exactly one match... replace contents of current field
73.      * (or current selection) with:
74.      *     displayName <mail>
75.      * or:
76.      *     cn <mail>
77.     */
78.     String newAddress;
79.     Enumeration enum;
80.     LDAPAttribute attr = entry.getAttribute( "displayName" );
81.     if ( attr == null ) {
82.         attr = entry.getAttribute( "cn" );
83.     }
84.     if ( attr != null ) {
85.         enum = attr.getStringValues();
86.         newAddress = (String)enum.nextElement() + " ";
87.     } else {
88.         newAddress = "";
89.     }
90.
91.     attr = entry.getAttribute( "mail" );
92.     enum = attr.getStringValues();
93.     newAddress += "<" + (String)enum.nextElement() + ">";
94.     if ( useSelection ) {
95.         currentFocus_.replaceSelection( newAddress );
96.     } else {
97.         currentFocus_.setText( newAddress );
98.     }
```

```

99.      }
100.
101.     } catch ( LDAPException e ) {
102.         JOptionPane.showMessageDialog( null,
103.             "LDAP connect or search error: "
104.             + e.errorCodeToString());
105.     }
106.
107.     // Clean up -- close the LDAP connection
108.     if ( ld != null && ld.isConnected() ) {
109.         try {
110.             ld.disconnect();
111.         } catch ( LDAPException e ) {
112.             /* ignore */
113.         }
114.     }
115. }

```

Although lengthy, the `ldapLookup()` code is straightforward. After some local variables are declared (lines 8 and 9), the code on lines 11 to 19 retrieves the text that is used to search and places it in a local variable named `textToLookup`. The text to look up is taken from the active text input field within the composition window. If any text is selected, it is used; otherwise, the entire content of the text field is used. If there is no text at all, the method returns without doing anything (lines 20–22).

The code on lines 24 to 32 creates an LDAP search filter based on the `textToLookup` value. An `(objectClass=person)` ANDed component is always included. If the text to look up contains an equal sign, it is used directly to form the rest of the filter; otherwise, a filter that exactly matches against `cn`, `sn`, or `uid` is used.

The code on lines 34–48 connects to the LDAP directory and performs one search. The LDAP host, port, and search base are obtained from the `ldapDirectoryURL` private member variable that was initialized by the code examined earlier (see [Listing 22.13](#)). Only three attributes are retrieved from the LDAP server: `cn`, `displayName`, and `mail` (as specified in the `ldAttrs` array). Size and time limit values are retrieved from the user's `icemailrc.txt` properties file; default values of 100 entries and 60 seconds, respectively, are used if the properties file does not contain values.

The code on lines 50 to 62 examines the search results and counts the number of entries returned. The code on lines 64 to 99 handles the three possible outcomes of a successful search. If no entries are found, the code on lines 64 to 66 displays a message box to inform the user. If more than one match is found, a different message box is presented to let the user

know (lines 67–69). The interesting case in which a single matching entry is found is handled by lines 70 to 99. If the entry has an e-mail address, the content of the active text entry field (or the selected text within that field) is replaced with a string formed from the entry's `displayName` or common name (`cn`) attribute and the `mail` attribute. The remainder of the code in the `ldapLookup()` method is concerned with error handling and cleanup.

Ideas for Improvement

Many things could be done to improve the LDAP lookup feature in ICEMail. Here are some ideas:

- Present a list to allow the user to choose the desired entry if more than one match is found. The lack of this capability is the biggest limitation of the example code.
- Improve the generation of search filters. As written, the code in `ldapLookup()` simply searches for entries whose surname, full name, or user ID exactly matches the query string, unless the user types an LDAP search filter (which most people will not know how to do). A more intelligent approach would be to examine the query string and generate different filters depending on its content. For example, a string that consists only of numbers could trigger a search by telephone number. Another approach would be to make the filter configurable by each ICEMail user.
- Provide a GUI for setting the `ldapDirectory`, `ldapTimeLimit`, and `ldapSizeLimit` preferences that are stored in the ICEMail properties file. You could do this by making changes to ICEMail's `Configuration` class.
- Allow entries found via LDAP to be added to the ICEMail address book.
- Support more than one LDAP server.
- Provide feedback to the user while the LDAP lookup is being done. Such feedback might be as simple as displaying an hourglass cursor, or as complex as displaying a dialog with a status message showing the progress of the LDAP search. For example, the dialog might show "Connecting to ldap.example.com" followed by "Executing search..." followed by "Retrieved entry 1..." and so on.

Directory-Enabling Existing Applications Checklist

The following checklist summarizes the steps you should take when directory-enabling existing applications:

- Identify one or more needs that can be met by integration of an existing application with an LDAP directory service.
- Choose the best LDAP development tool for the task at hand.
- Choose an approach to LDAP integration that allows you to meet your goals and satisfy the application's users.
- Design your LDAP integration so that the resulting application coexists well with other directory-enabled applications and the directory service itself.
- Document how the application will be integrated with the directory service, and share that information with the people who maintain the directory service.
- Consider performance and scalability during LDAP integration design and implementation.
- Plan for the transition from the older version of the application to the new LDAP directory-enabled version.

Further Reading

The Java LDAP Application Program Interface (Internet Draft). R. Weltman, C. Tomlinson, M. Kekic, S. Sonntag, J. Sermersheim, T. Howes, and M. Smith, 2001. Available on the World Wide Web at <http://www.ietf.org>.

The Java Programming Language (3rd edition). K. Arnold, J. Gosling, and D. Holmes. Addison-Wesley, 2000.

LDAP Programming with Java. R. Weltman and T. Dahbura, Addison-Wesley, 2000.

Netscape Directory SDK for Java and PerLDAP. Available on the World Wide Web at <http://enterprise.netscape.com>.

PerLDAP source code release. Available on the World Wide Web at <http://mozilla.org/directory/perldap>.

Programming Perl (3rd edition). L. Wall, T. Christiansen, and J. Orwant. O'Reilly, 2000.

Looking Ahead

In addition to the ideas provided in this chapter and in [Chapter 21](#), Developing New Applications, another way to leverage your directory service is to integrate with additional data stores within your organization. This complex topic will be covered in [Chapter 23](#), Directory Coexistence.

Chapter 23. Directory Coexistence

- Why Is Coexistence Important?
- Coexistence Techniques
- Privacy and Security Considerations
- Determining Your Coexistence Requirements
- Directory Coexistence Implementation Considerations
- Example: The Idapsync Tool: One-Way Synchronization with Join
- Directory Coexistence Checklist
- Further Reading
- Looking Ahead

The previous two chapters talked about how existing and new applications can use your directory. The focus of this chapter is *directory coexistence*, the task of integrating other data sources with your directory.

Integrating other data sources requires that you understand the desired relationships between the data in your directory and data maintained elsewhere, and that you create procedures and policies to correctly maintain these relationships. Directory coexistence also allows your directory to complement existing business processes within your organization. Your directory coexistence strategy determines whether your directory is an island—isolated from the rest of your enterprise—or is integrated and coexists peacefully with your existing infrastructure and business processes.

Directory coexistence is often the biggest problem that directory administrators face. Not addressing this problem can lead to user complaints, spiraling data management costs, and an ineffective and unpopular directory service. Therefore, it is important to address this issue in your directory deployment (or else at least to convince yourself that you don't need to).

There are many reasons to integrate or link data in your directory with data held in other data sources. Here are some examples:

- You have existing data in another database that you want to make available to directory applications.
- You want to aggregate widely used information in a logically centralized directory service (or in another database that is widely used) so that people can access it in one place.
- You want to provide a central management point for data, reducing the cost of maintaining it.
- You want to use the directory as the authoritative source for new data elements that are used to populate other databases.
- You want to use the directory as a conduit to send data between other systems that otherwise could not share any data.

As these examples suggest, it is rare to find a directory service that does not need to integrate with any other data sources.

The correct approach for achieving directory coexistence depends on your requirements, the capabilities of your directory, and the capabilities and business processes associated with the other data sources you need to integrate. In some situations, a simple one-time population of the directory from another source (or vice versa) may be all that is needed. In other cases you may need to set up an ongoing two-way relationship between your directory and a data source, in which data may be updated by either one. These two examples are extremes; your actual requirements will probably lie somewhere in between.

Whatever your needs, there are a variety of techniques for accomplishing integration and coexistence, ranging from homegrown scripts and programs to off-the-shelf software. After you determine your needs, you'll need to figure out whether any off-the-shelf software fits the bill. Chances are that even if you do find suitable software, you will still need to develop some custom tools and scripts. Directories, databases, and the environments in which they run are so variable that it is unlikely that an off-the-shelf software package will meet all of your needs.

This chapter takes you through the process of determining your directory coexistence needs and how to meet them. You will learn about the available techniques for establishing coexistence, the data sources with which your directory needs to coexist, the kind of integration you need, and how to accomplish it. You will learn to distinguish among directory migration, directory synchronization, metadirectories, and virtual directories—all of which are tools that help accomplish directory coexistence. The chapter ends with an example of a homegrown directory coexistence tool written in Perl.

Often it is difficult and time-consuming to achieve coexistence, and many organizations underestimate the amount of work involved. This area is both technically and politically challenging (see [Chapter 6](#), Defining Your Directory Needs, for a discussion of political considerations). But coexistence with other data sources makes a directory service more valuable.

Why Is Coexistence Important?

In [Chapter 18](#), Maintaining Data, we discussed the importance of managing directory data to keep it accurate and up-to-date. The reasons to do so are simple, and they all boil down to increasing the usefulness of the directory service. The arguments for directory coexistence are similar, but they are motivated by the fact that you may have many data sources that contain useful, related data. Chances are that many directories, databases, and other data sources are already deployed in your organization. Here is a list of common data sources:

- **Network operating system (NOS) directories.** These include most deployments of Novell eDirectory, Microsoft Active Directory, and Unix's Network Information Service (NIS). These directories typically contain minimal information about users (such as name, login ID, and password) and information about devices and services related to operating systems (such as printers and file servers). The main job of these directories is to serve the needs of the network operating system in which they are embedded. Although quite adept at that task, most of these directories make poor general-purpose directories because of their lack of flexibility and specialized feature set; certainly it is difficult to use a single deployment of these directories as an enterprise-wide directory and as the operating system directory. However, NOS directories often do contain information that might be useful in an enterprise or extranet directory, such as people's names and login IDs.
- **Application-specific directories.** These include the Lotus Notes name and address book, Microsoft's Exchange directory, Novell's GroupWise directory, the built-in authentication databases provided with most network services, and other directories that are designed to meet the needs of a single application. These directories sometimes are accessible only via proprietary protocols or application programming interfaces (APIs), seldom are extensible, and like NOS directories, typically make poor general-purpose directories. However, also like NOS directories, application-specific directories may contain information you would like to make available in an enterprise or extranet directory.
- **Corporate databases.** This broad category refers to databases that hold information about people (human resources), telephone operations, customers, payroll, and so on. These databases often contain a lot of interesting and useful information that should be made available through your general-purpose, enterprise directory service. Corporate databases are also usually integrated with existing business processes, from hiring and termination of employees to procurement and financial processes.
- **External databases.** External databases are not maintained within or by your organization; they live outside your corporate firewall (literally or figuratively) and are maintained by another organization. Like corporate databases, these data sources may hold useful information about people and resources—for example, from some of your corporate partners. However, it is rare (but not unheard of) to integrate data sources across corporate boundaries.
- **Homegrown databases.** As the name suggests, this category refers to the inevitable collection of ad hoc databases present in all organizations. Homegrown databases might be maintained by departmental secretaries or local administrators, or they might have been created for projects that have since been abandoned. For example, in many companies a secretary or an administrative assistant maintains a spreadsheet that contains up-to-date information about all employees that work in his department. You will need to determine how many of these databases exist in your organization and whether any of them contain information that should be included in your directory service.

- **The directory service itself.** Some data may be maintained only in your directory service, either by administrators or by end users. If so, new data will flow from various LDAP clients back to your directory servers. For example, you might allow users to update their own contact information, such as home address and telephone number, directly in the directory, or you might ask administrative assistants to maintain departmental information such as fax phone numbers. It might be useful to allow this data not only to be published in the directory, but also to flow back to your human resources database.

All the data sources in your organization serve a purpose (or did at one time). Some of them may be in the process of being phased out; others are there to stay. Why should you be concerned about making your directory service coexist with these other data sources? Here are several reasons:

- **Jump-starting your directory.** One of the problems you face, as discussed in [Chapter 7](#), Data Design, is how to populate your directory with useful data. The best and easiest way to do this is to find another data source that already contains the data you want. Importing data from an existing source can save you substantial time and money in getting your directory populated with data.
- **Ease of data maintenance.** As discussed in [Chapter 18](#), Maintaining Data, it is important to keep the data in your directory accurate and up-to-date. Feeding your directory periodically from existing data sources can help achieve this goal. This arrangement can save you from having to institute complicated update procedures yourself. Instead, you can simply track changes made to the data in the source database.
- **Avoiding duplicated data.** Maintaining data in multiple locations leads to inconsistencies, duplicated effort, increased costs, and a variety of other problems. Using directory coexistence to maintain data in a single location but make that data available everywhere it is needed helps reduce costs and problems.
- **Avoiding user confusion.** Duplicated data often leads to confusion among users. Users often assume (reasonably so) that if they change a piece of information about themselves in one place, the same change will be made in all other locations. Your users will be confused and unhappy if they find that a change of address made in the human resources database needs to be made again in your enterprise directory.

As you can see, directory coexistence is a large, complex area. Developing a good coexistence strategy is not an easy task, but it can pay big dividends.

Coexistence Techniques

There are multiple ways to achieve directory coexistence. In this section the various techniques are described, and information is provided about when to use which technique. Depending on your requirements, you may need to use only one of these techniques or all of them.

A directory system that implements any of these techniques is sometimes broadly called a *metadirectory*. The idea behind the term is that the directory serves as an aggregation point for information that lives in other directories and data sources.

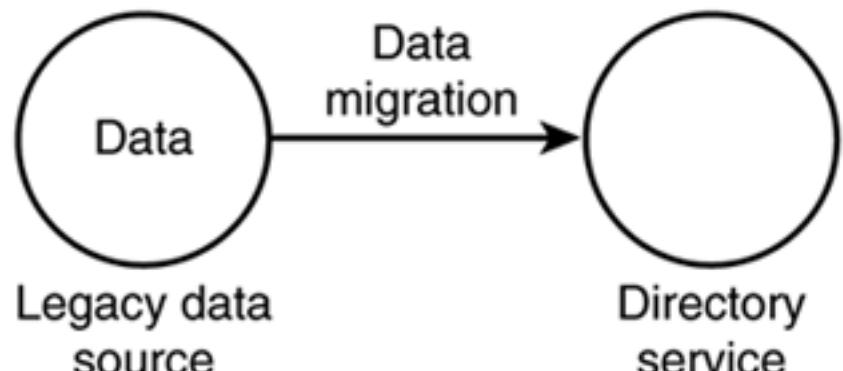
Migration

Data migration is the most rudimentary form of coexistence. We are hard-pressed to even describe it as coexistence because it refers to a one-time event rather than an ongoing process. Nevertheless, it is a good starting point for our discussion.

Data migration is simply a way of populating your directory from a data source. Implicit in the migration concept is the fact that the source is not used after its data is copied into the directory. A good time to use data migration might be when switching from one e-mail system to another. If the old e-mail system has its own application-specific directory containing user and group information, and the new e-mail system uses your enterprise directory, migration is a good way to get the data out of the old system and installed in the new system with minimal disruption and inconvenience to your users.

[Figure 23.1](#) illustrates data migration. You must take care to ensure that once you begin migrating the data, the old system does not accept updates because any such updates will be lost.

Figure 23.1. Data Migration



Most directory products come with migration tools. Some, such as Netscape's LDAP Data Interchange Format (LDIF) and Directory Services Markup Language (DSML) import tools, rely on you to provide a text file of information in a standard format. The system from which you migrate is often able to produce a text file, in either that format or one that is easily convertible. Other tools are specific to a particular application. For example, most directory-enabled mail server software includes tools that migrate data from legacy and competitors' e-mail packages to an LDAP directory service.

One-Way Synchronization

A big step up from data migration in terms of complexity and functionality is one-way data synchronization. *Synchronization* is simply the process of copying data from one data source to another on an ongoing basis. In one-way synchronization, your directory service is periodically populated from a data source. The reverse is also possible: An external database can be populated periodically from the directory. In *one-way synchronization*, external changes to the data are allowed only at the synchronization source, not at the destination. Changes to the data are propagated either by replacement of the entire content of a portion of the directory information tree or by application of only those changes that have occurred since the last update.

The advantage of doing a total replacement is primarily simplicity: You just delete the old data and replace it with the new data. Some older database and directory systems do not have a facility for tracking changes, making it difficult to generate incremental updates and easier just to perform a total replacement. The disadvantage of the total update method is poor performance: For large data sets, it can take a long time to completely re-create the entire directory each time a change is made. Your directory service may even need to be offline during this process.

An incremental synchronization process typically performs much better. If only 5 percent of

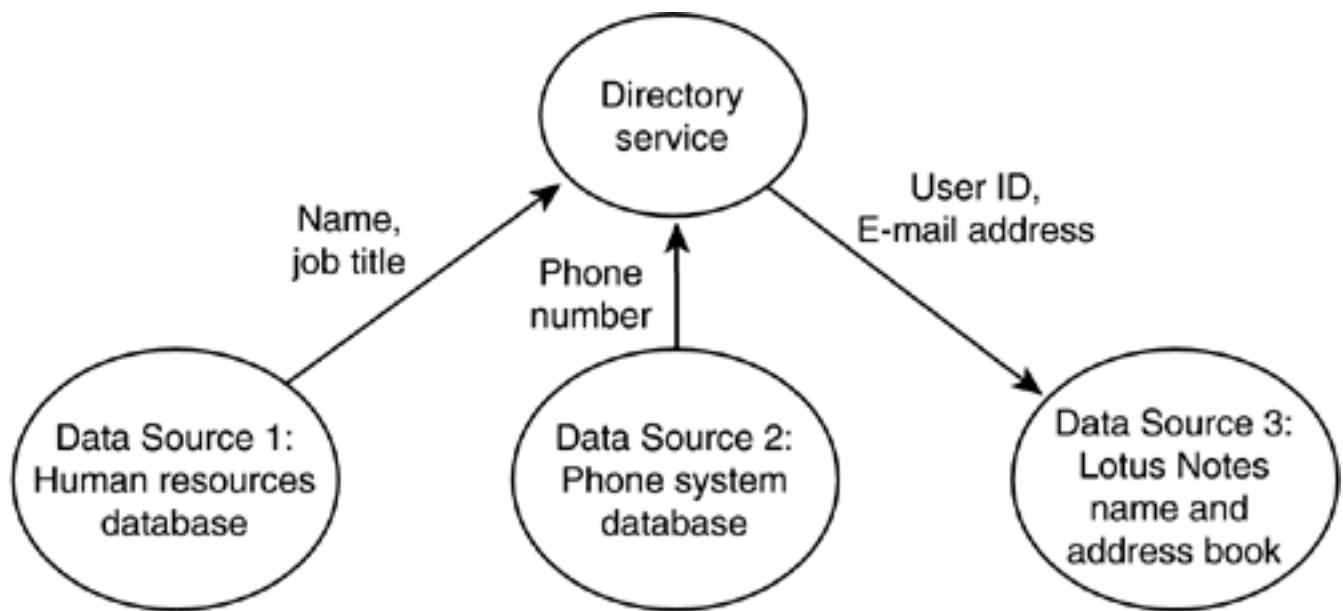
the data changes between updates, an incremental update will be cheaper than a total replacement by a factor of 20. Another advantage of incremental updates is that they are usually done over LDAP while the directory is up and running, a practice that reduces service disruptions and ensures that the regular directory access control checks are used to control access to the data. One more problem is that extra changes made in the directory service may be accompanied by unpleasant side effects. For example, if each person entry is deleted and re-added once each day as part of a total update process, external systems that watch for "new person entry" events may be confused or do more work than is necessary.

Even if the end system with which you synchronize does not support the generation of incremental changes, consider implementing this capability yourself. For example, you could save the last full extract from the system and compare it to the next full extract, thus making it possible to perform an incremental update.

One-way synchronization is often used to extract data from your directory on an ongoing basis and populate other directories. Central control over the data may be maintained while at the same time the data is made available for use in a variety of other systems. One-way synchronization is also often used to extract information from corporate data sources, such as a human resources database, in order to populate a read-only directory. Replicating data from a central source gives directory clients access to the data they need while leveraging the corporate data management procedures you already have in place.

Most directory coexistence plans require several different one-way synchronization relationships. For example, a user's name and job title might be pulled from the human resources (HR) database, whereas the telephone number might be pulled from the phone system database. The directory service itself might send its user ID and e-mail address attributes to several application-specific directories for use in e-mail address books. [Figure 23.2](#) shows an example of multiple one-way synchronization relationships.

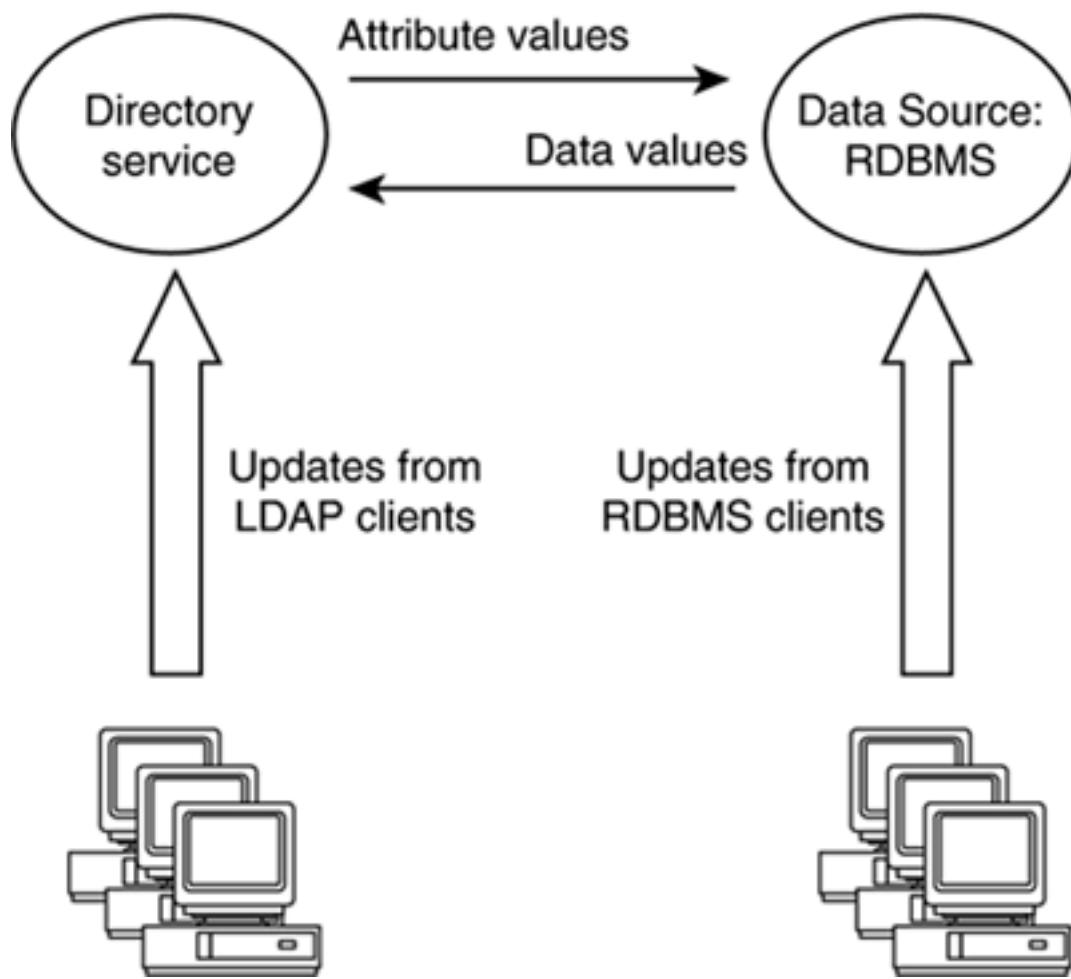
Figure 23.2. Multiple One-Way Synchronization Relationships



Two-Way Synchronization

In *two-way synchronization*, data element changes are propagated in both directions between your directory and another data source. Changes to the data element may be made at either location, and the changes are propagated to all data repositories participating in the synchronization effort. [Figure 23.3](#) shows a graphical view of two-way synchronization.

Figure 23.3. Two-Way Synchronization



The advantage of two-way synchronization over one-way synchronization is that it provides maximum flexibility. There is no need to select a single owner of the data and make other repositories read-only. Instead, every repository can continue to be a read/write source for the data.

An example of data that you might want to maintain in multiple data sources and synchronize in both directions is user password data. It would be nice if user passwords were synchronized across your enterprise directory, NOS directories, and various application directories and if password changes were propagated to all systems. Of course the security implications of synchronizing passwords among multiple systems might prevent you from doing so.

The disadvantages of two-way propagation are its complexity, its occasional unpredictability, and the political difficulties inherent in implementing it. When changes are allowed to be made in more than one system, it is relatively easy for conflicts to arise. A change made to a data element in one location may conflict with a change made at roughly the same time to the same data element in another location. For example, a person's telephone number may be replaced in both systems with a different value. This fact will cause some people who maintain data sources to become very nervous, and they may not like the idea of two-way propagation at all. To maintain a consistent view of the data in all locations, conflicts must be avoided entirely or resolved in a predictable and efficient way.

Resolving update conflicts can be difficult. Even a simple approach, such as serializing access on the basis of synchronized time, requires an additional service to keep the times synchronized—and ties can still occur. Other solutions, perhaps involving a policy-based conflict resolution strategy, can be simpler to implement but often result in unexpected

behavior from the user's point of view. If a user does not understand the conflict resolution policy in place on his system, he may be surprised if a change he makes is overwritten by someone else's change.

Although some circumstances may require two-way synchronization, the added complexity and potential user confusion are usually reason enough to avoid it. If you think you have a specific need for two-way synchronization, be creative and try to think of a way to avoid it. For example, you might deploy a centralized service that allows password changes in only one location and uses one-way synchronization to quickly push those changes out to all other data sources. This kind of system would probably be a minor inconvenience to users, but it would make life a lot easier for everyone in the long term.

Alternatively, consider how you could intercept calls to change passwords at other data sources and reroute them to the centralized service. This approach would give the illusion of two-way synchronization without the same complexity.

N-Way Join

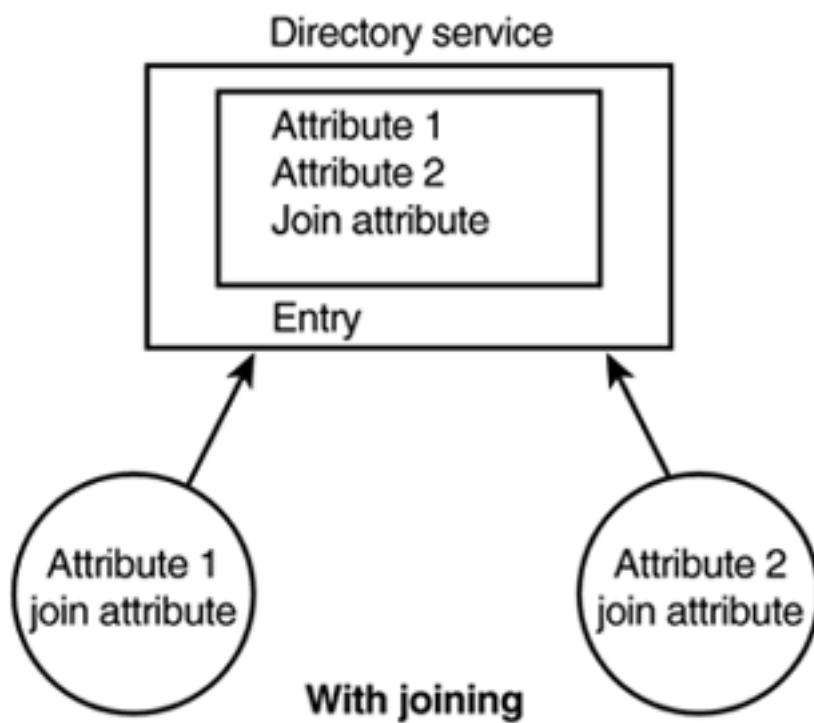
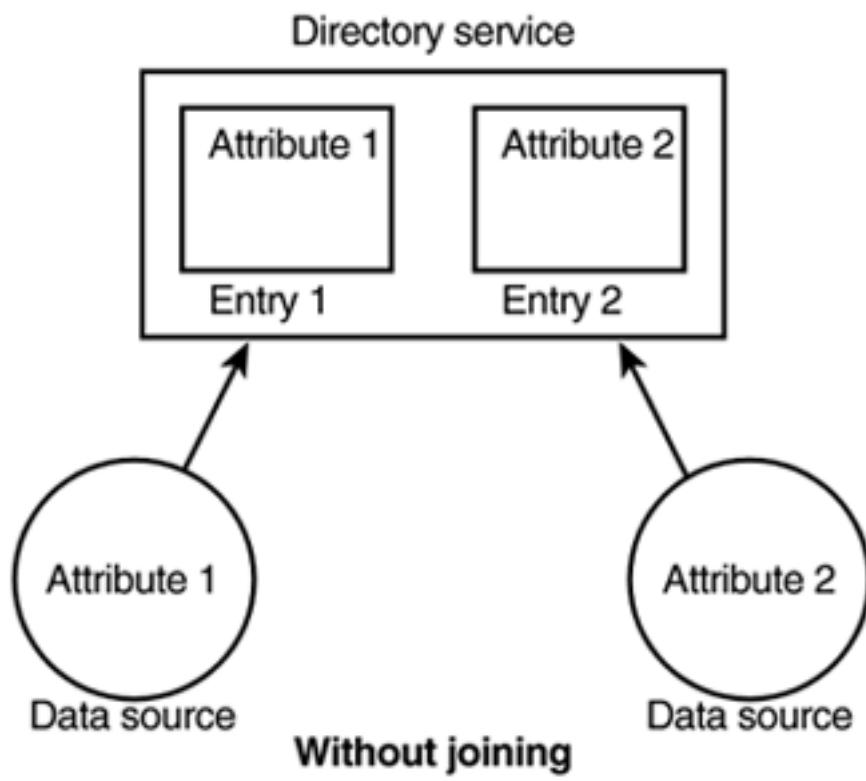
When synchronizing data from multiple sources, usually you want to match up related information from each data source. For example, if your coexistence policy calls for pulling employees' names, job titles, and manager information from the corporate human resources database and telephone numbers from the telephone operations database, you would like to end up with one entry in your directory service that contains all this information for a single person. You do not want to end up with two entries for each person—one from the HR database and one from the telephone operations database. The process of matching up entries from disparate databases is called *joining*, and to many people this capability is what defines a true metadirectory system. Sometimes the term *N-way join* is used to emphasize the fact that data may be joined from an arbitrary number of different data sources.

Note

Several vendors of directory coexistence software sell packages that include the term *metadirectory* in the product name. Some examples are Microsoft Metadirectory Services, Critical Path's CP Meta-Directory Server, and Sun ONE Meta-Directory. These products vary in their capabilities, but all of them support N-way join.

To join two entries in different data sources, you need to have a data element that is common to both sources. We refer to this data element as the *join attribute*. [Figure 23.4](#) illustrates the general concept of an attribute-based join.

Figure 23.4. An Attribute-Based Join



For example, if each of your source databases contains a field for U.S. Social Security number (SSN), you can use it to determine which entries correspond to the same people in both databases. Using a unique identifier such as an SSN or an employee ID number is much better than using a potentially nonunique identifier, such as a person's name. But SSNs are typically viewed as sensitive information; see the Privacy and Security Considerations section later in this chapter for a detailed discussion of SSN-related concerns.

A unique identifier may not be available, or you may not be able to use it (for example, employee IDs are also viewed as sensitive information in some companies). Sometimes you will not have anything better than a person's name to use for your join attribute. Such a limitation can reduce the efficiency of your synchronization procedure, create the need for manual synchronization and repair, and cause incorrect joining of information.

Overcoming these problems is one of the biggest challenges in providing a metadirectory service that has accurate data.

A join on first and last names may typically match no better than 50 percent of the names in your databases, and some of the matches may be false positives. The numbers get significantly worse as the number of entries in your databases increases (for example, the chance of having two people with the name "Babs Jensen" in a database of 100,000 people is significantly greater than the chance of having two people with the name "Babs Jensen" in a database of 100 people).

The result of inadequate matching is usually a lot of manual work. An administrator typically goes through the unmatched entries by hand, comparing other information to try to determine a match. A worse outcome could be that the wrong match is made automatically or by a careless administrator. In this situation, Person A's information can appear in Person B's entry. Depending on the type of information, how the directory service is used, and what people are involved, the consequences of this kind of error can range from annoying to serious.

When a good join attribute is not available, you can use multiple criteria to join data. Although each data value may not uniquely match the value in another data source, a combination of values might. For example, you might require matches on both name and city in order to reduce false positive matches involving names such as "John Smith." You can also use this technique to produce more matches than any one join attribute would. Choose a set of join attributes and assign a weight to each one. Then choose a threshold at or above which the data is assumed to match. For example, you might assign 2 points to surname, 1 point to first name, and 1 point to city, and use a threshold of 3 points. In this case if surname and city match, or if surname and first name match, you have a join. But a match of first name and city is not enough to conclude that you have a valid join.

If you have more than one data source that is authoritative for the same data element, you will probably need to use different join rules for each data source. For example, there may be one centralized personnel database that stores information about employees, but information about contractors may be managed independently by each department. The optimal join criteria must be selected for each data source.

Joining entries is an important capability that is needed to create an efficient and accurate directory coexistence process. When you evaluate directory coexistence software, be sure to consider its ability to provide this feature. Also investigate the software's advanced abilities, such as joining across multiple attributes and the extent to which you can tune the joining algorithm. In some environments it may make sense to sacrifice accuracy for a reduction in manual administration. Other environments may not be able to tolerate inaccurate joins.

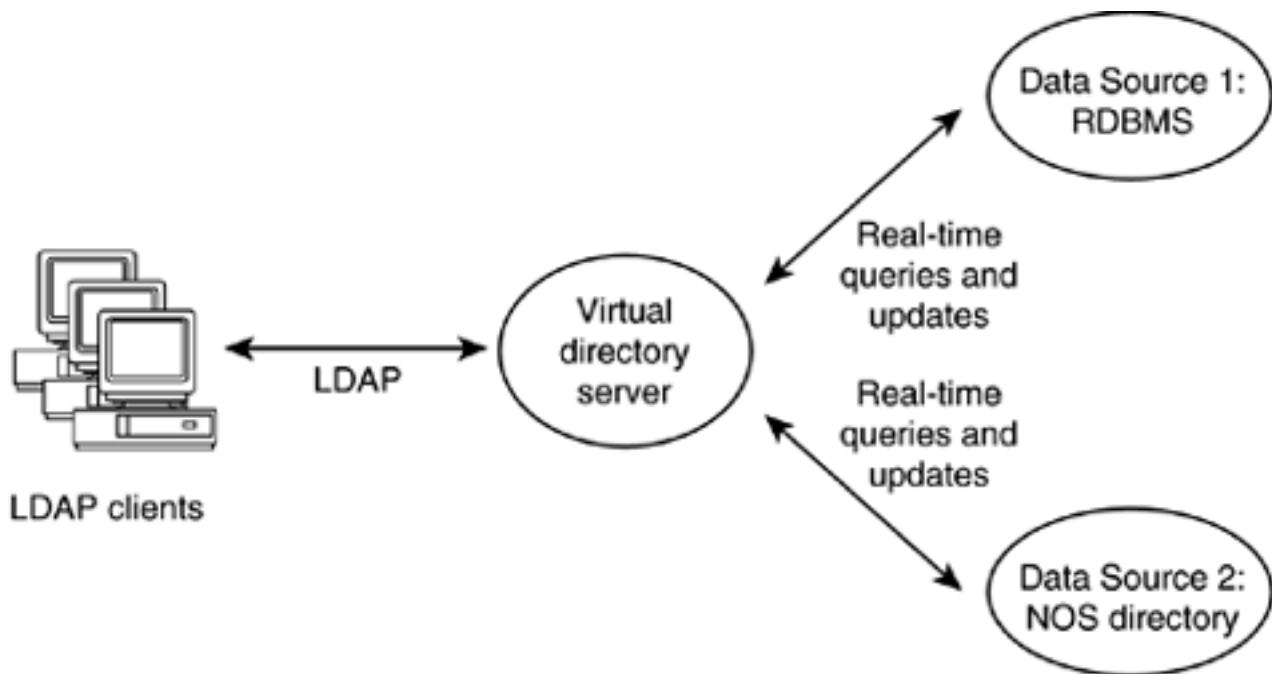
Virtual Directory

Another directory coexistence technique involves use of a virtual directory system. A relatively new addition to the directory world, the virtual directory uses a different approach from that of most synchronization techniques. Instead of copying data from one data source to another, a virtual directory provides a real-time directory view of selected data from multiple data sources. You can think of this as *virtual synchronization*: The virtual directory generally does not store its own persistent copy of the data (although it may cache the data to improve performance).

The implementation is simple in concept: A virtual directory looks to the outside world like a regular LDAP server, but it holds no persistent data of its own. When it receives a request, it reformats and reroutes that request to the necessary data sources. The answers received

are collated, reformatted, and sent back to the requestor. [Figure 23.5](#) shows the general architecture of a virtual directory system.

Figure 23.5. The General Architecture of a Virtual Directory System



A virtual directory system has several advantages:

- The question of who updates the multiple copies of data is neatly solved. The virtual directory simply routes update requests to the appropriate data sources; no copies of the data are made.
- The propagation delays inherent in a synchronization-based approach are avoided. No data is copied, and each query is mapped onto the source data store in real time.
- The virtual directory allows you to dispense with messy and costly data management procedures designed to synchronize data.

The virtual directory scheme also has several drawbacks:

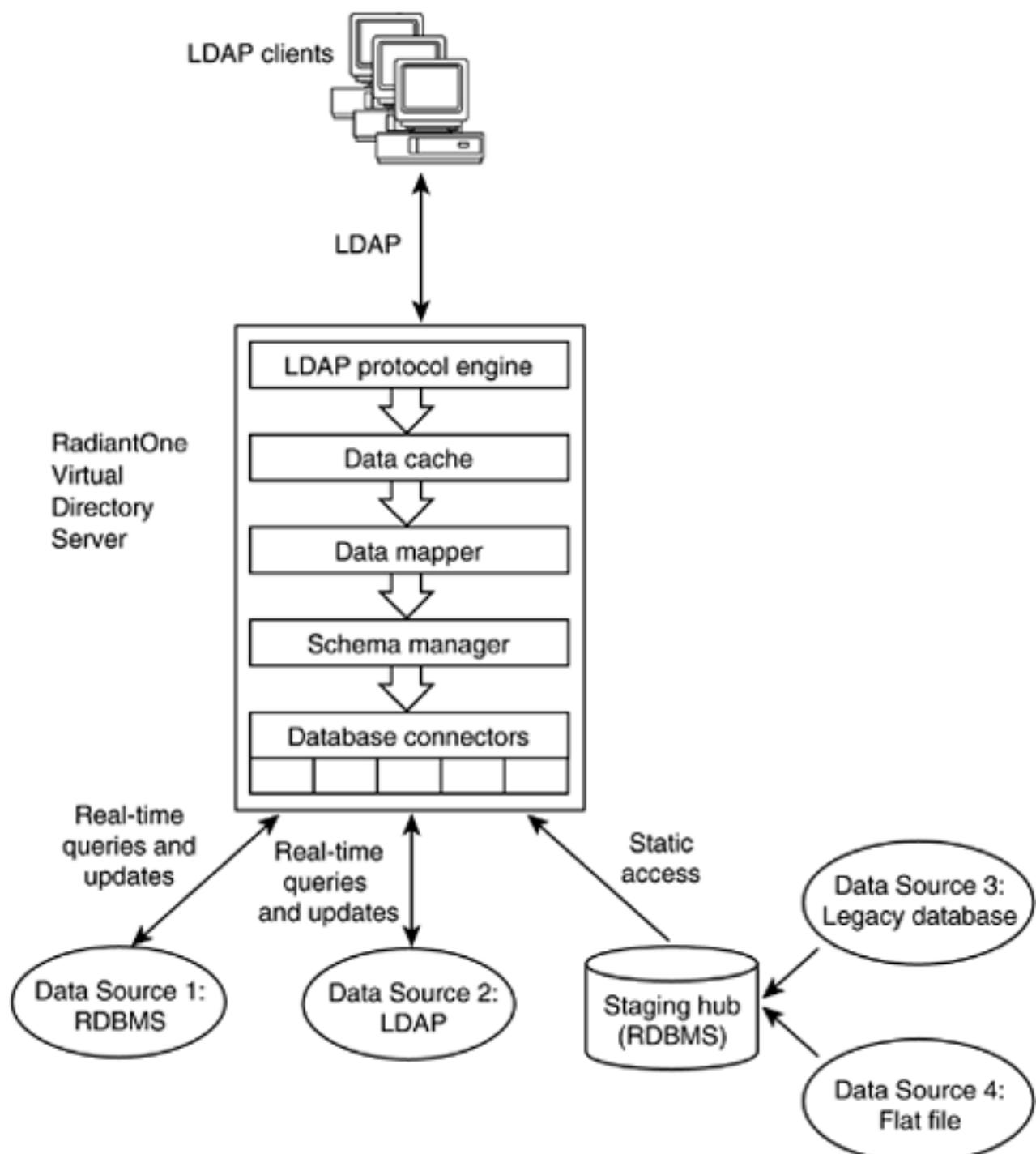
- In practice, a virtual directory system is complex to implement and deploy. The algorithms required to map queries among a set of data sources, collate results, deal with failures that may have occurred in one source database but not another, and so on can be difficult to determine and implement.
- Performance is likely to suffer compared to that of a centralized, synchronized directory approach. The source databases are not likely to be as fast as your general-purpose enterprise or extranet directory to begin with, and adding extra network round-trips, real-time query and data mapping, and result and error processing can reduce performance even further.
- The "real-time" response of the resulting system may be only as reliable as the least reliable data source that is included. For this reason, virtual directory deployments often combine traditional synchronization techniques in which data is copied (for slow or difficult-to-access data sources) with the real-time access techniques described in this section. Doing so makes access to the virtual directory more reliable; the trade-off is that the information that is synchronized will not be as up-to-date as the information that is accessed on the fly.

The pros and cons of a virtual directory add up to a few conclusions. Virtual directories are best suited for environments in which your goal is to provide access from a few well-known

applications to an existing database or set of databases. Knowing the applications, and therefore the kinds of requests they generate, allows you to significantly reduce the scope of your virtual directory deployment project. Another important consideration is performance: A virtual directory will be slower than a high-performance directory server that holds a copy of all data locally. You should be sure to evaluate virtual directory system performance early to ensure that it meets the performance needs of your applications.

Although pioneering directory deployers have created their own custom virtual directory by using facilities such as Netscape's Directory Server plug-in API, a variety of off-the-shelf software is now available. The best-known product is Radiant Logic's RadiantOne Virtual Directory Server. [Figure 23.6](#) shows an example of a deployment that uses this product.

Figure 23.6. A Deployment That Uses RadiantOne Virtual Directory Server



Data Translation

Regardless of the approach used to move data to and from your directory service, some form of data translation is usually necessary. Data translation involves any of the following:

- **Reformatting data elements to meet data source requirements.** It may be impossible to store data pulled from one data source in another source without reformatting the data. For example, in LDAP it is recommended that `postalAddress` attribute values consist of no more than six lines (separated by dollar signs) of no more than 30 characters each. If you need to move the postal address information from your directory service to a human resources database that supports only three lines of 40 characters each, some reformatting is required as part of your synchronization system.
- **Reformatting data elements to improve consistency.** Sometimes data is translated as a convenience to end users or applications. For example, even if the names that are synchronized into your directory service from an HR database consist entirely of uppercase letters, you may want to present the data in mixed case in your directory service (this kind of translation is difficult to do perfectly because of the wide variation in how names are constructed). Another example of this kind of reformatting is adding a prefix to telephone number values that are incomplete. Some data sources may store only a telephone extension (typically four or five digits), but you may want the values in your directory to follow the international standard for phone numbers, which includes the country code, area code, and local number.
- **Combining two or more data elements.** Sometimes your data sources may not have a data value you need, but they may contain two or more other data elements that can be used to construct the value. For example, separate street number, street name, city, state, and zip code fields can be combined to form one LDAP `postalAddress` value.
- **Removing duplicate data.** Some data sources contain duplicate records (for example, two entries for the same person) or duplicate values, and these should be removed as part of your directory coexistence process.
- **Removing obsolete or incomplete data.** Some data sources contain outdated or incomplete records that you will not want to include in your directory service. For example, a human resources database may contain information about former employees or contractors that no longer have an active contract with your company. Another example is a database of Web site visitors that contains many incomplete records (a very likely scenario if Web site visitors are not forced to enter all the information when they sign up).

Most off-the-shelf metadirectory and virtual directory software packages provide data translation features. However, the set of possible translation requirements is very large, so you may have to create custom code to handle some of the translations you need.

Privacy and Security Considerations

As discussed in [Chapter 12](#), Privacy and Security Design, different data elements have different security needs. Some data elements contain highly sensitive information and should be protected from unauthorized access and tampering. Other data elements do not contain sensitive information, and they need only to be protected from unauthorized modification. The coexistence process itself may present another way to compromise your data. In this section we explore issues related specifically to your directory coexistence strategy, including the privacy of the join attribute, the security of links in the coexistence process, and the security of data sources.

Tip

Design your directory coexistence procedures with security in mind. A poorly designed directory coexistence process often presents backdoor holes that can be used to compromise the security of your directory and the other data sources. Make sure the systems with which your directory exchanges data are as secure as the directory itself, and vice versa. Also be sure to secure the processes and links between the different systems.

The Join Attribute

As described earlier in this chapter, joining data among multiple directories is much easier when you have one or more join attributes that are common among all the directories. The more likely you are to have unique join attribute values, the more accurate the joining process will be and the less the administrative burden placed on directory administrators will be. This fact argues for choosing a unique join attribute if possible.

The most convenient attribute to use is often the Social Security number or something similar. However, SSNs are sensitive information that should be protected carefully. SSNs are issued by the U.S. government for tax identification purposes, but they are also used by many banks and other institutions as something akin to a password. If a person's SSN falls into the wrong hands, all kinds of trouble can follow, such as stolen identity, ruined credit, and unauthorized access to personal information. For those reasons, most users consider SSNs to be private information, and they will certainly be upset if care is not taken to protect their private information.

SSNs are also not immutable; they can change for various reasons. Finally, SSNs are not universal; not everyone has one, and you may not have access to SSNs for all the people represented in your directory. Non-U.S. employees, customers, and others may not have SSNs or want to give theirs to you. Therefore, the SSN may not be the best choice for a join attribute. But sometimes it is the only realistic choice you have.

When designing your directory coexistence processes, you must take care to protect sensitive join attributes such as a U.S. Social Security number. Consider who has access to the sensitive join attribute at all stages of the coexistence process. Do you really need to use SSNs, or is there another less sensitive attribute or set of attributes that would work just as well? Could you use a hashed form of the SSN, providing a measure of privacy protection? Does the coexistence process expose the join attribute to other less trusted directory or database administrators? These questions and others all need to be answered.

Data Transport

Another common security trouble spot in the directory coexistence process is the procedures used to transfer information between your directory service and the data sources. If these transfers are not protected with the same care that is used to protect data in the directory service itself, an attacker can potentially gain unauthorized access to data or even insert fraudulent data into the directory service.

There are a variety of techniques for protecting the transfer of data. If directory coexistence is maintained via LDAP, the normal security features of your directory, such as access control and Secure Sockets Layer (SSL) or Transport Layer Security (TLS) communication, can be applied. This is probably the best approach from a security standpoint; it minimizes the possible avenues of attack and allows you to focus your security design on the directory service itself. It also minimizes potential directory downtime that can be caused by non-LDAP data import.

If you choose an offline technique for transferring data to and from your data sources, consider other ways to protect the transfer. Several techniques are available, such as secure file transfer (for example, using the Unix `scp` or `sftp` commands), Secure Shell (the `ssh` command), encrypting the data itself and then transferring it via unprotected means, and so on. These techniques are described in more detail in [Chapter 12](#), Privacy and Security Design.

Data Source Security

A final consideration in the protection of your directory coexistence solution is the security of the data sources themselves. If the data sources are not secured, there may be little point in securing the same data in your directory service. The reverse argument also applies: You should avoid making your directory service a security weak point. An attacker may simply be able to access or change data in the data source, bypassing all the security you have labored to implement for your directory service.

For example, suppose your directory obtains telephone number information from a telephone operations database. If this information is considered sensitive, you might protect it in your directory service via authentication, access controls, and other techniques. A determined attacker bent on inserting invalid phone number data for another user cannot do it through the directory but may be able to do it through the telephone operations database. After the data is changed in the telephone operations database, it will find its way into your directory service via your synchronization procedures.

Be sure to consider more than just physical and technology-related security issues. Personnel and procedural security are also important. The most secure data source in the world can be compromised if an attacker is able to call up an administrator and talk him or her into making unauthorized changes. Masquerading as a legitimate user, trickery, flattery, and downright bribery are all potential weapons at an attacker's disposal. Make sure your procedures guard against such attacks and that your personnel are aware of the possibilities. Maintaining audit logs and similar information about update and synchronization activity will help in tracking down the root cause if your data does come under attack.

Determining Your Coexistence Requirements

Now that you have an idea why directory coexistence is important and what techniques are available, it is time to examine your specific directory coexistence requirements. Your requirements-gathering process should cover the following areas related to synchronization:

- **Data to be synchronized.** Determine which directory data needs to coexist with other data. Some data in your directory will not be needed outside of the directory. Some data in your other data sources will not be needed in your directory. But some data will need to be shared by the directory and other data sources. You need to determine which data elements fall into each category, and how the data needs to be reformatted or translated.
- **Type of synchronization.** Determine how the data should coexist. Will the data be maintained in (owned by) the directory and periodically fed to an external database? Will the data source own the data and feed it to the directory? Does the data need to be co-owned—that is, maintained in two data sources and synchronized in two directions via two-way synchronization? The answers to these questions determine the kind of synchronization technology you should use to maintain the coexistence.
- **Synchronization frequency.** Analyze each piece of coexisting data to determine how often it needs to be synchronized among all the participating data sources. Time frames may range from a one-time synchronization to as close to real-time synchronization as you can get (perhaps leading you to a virtual directory solution). Different data elements and data sources will have different requirements. You need to strike a balance among complexity, ease of implementation and management, and system performance (which synchronization frequency can affect substantially).
- **Synchronization security.** Depending on the sensitivity of the data being synchronized, security of the synchronization process may be an important consideration. Again, security needs vary among data elements and data sources. Seldom will you be able to choose a one-size-fits-all solution that is adequate for all your data elements and data sources. You need to strike a compromise among the complexity of maintaining multiple levels of security, the difficulty of development, and the performance and usability implications of making everything secure enough to protect the most sensitive data element.
- **Synchronization performance.** This final consideration is often overlooked. It is important to think of synchronization performance in the needs assessment phase. Otherwise you are likely to end up with unreasonable requirements, such as real-time synchronization for all data elements. Although often desirable, this kind of solution may have severe performance implications. A service that spends all of its available resources on directory coexistence and none serving directory clients is not a very useful service. Consider the trade-offs among synchronization frequency, security requirements, number of data sources that must be synchronized, and other factors that will affect the overall performance of your directory service and the other data sources.

To get started, you can use much of the work you began in [Chapter 7](#), Data Design. In that chapter we showed you how to create a table of data sources and data elements. This table should show where existing data in your organization is located and which data elements are of interest to your directory service. Your next task is to determine which of these data elements and sources need to coexist with the directory and which can serve as simple, one-time data sources.

Tip

You should always be suspicious of a decision to conduct a one-time data migration from a data source. This is a convenient way to jump-start your directory and load it with data, but often it can create problems down the road. Duplicate, uncoordinated data tends to increase your overall data maintenance costs and can lead to user frustration and confusion. On the other hand, if a data source is being phased out, migration may be the ideal approach. Be sure to think farther ahead than just populating your directory when you consider whether an ongoing coexistence solution is needed.

For those data elements that must coexist, you need to determine the following:

- **Data owner.** Where is the data updated? What is the authoritative data source for this data element? Most data is updated in one place, but you may have data that you would like to be updated in multiple places. In such cases the same data element has multiple data sources. As we discussed earlier, in the section titled Two-Way Synchronization, although having multiple data sources is desirable in rare circumstances, it should be avoided as much as possible. Allowing more than one place where a data element can change brings with it complications that make your job of implementing and administering a coexistence solution much harder.
- **Data flow.** What data source or directory service is the destination for the data? Does it flow in one direction or both directions?
- **Frequency.** How often must the data element be synchronized? Is real-time access required, or will a daily or weekly one-way synchronization process suffice?
- **Special considerations.** Note any special considerations that are related to areas such as security, data translation, and so on.

Create a directory coexistence table like [Table 23.1](#) to aid your design process and to provide a summary of all your design work. This example assumes there are three sources for directory data in addition to the directory service itself: the corporate human resources database, the telephone operations database, and end users. Data elements flow in one direction only. Some information flows from the directory to these data sources, some flows from the data sources to the directory, and some information is simply maintained in the directory service and does not need to integrate with any other data sources.

[Figure 23.7](#) shows how this information can be represented in graphical form. Creating a data flow graph can help you better visualize how the different systems need to interact and how data should flow across all of the integrated systems.

Figure 23.7. A Sample Data Flow Graph

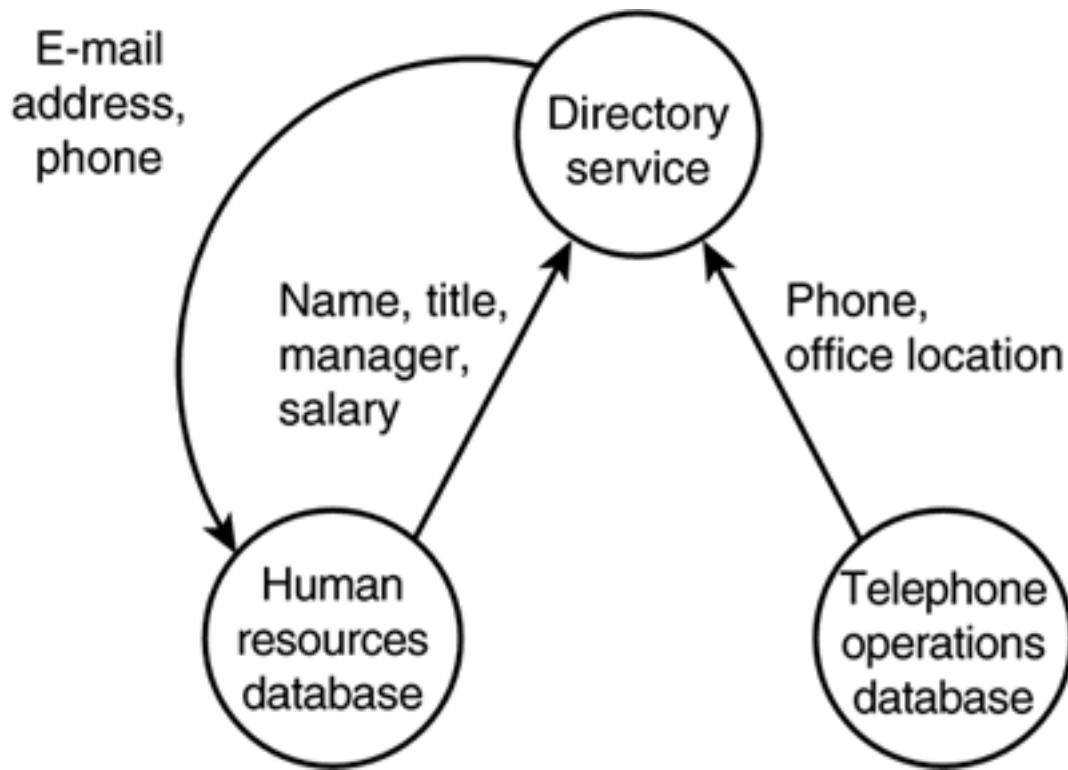


Table 23.1. A Sample Directory Coexistence Table

Data Element	Data Owner	Data Flows to	Frequency	Special Considerations
Name	Human resources	Directory service	Weekly	None
Title	Human resources	Directory service	Weekly	None
Manager	Human resources	Directory service	Weekly	None
Salary	Human resources	Directory service	Weekly	Sensitive information
Phone	Telephone operations	Directory service, human resources	Daily	Must reformat data values
Office location	Telephone operations	Directory service	Daily	None

E-mail address	Directory service	Human resources	Weekly	New field to be added to the HR database
Description	Directory user	Directory service	User-controlled	None
Other directory attributes	Directory service	None	Not applicable	All other attributes maintained in the directory service are not shared

Team LiB

◀ PREVIOUS

NEXT ▶

Directory Coexistence Implementation Considerations

The design and implementation of any directory coexistence solution inevitably involves sweat, trial and error, and some frustration. As always, we recommend an incremental approach: Tackle your most critical coexistence requirements first, while keeping the other needs in mind. Succeeding usually involves many trade-offs, and it may be difficult to meet all of your requirements on day one. This section describes a variety of implementation considerations, with the goal of helping you make the right trade-offs.

Implementation Options

Depending on your requirements, you may have a lot of flexibility in how you achieve coexistence between each data source and your directory service—or you may not. The techniques presented earlier in this chapter all have their pros and cons. Usually there is a trade-off between complexity of implementation and the functionality achieved.

For example, a one-time migration of data is easy, but it does not provide for future updates to or from the original data source. On the other hand, a two-way synchronization approach with N-way join that is based on commercial metadirectory software can provide a wide range of functionality, including extensive data translation facilities, near real-time synchronization of data, and multiple places where data may be updated. However, there is no free lunch: Such systems are often difficult to design, deploy, and maintain.

As has been our theme throughout this book, we recommend that you choose the simplest technique that adequately meets your needs. Remember, different techniques will be appropriate for different data sources and elements.

Performance Implications

One important area of directory coexistence that is often overlooked is the performance of the systems used to achieve that coexistence and the impact of coexistence on your directory service and other data sources. Some coexistence techniques are by nature more performance intensive than others; for example, two-way synchronization may impose twice as much system load as one-way synchronization. But you can dramatically improve performance by reducing the frequency of synchronization or by using incremental updates instead of completely refreshing the data.

Before you decide which coexistence techniques and tools to use, think about performance and engage in some thought experiments. After you have made a preliminary decision about which techniques to use, evaluate the actual performance by running a directory coexistence pilot. See [Chapter 14](#), Piloting Your Directory Service, for more information on conducting meaningful pilot deployments.

Directory Coexistence Tools

Once you have chosen your preferred coexistence techniques, you need to select a set of tools to implement them. Here are some of the many choices:

- **Application-specific synchronization tools.** Some application packages, such as e-mail server software, include synchronization and data translation tools. At the very least, file-based import and export tools are usually provided.

- **Tools that come with your directory server software.** Most directory server implementations have the ability to import and export data in a variety of formats. Some also include synchronization software.
- **Metadirectory and virtual directory software.** Sophisticated (and expensive) commercial packages are available that include synchronization, join, data translation, and virtual directory capabilities. Some open-source software is also available.
- **Custom tools that you develop yourself.** To paraphrase an old saying, "If you want a solution that does just what you need and no more, develop it yourself." Even if you use off-the-shelf software to meet most of your requirements, directory coexistence is an area where some custom software development is usually needed. An example of a simple one-way synchronization tool that is implemented in Perl is presented a little later in the chapter.

The most important issues to consider when selecting coexistence tools are whether a given tool will meet your requirements and how well that tool will work in your environment. Cost, ease of deployment, vendor support, and availability of consulting services are also important factors. Refer to [Chapter 13](#), Evaluating Directory Products, for general information on performing an appropriate evaluation.

Tuning and Troubleshooting

Perhaps the most painful step in achieving directory coexistence is tuning the system until it works well. Why is this painful? As you have probably figured out by now, directory coexistence is an ugly problem. Therefore, challenges unique to your environment will inevitably arise. Most tuning and troubleshooting should be done during a pilot deployment, and the ability to get advice from the following people during the design and pilot phases is extremely valuable:

- The designers and system administrators of your directory service
- The designers and system administrators of the data sources included in your coexistence effort
- The vendor or author of the directory coexistence tools you're using
- The data security team within your organization, if you have one
- Consultants and other experts you bring in to assist with the directory coexistence implementation

The last group is worth emphasizing. Many organizations shy away from hiring outside consultants, but directory coexistence is one area where experience counts for a lot. Hiring the right consultant may save a lot of time and frustration. Whatever you do, don't rush a coexistence solution that is not ready into production. See [Chapter 16](#), Putting Your Directory Service into Production, for general information on the optimal process.

Monitoring and Caring for Your Coexistence Solution

Finally, don't expect your directory coexistence solution to take care of itself. As with all other aspects of your directory deployment, it is important to implement automated monitoring of the system. [Chapter 19](#), Monitoring, includes information on how to monitor your directory service; similar techniques should be used to monitor the components of your directory coexistence solution.

The type of monitoring required varies depending on the coexistence techniques you use and the frequency of synchronization. For example, for daily or less frequent

synchronization, sending out a nightly e-mail message that indicates whether synchronization was successful may be sufficient. On the other hand, continuous monitoring that includes automated notification of problems is appropriate for a virtual directory system.

You also need to create troubleshooting and problem escalation procedures for the various components of your directory coexistence solution. For busy systems that use incremental synchronization, it is especially important to take action promptly. If the synchronization process gets stuck, the backlog of unapplied updates may quickly grow to an unmanageable amount.

Example: The Idapsync Tool: One-Way Synchronization with Join

The Idapsync tool presented in this example may be used to implement periodic one-way synchronization into an LDAP server from any system whose data can be expressed as a delimited text file. Many data sources provide tools that make it easy to generate this kind of data extract. Some systems provide the ability to extract only those changes that have occurred since the previous extract, in which case the Idapsync tool presented here runs more efficiently.

Our tool is a homegrown tool that is written in the Perl 5 language. Perl allows for rapid prototyping, provides good portability of the code, performs acceptably, and may be easily modified by the thousands of system administrators who have learned to program in it.

How It Works

The Idapsync tool connects to an LDAP directory server that holds a writable copy of people data. It authenticates using a distinguished name (DN) and password and then reads a series of comma-delimited lines from standard input.

Within the Idapsync input stream, all lines that begin with a pound sign (#) are treated as comments and ignored. The first noncomment line must look like this:

join-attribute-name, update-attribute-name1, update-attribute-name2...

One or more update attribute names may be listed. For example, here's the input line to specify `telephoneNumber` as the join attribute and to support updates to the full name (`cn`) and surname (`sn`) attributes:

`telephoneNumber, cn, sn`

The remaining input lines must look like this:

join-attribute-value, update-attribute-value1, update-attribute-value2...

The number of comma-separated values on each of these lines should match the number of attribute names listed on the first noncomment line, although the Idapsync tool is smart enough to ignore extra values and treat missing ones as absent (no update needed).

Here's an example of an input line that specifies a join value of `+1 650 555-1234` and updates the `cn` attribute with `Babs Jensen` and the `sn` attribute with `Jensen`:

`+1 650 555-1234, Babs Jensen, Jensen`

And here's an example of an input line that specifies a join value of `+1 650 555-4567`, no `cn` update, and an `sn` update of `Jones`:

```
+1 650 555-4567, ,Jones
```

The synchronization process is driven entirely from the input data. For each line that includes values, the Idapsync tool issues an LDAP search to locate an entry. The LDAP search filter is constructed on the basis of the join attribute value. For example, if the join attribute is user ID (`uid`) and the join value provided is `mcs`, then a search filter like this is used: `(&(objectClass=person)(uid=mcs))`. If one entry is found, the Idapsync tool checks each new value provided to see if that value is already present in the entry. If any values are missing, updates are required and the Idapsync tool uses an LDAP modify operation to replace the values that need to be updated. As it processes the input data, the Idapsync tool logs error and informational messages to standard output.

Usage Examples

Suppose that you want the telephone numbers for people in your directory service to come from a human resources database and that both the HR database and the directory store a user ID value (the `uid` attribute in the directory). [Listing 23.1](#) shows a sample Idapsync input file whose purpose is to update many people's `telephoneNumber` attribute values in your directory. Telephone number changes were extracted from the HR database to create the input file. Some lines have been omitted and replaced with "..." to reduce the length of the listing; in all there are 150 `telephoneNumber` changes.

Listing 23.1 An Idapsync Input File That Contains `telephoneNumber` Updates

```
# join attribute name,attribute name
uid,telephoneNumber
#
# uid value, telephoneNumber value
scarter,+1 650 555 4798
tmorris,+1 650 555 9187
kvaughan,+1 650 555 5625
...
elott,+1 650 555 0932
cnewport,+1 650 555 0066
jvedder,+1 650 555 4668
```

If the data in [Listing 23.1](#) were stored in a file called `changes-from-hr`, the Idapsync tool

could be invoked like this:

```
ldapsync.pl < changes-from-hr
```

[Listing 23.2](#) shows the output of this command when run against a directory server that contains the contents of the `Example.ldif` file that Netscape bundles with its Directory Server product.

Listing 23.2 Output from the First Idapsync Sample Run

```
INFO.: Mon Jul 15 11:40:38 2002 - ldapsync started
INFO.: Found one entry with uid scarter. Checking for updates...
INFO.:      - replace telephoneNumber with +1 650 555 4798
INFO.: Modifying entry with uid scarter
INFO.: Modify done.

INFO.: Found one entry with uid tmorris. Checking for updates...
INFO.:      - replace telephoneNumber with +1 650 555 9187
INFO.: Modifying entry with uid tmorris
INFO.: Modify done.

INFO.: Found one entry with uid kvaughan. Checking for updates...
INFO.:      - replace telephoneNumber with +1 650 555 5625
INFO.: Modifying entry with uid kvaughan
INFO.: Modify done.

...
INFO.: Found one entry with uid elott. Checking for updates...
INFO.:      - replace telephoneNumber with +1 650 555 0932
INFO.: Modifying entry with uid elott
INFO.: Modify done.

INFO.: Found one entry with uid cnewport. Checking for updates...
INFO.:      - replace telephoneNumber with +1 650 555 0066
INFO.: Modifying entry with uid cnewport
INFO.: Modify done.

INFO.: Found one entry with uid jvedder. Checking for updates...
```

```
INFO.:      - replace telephoneNumber with +1 650 555 4668
INFO.: Modifying entry with uid jvedder
INFO.: Modify done.

INFO.: -----
INFO.: Summary:
INFO.: 150 change records were processed.
INFO.: 150 entries were updated.
INFO.: Mon Jul 15 11:40:53 2002 - ldapsync finished.
```

Again, some lines have been omitted and replaced with "..." to reduce the length of the listing. Notice the summary at the end: "150 entries were updated." If the same `ldapsync.pl` command is executed a second time, no changes are made to the LDAP directory (because all of the values provided in the input file are now present in the directory server). [Listing 23.3](#) shows the resulting output.

Listing 23.3 Output from the Second Idapsync Sample Run

```
INFO.: Mon Jul 15 11:40:55 2002 - ldapsync started
INFO.: Found one entry with uid scarter. Checking for updates...
INFO.: Found one entry with uid tmorris. Checking for updates...
INFO.: Found one entry with uid kvaughan. Checking for updates...
...
INFO.: Found one entry with uid elott. Checking for updates...
INFO.: Found one entry with uid cnewport. Checking for updates...
INFO.: Found one entry with uid jvedder. Checking for updates...
INFO.: -----
INFO.: Summary:
INFO.: 150 change records were processed.
INFO.: No entries were updated.
INFO.: Mon Jul 15 11:40:58 2002 - ldapsync finished.
```

[Listing 23.4](#) shows another sample input file. This file also uses `uid` as the join attribute, but it specifies an assortment of changes to the `cn`, `sn`, and `telephoneNumber` directory attributes.

Listing 23.4 An Idapsync Input File That Contains Updates to Three Different Attribute Types

```
# Sample ldapsync input file

#
# the first noncomment line lists the attribute names (key first):
uid,cn,sn,telephoneNumber

#
# updates:

bjensen,B Jensen,Jones,+1 408 555 4321

scarter,S Carter

kwinters,,,+1 408 555 1234
```

For the `bjensen` entry, `cn`, `sn`, and `telephoneNumber` are all to be updated. For the `scarter` entry, only `cn` is to be updated. For the `kwinters` entry, only `telephoneNumber` is to be updated. [Listing 23.5](#) shows the output produced by the `Idapsync` tool when it is driven from the data in [Listing 23.4](#).

Listing 23.5 Output from the Third Idapsync Sample Run

```
INFO.: Mon Jul 15 11:49:31 2002 - ldapsync started

INFO.: Found one entry with uid bjensen. Checking for updates...

INFO.:      - replace cn with B Jensen

INFO.:      - replace sn with Jones

INFO.:      - replace telephoneNumber with +1 408 555 4321

INFO.: Modifying entry with uid bjensen

INFO.: Modify done.

INFO.: Found one entry with uid scarter. Checking for updates...

INFO.:      - replace cn with S Carter

INFO.: Modifying entry with uid scarter

INFO.: Modify done.

INFO.: Found one entry with uid kwinters. Checking for updates...

INFO.:      - replace telephoneNumber with +1 408 555 1234
```

```
INFO.: Modifying entry with uid kwinters
INFO.: Modify done.

INFO.: -----
INFO.: Summary:
INFO.:    3 change records were processed.
INFO.:    3 entries were updated.
INFO.: Mon Jul 15 11:49:32 2002 - ldapsync finished.
```

As these examples show, Idapsync is a simple but versatile tool.

The Source Code

The Idapsync tool uses the PerLDAP object-oriented LDAP access module available from the Mozilla Web site at <http://mozilla.org/directory/perldap>. The Perl source code for Idapsync is all in one file, named, logically enough, `ldapsync.pl`. [Listing 23.6](#) shows the first portion of the Idapsync code.

Listing 23.6 The Idapsync Code (Part 1 of 5)

```
1.#!/usr/local/bin/perl
2.#
3. # ldapsync -- Perl 5 script that synchronizes a
4. #   comma-separated text file of attribute values.
5.#
6. # From the 2nd Edition of the book:
7. #   "Understanding and Deploying LDAP Directory Services"
8. #   by Timothy A. Howes, Mark C. Smith, and Gordon S. Good.
9.#
10.# usage: ldapsync.pl < file
11.#
12.# Where the contents of file are of the form:
13.#   joinAttrName,attrname1,attrname2...
14.#   joinAttrValue,attrvalue1,attrvalue2...
15.#   joinAttrValue,attrvalue1,attrvalue2...
```

```
16. # ...
17. #
18. # For example:
19. # uid,cn,telephoneNumber
20. # bjensen,Barbara Jensen,+1 650 555-1212
21. # cms,Christina Smith
22. # jjones,John Jones,+1 734 555-1212
23. # ajackson,,+1 810 555-1212
24. #
25. # Requires: PerLDAP
26. #
27.
28. use Mozilla::LDAP::Conn;
29.
30. # LDAP server information:
31. $ldapSearchBase = "dc=example,dc=com";
32. $ldapHost      = "ldap.example.com";
33. $ldapPort      = "389";
34. $ldapBindDN    = "cn=LDAP Sync Tool,ou=Special Users,"
35. .           $ldapSearchBase;
36. $ldapBindPW   = "Tb58-#Wxza";
37.
38. # Start of main:
39.
40. logInfo( scalar localtime ) . " - ldapsync started" );
41.
42. # Open an authenticated connection to the LDAP server.
43. $ldap = new Mozilla::LDAP::Conn( $ldapHost, $ldapPort,
44.           $ldapBindDN, $ldapBindPW );
45. if ( ! $ldap ) {
```

```

46.     logError( "Unable to connect to server at "
47.                 . "ldap://$ldapHost:$ldapPort" );
48.     exit 1;
49. }
50.

```

The Perl interpreter identified on line 1 must be one that has the PerLDAP module installed (the copy of Perl that Netscape bundles with its Directory Server 6 product was used in testing this script). A set of variables that holds the LDAP server information is set by the code on lines 30–36. A subroutine is called on line 40 to log a startup message (the code for `logInfo()` is shown later in this section, in [Listing 23.10](#)). An LDAP connection is opened by the code on lines 42 to 49, and the synchronization tool authenticates itself as a special directory entry that presumably has been given permission to update the desired attributes in the LDAP server.

[Listing 23.7](#) shows the code that reads and parses the `ldapsync` input. The `while` loop that begins on line 54 encloses the main body of the `ldapsync.pl` program. The loop is executed as long as there is more input to be read.

Listing 23.7 The `ldapsync` Code (Part 2 of 5): Reading and Parsing Input

```

51. # For each line of input, search for the directory entry
52. # corresponding to the first field, and see if its value
53. # for the second field needs to be updated
54. while ( <STDIN> ) {
55.     # Read one line.
56.     $line = $_;
57.
58.     # Skip lines that begin with '#' (comments)
59.     if ( $line =~ /^#/ ) {
60.         next;
61.     }
62.
63.     # Discard newline and return characters; break at commas
64.     chop $line;
65.     if ( $line =~ /\r$/ ) {

```

```

66.     chop $line;
67. }
68. @args = split( //, $line );
69.
70. # If this is the first line, read attribute names
71. if ( ! $joinAttrName ) {
72.     $joinAttrName = @args[0];
73.     @attrNameList = @args;
74.     next;
75. }
76.
77. ++$changeCount;
78.
79. # Parse join attribute and attribute values to update
80. $joinAttrValue = @args[0];
81. @attrValueList = @args;
82.

```

The code on lines 55 to 61 reads one input line and takes care of skipping comment lines. The code on lines 63 to 68 parses the input line into an array of strings (using commas as separators). The code on lines 70 to 75 processes the first noncomment input line; it creates a `joinAttrName` string variable and an `attrNameList` array variable to store the relevant attribute names. Line 77 increments a counter that is used to generate the summary, and the code on lines 79–81 sets the `joinAttrValue` and `attrValueList` variables that hold the join value and any values to update that are present on the input line.

Next some LDAP-related work is done. [Listing 23.8](#) shows the code that searches for an entry to update. It uses the PerlLDAP `search()` method.

Listing 23.8 The Idapsync Code (Part 3 of 5): Searching for an Entry to Update

```

83. # Search for entry with uid equal to the join attribute
84. $filter = "(&(objectClass=person)"
85.         . "($joinAttrName=$joinAttrValue))";

```

```

86.     $entry = $ldap->search( $ldapSearchBase, "subtree",
87.                               $filter, 0, @attrNameList );
88.
89. # Found a match - update if necessary
90.     $matchLogString = "with $joinAttrName $joinAttrValue";
91.     $entryCount = 0;
92.     for ( $tmpEntry = $entry; $tmpEntry;
93.           $tmpEntry = $ldap->nextEntry ) {
94.         ++$entryCount;
95.     }
96.     if ( $entryCount == 0 ) {
97.         logError( "Found no entry $matchLogString." );
98.         ++$notFoundCount;
99.     } elsif ( $entryCount > 1 ) {
100.        logError( "$entryCount entries found $matchLogString." );
101.        ++$multipleMatchCount;
102.    } else {
103.        logInfo( "Found one entry $matchLogString."
104.                  . " Checking for updates..." );

```

The code on lines 83 to 87 constructs a search filter and issues an LDAP search operation against the directory server. The remainder of the code checks that exactly one entry was found and logs an error message if not (the code for the `logError()` subroutine is shown later in this section, in [Listing 23.10](#)).

[Listing 23.9](#) shows the code that determines whether any values in the entry that was found need to be updated and issues an LDAP modify command if updates are needed.

Listing 23.9 The Idapsync Code (Part 4 of 5): Modifying an Entry

```

105.     $entryNeedsUpdating = 0;
106.     for ( $i = 1; $i < @attrNameList.size; ++$i ) {
107.       if ( $i > @attrValueList.size ) {
108. # No more new values: break out of for loop.

```

```
109.         done;
110.     }
111.     if ( @attrValueList[$i] eq "" ) {
112. # New value is empty: try the next one
113.         next;
114.     }
115.
116. # Replace attribute's values if new value does not match old
117.     if ( ! $entry->hasValue( @attrNameList[$i],
118.                               @attrValueList[$i], 0 ) ) {
119.         $entry->setValues( @attrNameList[$i],
120.                               @attrValueList[$i] );
121.         logInfo( "      - replace @attrNameList[$i]"
122.                 . " with @attrValueList[$i]" );
123.         $entryNeedsUpdating = 1;
124.     }
125. }
126.
127. # Update entry over LDAP if needed
128.     if ( $entryNeedsUpdating ) {
129.         logInfo( "Modifying entry $matchLogString" );
130.         $ldap->update( $entry );
131.         if ( $ldap->getErrorCode() ) {
132.             logError( "Modify failed for entry"
133.                     . " $matchLogString: "
134.                     . $ldap->getErrorMessage() );
135.             ++$updateFailureCount;
136.         } else {
137.             logInfo( "Modify done." );
138.             ++$updateSuccessCount;
```

```

139.         }
140.     }
141.   }
142. }
143.

```

The PerLDAP `hasValue()` method is used by the code on lines 117 and 118 to determine whether a value read from the input file is already present in the entry; this check avoids unnecessary updates. If the value is not present, the code on lines 119 and 120 uses the PerLDAP `setValues()` method to replace the existing values in the entry with the new ones. An `entryNeedsUpdating` flag is set on line 123 so that the code on lines 127 to 141 can determine whether an LDAP modify operation needs to be issued. If so, the code on line 130 uses the PerLDAP `update()` method to perform the LDAP modify.

[Listing 23.10](#) shows the remainder of the Idapsync source code. The code on lines 144 to 171 closes the LDAP connection and prints a series of "INFO.:" lines that summarize the Idapsync run. The code segments for the `logError()` and `logInfo()` subroutines start on lines 179 and 189, respectively. These subroutines simply print error and informative lines in a consistent way.

Listing 23.10 The Idapsync Code (Part 5 of 5)

```

144. # Close LDAP connection and clean up
145. $ldap->close;
146.
147. # Print an activity summary and determine exitCode
148. logInfo( "-----" );
149. logInfo( "Summary:" );
150. $exitCode = 0;
151. if ( $changeCount > 0 ) {
152.   logInfo( " $changeCount change records were processed." );
153.   if ( $updateSuccessCount > 0 ) {
154.     logInfo( " $updateSuccessCount entries were updated." );
155.   } else {
156.     logInfo( " No entries were updated." );
157.   }
158.   if ( $updateFailureCount > 0 ) {

```

```
159.     logInfo( " $updateFailureCount failed update(s)." );
160.     $exitCode = 1;
161. }
162. if ( $notFoundCount > 0 ) {
163.     logInfo( " $notFoundCount entries were not found." );
164.     $exitCode = 1;
165. }
166. if ( $multipleMatchCount > 0 ) {
167.     logInfo( " $multipleMatchCount $joinAttrName values"
168.             . " matched more than one entry." );
169.     $exitCode = 1;
170. }
171. }
172.
173. # That's all folks!
174. logInfo( scalar localtime) . " - ldapsync finished." );
175. exit $exitCode;
176. # End of main.
177.
178.
179. # Start of logError():
180. sub
181. logError {
182.     local( $msg ) = @_;
183.
184.     print "ERROR: $msg\n";
185. }
186. # End of logError().
187.
188.
```

```
189. # Start of logInfo():

190. sub

191. logInfo {

192.     local( $msg ) = @_;
193.

194.     print "INFO.: $msg\n";
195. }

196. # End of logInfo().
```

Ideas for Improvement

Many things could be done to improve the Idapsync tool; if you decide to use it, some customization will likely be needed to meet your specific synchronization requirements. Here are a few general ideas for improvement:

- Add an option to create entries that are missing from the LDAP directory service.
- Add data translation features. For example, user IDs could be changed into e-mail addresses by the addition of "@domain." Telephone extensions could be turned into complete international standard phone numbers by the addition of "+1" and the other missing digits.
- Improve efficiency in the face of errors by adding code to create a "reject" file that contains all updates that could not be made (for example, some updates might fail because of an LDAP server outage or a permissions problem). Once the problem that caused the updates to fail has been corrected, the Idapsync tool could be run again with the reject file as input.
- Improve security by using SSL or TLS to protect the TCP connection to the LDAP server and by using certificate-based authentication instead of password-based authentication.

Directory Coexistence Checklist

To recap, these tasks should be performed to accomplish directory coexistence:

- Identify external data sources and elements that need to coexist.
- Identify the desired type of synchronization.
- Determine the desired frequency of synchronization.
- Determine the sensitivity of synchronized data.
- Evaluate coexistence implementation options.
- Consider performance implications on the directory service.
- Design and implement a coexistence solution.
- Monitor coexistence, and take action when you find a problem.

Further Reading

CP Meta-Directory Server. Critical Path. Information is available on the World Wide Web at <http://www.cp.net/solutions/platform-ims-metadirectory.html>.

Enterprise Application Integration. D. Linthicum, Addison-Wesley, 1999.

Enterprise-wide Software Solutions: Integration Strategies and Practices. S. Lozinsky and P. Wahl, Addison-Wesley, 1998.

Examining Data Quality. G. K. Tayi and D. P. Ballou, *Communications of the ACM*, February 1998, pp. 54–57.

Implementing SAP R/3: How to Introduce a Large System into a Large Organization (2nd edition). N. H. Bancroft, H. Seip, and A. Sprengel, Prentice Hall, 1997.

Metadirectory Practices for Enterprise Directories in Higher Education. R. Jones, T. Barton, K. Hazelton, B. Bellina, E. Shepard, and A. West, NSF Middleware Initiative, 2002. Available on the World Wide Web at <http://middleware.internet2.edu/dir/metadirectories>.

Microsoft Metadirectory Services. Information is available on the World Wide Web at <http://www.microsoft.com/windows2000/technologies/directory/mms>.

Netscape Directory Server Deployment Guide. Available on the World Wide Web at <http://enterprise.netscape.com/docs/directory>.

RadiantONE Virtual Directory Server. Radiant Logic. Information is available on the World Wide Web at <http://www.radiantlogic.com>.

Sun ONE Meta-Directory. Information is available on the World Wide Web at http://www.sun.com/software/products/meta_directory.

Looking Ahead

Congratulations! Now that you have completed [Parts I](#) through [V](#) of this book, you should know how to design, deploy, maintain, and leverage a first-class directory service. In the final section of the book—[Part VI](#), Case Studies—we will turn our attention to several directory deployment case studies. The case studies are presented to give you more practical knowledge of directory services, as well as to show how the advice contained in [Parts I](#) through [V](#) applies to the big picture of directory design.

Part VI: Case Studies

[24. Case Study: Netscape Communications Corporation](#)

[25. Case Study: A Large Multinational Enterprise](#)

[26. Case Study: An Enterprise with an Extranet](#)

Chapter 24. Case Study: Netscape Communications Corporation

- Overview of the Organization
- Directory Drivers
- Directory Service Design
- Directory Service Deployment
- Directory Service Maintenance
- Leveraging the Directory Service
- Summary and Lessons Learned
- Further Reading
- Looking Ahead

[Part VI](#), Case Studies, shows how organizations actually plan for and deploy directory services. The case study presented in this chapter represents a real-life directory deployment with which the authors had direct experience. The other two case studies (presented in [Chapter 25](#), Case Study: A Large Multinational Enterprise, and [Chapter 26](#), Case Study: An Enterprise with an Extranet) are fictional, and they represent the authors' concept of how a directory service could be deployed to meet the needs of such organizations.

By presenting these three case studies, we hope to show how a directory can be applied within several different organizational settings. Some aspects of each organization depicted in these case studies will probably match those of your own organization, although no one organization will match in all respects. Use these case studies as starting points for developing a directory strategy for your own needs.

Overview of the Organization

Netscape Communications Corporation is a leading supplier of open client and server software that links people and information over the Internet and intranets. In addition, Netscape's Web portal, located at <http://home.netscape.com>, is a leading rendezvous point for people wanting to navigate the Net, access products and services, and communicate with others online.

Netscape was founded in 1994 by Jim Clark and Marc Andreessen. Starting in 1996, the authors were employed by Netscape as software engineers and architects. That same year, a project was started inside Netscape to deploy an enterprise directory service to meet the needs of the growing company and to provide a test bed for Netscape's directory services software products. In March of 1999, Netscape completed a merger with America Online, Inc. In January 2001, the combined company merged with Time Warner to form AOL Time Warner, the world's largest Internet and media company. In this chapter we focus mainly on the initial directory deployment that was created by Netscape prior to its merger with America Online.

Most of Netscape's employees and contractors are based in Mountain View, California. The Mountain View main campus, which is also home to other AOL employees, consists of numerous individual buildings connected via high-speed fiber-optic network connections. The Mountain View campus is connected to the Internet via a set of OC3 connections, which are also used to support its World Wide Web presence. The campus is also connected to the main AOL campus, which is located in Dulles, Virginia, via AOL's backbone network.

In addition to the main campus, smaller engineering, sales, and support offices are located throughout North America, Europe, and Asia. Connections to each of the satellite offices vary depending on the number of employees and the location of the office, and they range from dial-up lines to higher-speed leased lines.

At the time the directory service described in this chapter was deployed, Netscape was a standalone company with a central Information Services (IS) department and a central Human Resources (HR) department. The Netscape IS group created a team that was responsible for developing and deploying the directory and directory-enabled applications, and the HR group provided the data about employees to IS for inclusion in the directory. HR uses software from PeopleSoft to manage information about its employees, contractors, and contingent workers. This information is used for business processes such as payroll and benefits enrollment.

Unlike some organizations (see [Chapter 26](#) for a counterexample), most of Netscape's IS services are provided centrally rather than distributed throughout the various units within the company. Although desktop and network support personnel are deployed throughout Netscape, functions such as the employee Help Desk and development of infrastructure services such as the directory service are handled by a central team. This means that a high degree of control and centralization exists, so decision-making and deployment of new computing services are fast. Your organization may be much more "balkanized," especially if it is very large or has grown through mergers and acquisitions. (In fact, Netscape's original Information Services group was more centralized than the AOL Internal Computing division it eventually merged with, and across the entire set of AOL Time Warner companies there are several IS organizations.)

In addition to being used inside Netscape, directory technology (specifically Netscape Directory Server) is a core component of Netscape's Web portal site, Netscape.com. Although Netscape.com would certainly make an interesting case study itself, we don't discuss it here. Netscape Directory Server is also used by many other internal and externally

visible services within AOL Time Warner; for example, it is used within the Road Runner division to help provide their cable Internet service. But this case study focuses on how directory technology is used within Netscape to support and enhance basic business processes and to help employees be more productive.

Team LiB

◀ PREVIOUS

NEXT ▶

Directory Drivers

Two main motivating factors drove Netscape to deploy an organizationwide LDAP directory:

1. **Ensuring the quality of the Netscape Directory Server.** The original motivation behind the internal deployment was to provide a test bed for Netscape Directory Server. The internal deployment provided valuable feedback and helped the software engineers improve the quality of the product before it was made available for beta testing.
2. **Supporting directory-enabled applications.** Nearly all of Netscape's software products are directory-enabled. As each new directory-enabled product neared completion, it was deployed internally both to test the product in a real-world situation and to improve the productivity of Netscape's employees.

Directory Service Design

In this section we describe Netscape's directory design and how it was developed.

Needs

As in many businesses, Netscape's directory requirements were largely determined by two needs: the need to support a wide range of applications and the need to deploy them in a manageable and cost-effective manner.

As mentioned previously, Netscape's IS department is charged with providing computing support to employees at the main campus and satellite offices. With few exceptions, IS provides all the hardware, software, and network connectivity for every employee. (This arrangement is in contrast to the decentralized support methods that will be described in [Chapter 26](#).)

Despite this centralization, minor political problems arose when the directory service was being deployed. Obtaining a synchronization file that contained information on all full-time employees, contractors, temporary employees, and contingent employees proved difficult, and the issue had to be escalated to senior management. After the synchronization file was obtained, IS staff needed to spend a significant amount of time modifying existing IS and HR work processes to tie them into the directory. Connecting everything to the directory required some modifications to existing HR database tables and associated user interfaces, which required cooperation from the HR staff.

Another constraint was that the staff involved in the directory design and deployment, although extremely skilled, needed some time to become familiar with directory technology. Fortunately the developers responsible for the directory server software (Netscape Directory Server) were literally next door. The proximity of the two groups provided immense benefits for both IS and the Directory Server software engineering team; questions from IS were handled quickly and accurately, and Engineering obtained valuable information on how the product worked in an actual deployment. In essence, the Netscape internal directory deployment still provides an ongoing directory pilot that continually feeds back useful information to the Netscape Directory Server engineering group, ultimately resulting in a higher-quality, more functional product.

One of the primary motivations for the directory deployment was to support Netscape's suite of directory-enabled server products, including Netscape Messaging Server (electronic mail), Netscape Collabra Server (Usenet news and internal discussion groups), Netscape Enterprise Server (internal Web publishing), Netscape Certificate Management System (for public key certificates), and Netscape Calendar Server (for workgroup scheduling).^[1] Each of these products is directory-enabled and can utilize the directory for user and group management and access control. In addition, employees have become very dependent on Netscape Phone Book, an internally developed application that allows quick and easy lookup of contact information and location information for people and meeting rooms.

[1] Some of these Netscape products are no longer available.

The directory is also used for specific work processes, serving as the focal point for creation or removal of resources when a person joins or leaves Netscape. For example, when an employee is hired, a directory entry is created, triggering processes that create Unix and Microsoft Windows NT domain accounts, assign telephone numbers, add the employee to the security/badging database, allocate a network port, and include the employee on certain

companywide, divisionwide, and workgroup-specific mailing lists.

Early in the design process, Netscape IS made the assumption that directory users needed to feel confident that the central repository of directory data was secure, stable, valuable, and accurate. These needs led to the conclusions described here:

- **Security.** Users must believe that the information in the directory is protected from unauthorized use by other employees and persons outside the company. This means that each attribute type stored in the directory should be reviewed and a policy established for how that data may be used, who may use it, and how it will be protected.
- **Stability.** Users must feel that the data in the directory is stable. Data should not change unless in response to a change in the environment (for example, a new phone number). When users are allowed to change data in the directory, their changes should not arbitrarily disappear.
- **Value.** Users must find some value in the directory data. They should find useful data that allows them to perform their jobs, and they should not be forced to wade through nonessential or inappropriate data. The directory can also provide value by allowing the directory data to be put to use in new and novel applications (see the section titled Leveraging the Directory Service later in this chapter).
- **Accuracy.** Users must feel that the data in the directory is accurate. Stale data reduces the usefulness of the directory and causes users to use other, less efficient means of obtaining the information they need.

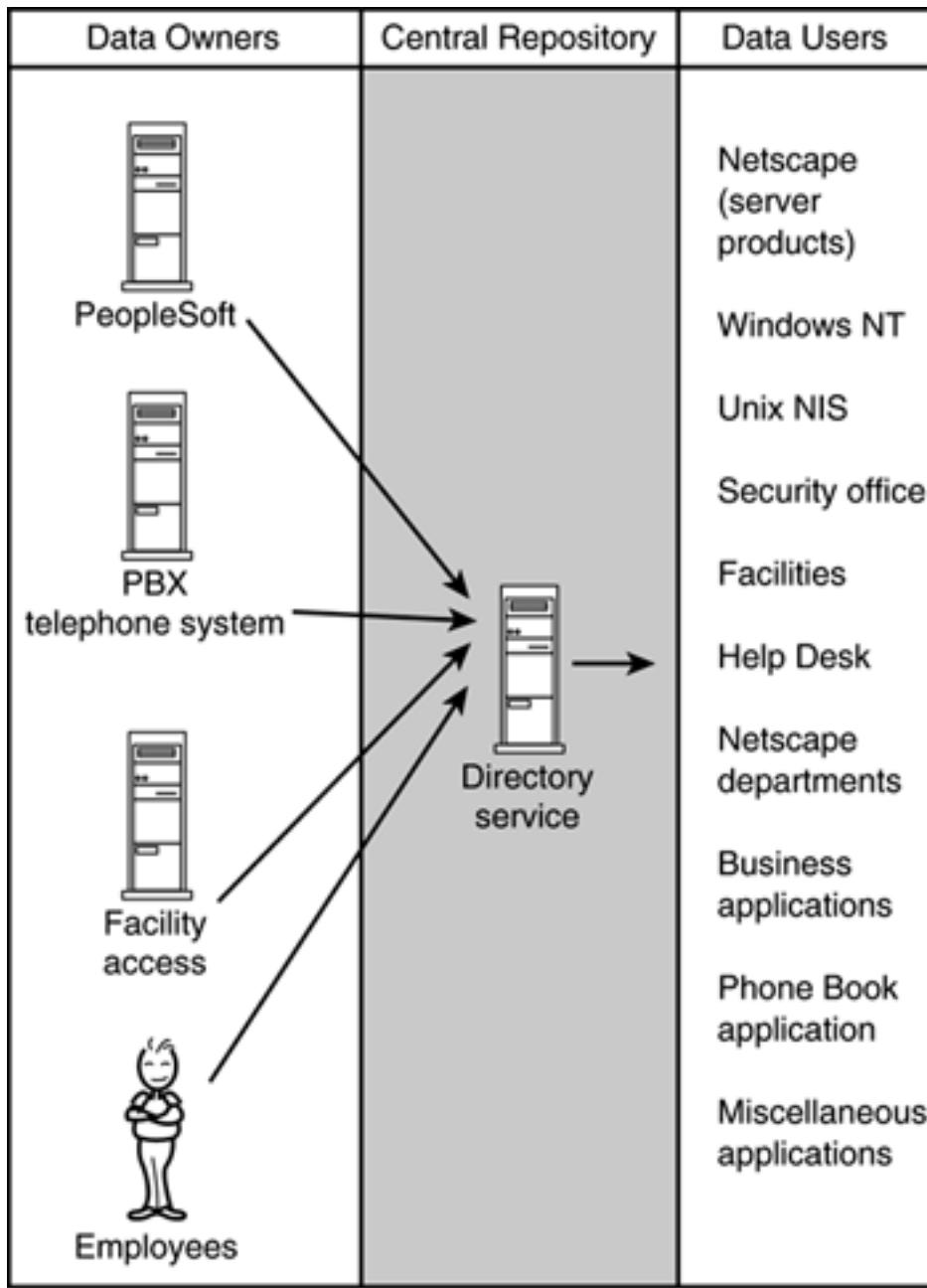
In addition, the directory itself must be flexible enough to accommodate new applications, and it must scale well so that users always receive adequate performance.

Data

As mentioned in the previous section, the primary drivers behind the directory service revolved around the directory-enabled applications slated for deployment. Those applications had well-defined schema requirements, which made identification of the necessary data elements very straightforward.

What was less straightforward, however, was identifying the "owner," or authoritative source, of each data element. The Netscape IS organization began by developing a conceptual framework for understanding the directory, external data sources, and external users of the data. [Figure 24.1](#) shows this framework.

Figure 24.1. A Conceptual Framework for Data Sources



Data owners are the databases that contain authoritative enterprise data. For example, data stored in the PeopleSoft system is considered authoritative for certain elements, such as a person's name and home postal address. The private branch exchange (PBX) telephone system is considered authoritative for office telephone numbers. In the Maintaining Data section of this chapter, we'll describe the process that synchronizes data between the directory and other data sources.

Data is collected from the authoritative data sources and placed into the directory service. In this role, the directory serves as a *central repository*, a rendezvous point where applications can meet to obtain data about people, groups, and other business-related objects. The directory serves a valuable role by publishing and grouping the data synchronized from the various distinct data sources.

The directory service itself is considered authoritative for some data elements, such as e-mail addresses. For these data elements, the directory functions simultaneously as the data owner and the central repository.

Data users are the applications, business processes, and people that make use of the directory data stored in the central repository.

After the fundamental data model had been developed, the Netscape directory deployment team refined the data plan further by establishing the following basic guidelines for deciding that a given data element should be stored in the directory:

- **If two or more applications require the data element.** If only one application needs it, the element does not need to be shared.
- **If the directory data changes in response to events that occur in the Netscape environment.** This guideline stands in contrast to other traditional databases that change in response to minute-by-minute data processing needs, such as an order fulfillment database. This policy is meant to discourage use of the directory as a relational database.
- **If the directory data can facilitate location independence.** Netscape believes that storing user preferences is a completely valid and desirable use of its directory service.

Next a detailed data element inventory was completed. This inventory described each data element, the directory's attribute name for it, the authoritative source, and the consumers that use it. [Table 24.1](#) shows a portion of that inventory.

Table 24.1. A Portion of Netscape's Data Element Inventory

Data Element	LDAP Attribute Name	Authoritative Source	Consumers of the Data Element
E-mail address	mail	Directory	Messaging server
			Phone Book application
			E-mail client address books
Business telephone number	telephoneNumber	PBX system	Phone Book application
			E-mail client address books
Employee's manager	manager	PeopleSoft system	Phone Book application
Employee photograph	jpegPhoto	Facility access	Security photograph (badging)

Finally, a set of synchronization procedures was developed to propagate changes from external authoritative data sources such as the PeopleSoft and PBX systems into the directory. These synchronization procedures are the core of Netscape's directory coexistence strategy and are designed to capture changed data in the external authoritative data sources and apply updates to the directory.

Schema

The schema used in the central directory is composed of three distinct sets of schema definitions:

1. Schema definitions packaged with the directory-enabled applications, such as Netscape Calendar Server, Netscape Messaging Server, and Netscape Certificate Management System.
2. The POSIX (Portable Operating Systems Interface) schema, as defined in RFC 2307. These schema elements define the user information stored by POSIX-compliant Unix operating systems. They allow the directory to store user information for use by the company's Unix systems.
3. Custom schema elements that were developed in-house. These elements support Netscape business processes. For example, one attribute describes the termination date for an employee who is leaving the company. Automated processes revoke access to directory-enabled applications after the termination date.

Netscape IS staff members added the custom schema definitions only after carefully examining available schema elements and determining that no existing attribute types met Netscape's needs.

All schema extensions were implemented as auxiliary, or "mix-in," object classes. To illustrate, consider the definition of the `nscpPerson` object class shown in [Listing 24.1](#) (note that `nscpPerson-oid` should be replaced by a real object identifier, or OID).

Listing 24.1 The `nscpPerson` Object Class

```
( nscpPerson-oid

  NAME 'nscpPerson'

  DESC 'Netscape-specific person information'

  SUP top

  AUXILIARY

  MUST uid

  MAY ( nscpBadgeImage $

        nscpPersonExpDate $

        nscpCurContactInfo $

        nscpBuildingNum $
```

```
nscpBuildingLev $  
nscpHarold  
)
```

This auxiliary object class contains only the Netscape-specific attributes. Entries in the directory that describe people have both the `inetOrgPerson` and `nscpPerson` object classes. Thus a person entry may contain any of the attributes allowed by either the `inetOrgPerson` or the `nscpPerson` object class. In this way, Netscape extended the schema with business-specific attributes without altering the standard object class definitions. The added attributes were as follows:

- **nscpBadgeImage**. The digitized photograph from the employee's ID badge. This attribute may be viewed only by campus security personnel (to protect employee privacy).
- **nscpPersonExpDate**. The date after which an entry is no longer valid. When an employee leaves the company, this attribute contains the date of his or her last day of employment.
- **nscpCurContactInfo**. A free-form text attribute that may be set by the employee to indicate how he or she may be contacted if not in the office. Inclusion of this information allows employees to be tracked down for important customer support issues, for example.
- **nscpBuildingNum**. The number of the building where an employee works.
- **nscpBuildingLev**. The building floor where an employee works.
- **nscpHarold**. The legal name of the employee. Because the common name (`cn`) attribute contains the name by which people commonly know a person, there needs to be another attribute to hold the employee's legal name. In case you were wondering where the name of this attribute came from, one member of the directory deployment team went by a name other than his legal name, which happened to be Harold.

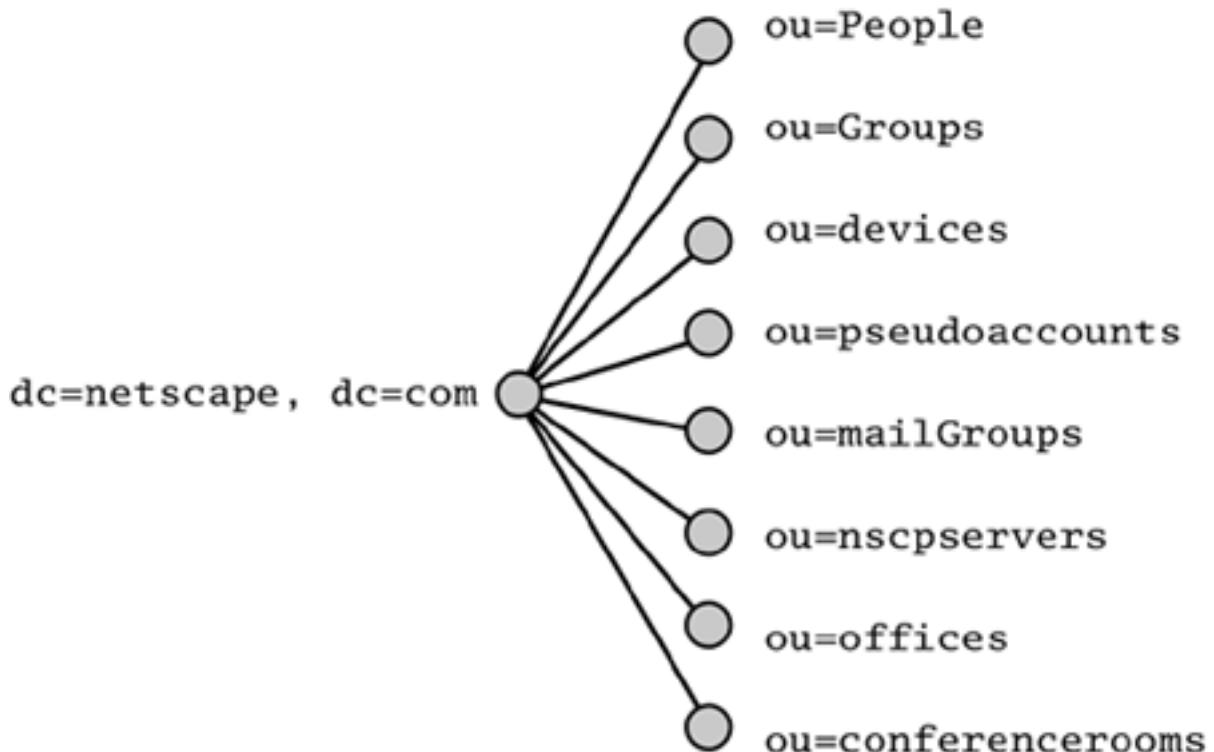
Namespace

[Figure 24.2](#) shows Netscape's directory namespace. Netscape decided to use domain component naming for the top or global portion of the namespace (`dc=netscape,dc=com`). The organizational units that define subnamespaces beneath the top level are as follows:

- **ou=People**. Holds information about all Netscape employees and contractors. The namespace within this container is flat (that is, all people entries are located directly below the organizational unit container).
- **ou=Groups**. Holds groups used for access control decisions. For example, the list of mailing list administrators is defined in this container. The namespace underneath this container is also flat.

- **ou=devices**. Will hold information about various network-accessible devices such as printers (not currently used). IP address, MAC (Media Access Control) address, owner, serial number, and other information will be recorded for each device.
- **ou=pseudoaccounts**. Holds definitions of directory identities that are used for special purposes. For example, when one directory server needs to authenticate to another directory server as part of a replication session, it uses one of these identities. The namespace under this node is flat.
- **ou=mailGroups**. Holds group definitions used for routing electronic mail to groups of employees and to external e-mail addresses. The mail group entries are maintained in a separate directory tree because of limitations in earlier versions of internal mailing list management software. In the future, a single **ou=Groups** container will hold any type of group, including mail groups. Like the **ou=Groups** container, the namespace under this node is flat.
- **ou=nscp servers**. Holds contact and configuration information for all the officially supported servers deployed within Netscape.
- **ou=offices**. Holds contact and location information for Netscape offices located throughout the world.
- **ou=conferencerooms**. Holds definitions of all the conference rooms at Netscape's main campus and at most of the other campuses. This collection of entries is used by Netscape Calendar Server to provide information about the various conference rooms and to support scheduling of the rooms. The namespace under this node is also flat.

Figure 24.2. Netscape's Directory Namespace



There are several important points to note about this namespace design. To some extent, the portion of the global directory namespace that Netscape uses at this time is unimportant because the internal directory is not available outside Netscape's firewalls. However, if

Netscape should want to share this information with external business partners in the future, it will be necessary to choose a portion of the namespace that cannot conflict with namespaces in use by other businesses.

With this constraint in mind, Netscape chose to use a naming method that leverages the existing Domain Name System (DNS) infrastructure to provide a unique namespace. The top-level name of `dc=netscape,dc=com` is derived from Netscape's assigned domain name, `netscape.com`. The other option would have been to use the X.500 naming method, which is based on a country–locality–organization name hierarchy. Using this scheme, Netscape's tree would have been rooted at `o=Netscape Communications Inc.,st=California,c=US`, or possibly `o=Netscape Communications Inc.,c=US`. Netscape chose the DNS-derived names because they are shorter and do not require registration with any standards organization (other than the domain name that was registered in the early days of the commercial Internet).

After the directory suffix was chosen, Netscape's IS department needed to decide how the directory information would be structured. The final decision took into consideration the following factors:

- IS knew it would be using Netscape Directory Server software to provide the service.
- Netscape Directory Server's access control model allows for directory access decisions to be based on attribute values within entries. This means that it is not necessary to divide the directory into subtrees for the purposes of delegating administrative authority, so it is much more feasible to use a flat namespace.
- The number of entries contained in the directory would be, at most, counted in the tens of thousands. This maximum is well within the capacity of a single Netscape directory server. Therefore, it was unnecessary to partition the directory among multiple servers.
- Employees within Netscape can and do move between departments. If the organizational hierarchy were part of the naming scheme, it would be necessary to change an entry's name (move it, in other words) whenever the employee changed departments. Such a requirement has several undesirable consequences, especially when there are references in the directory to the person's entry (for example, in electronic mail groups).
- During deployment of the directory server, the primary directory-enabled applications being supported were Netscape Messaging Server, Netscape Collabra (News) Server, and Netscape Calendar Server. All these products avoid making assumptions about the structure of the directory namespace, which provides great flexibility.

A flat namespace beneath `ou=People` was chosen to avoid the headaches caused when people move between departments. Within the `ou=People` container, the `uid` (user ID) attribute was chosen as the naming attribute for entries. This attribute has two desirable properties. First, it must be unique anyway because the employee's electronic mail address is `uid@netscape.com`. Second, because the namespace is flat, any attempts to create a duplicate `uid` would be rejected by the directory server (because the new entry would have the same distinguished name [DN] as an existing entry).

One trade-off in using a flat namespace under `ou=People` is that straightforward directory browsing is less interesting; all employees appear in a single, flat list of names. Browsing is more useful when the namespace reflects some organizational hierarchy. Netscape directory deployers realized that multiple hierarchies were present in the directory data. For example, each employee at Netscape is associated with a particular department, and this represents one type of hierarchy present in the data. Each employee also has a manager, and the reporting hierarchy represents a different type of hierarchy.

Instead of favoring one type of hierarchy, the deployers opted to make the namespace independent of any particular hierarchy. To construct multiple alternative hierarchies, such as the reporting hierarchy, DN-valued attributes are used. For example, each employee's entry contains a **manager** attribute that holds the DN of the employee's manager. A directory application can use this information to support several different views of the data. For example, an application might allow a user to jump to an employee's manager or to browse the directory according to employee/manager reporting relationships.

Topology

The topology chosen for the Netscape directory deployment is very simple: The entire directory is contained in a single partition. Two considerations led to this decision:

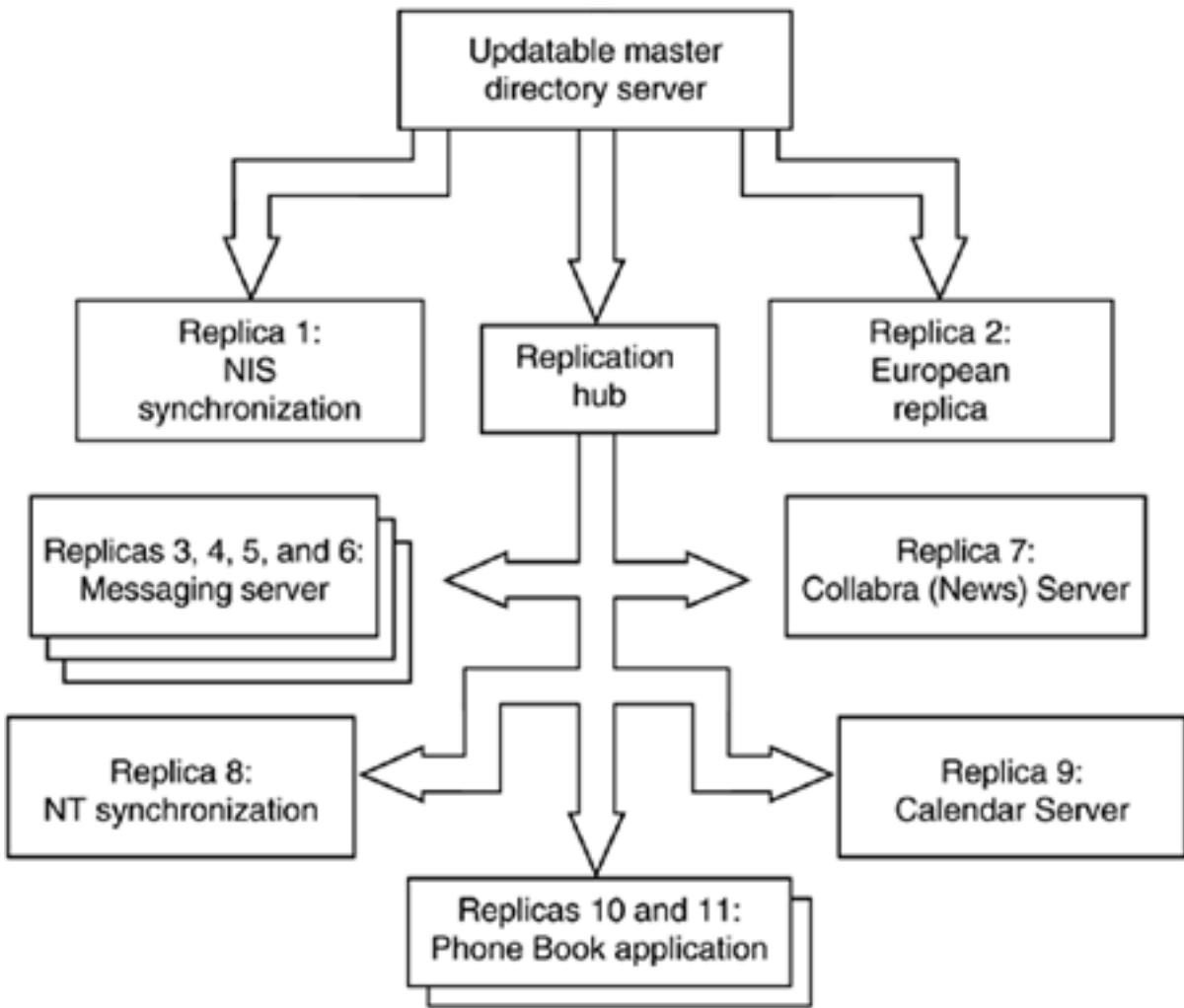
1. The number of directory entries is small relative to the capacity of the server hardware and software (the server can handle hundreds of thousands of entries or more). Therefore, it is not necessary to partition the data to achieve sufficient capacity. All the entries in the directory are contained in each replica (replicas are discussed in the next section).
2. Netscape had a central IS organization; therefore, there was no need to support separate areas of administrative control. Any required delegation of authority can be handled via the access control mechanisms supported by the server. Therefore, delegation of authority required no partitioning.

For these reasons, a single-partition design was chosen.

Replication

The primary motivator behind Netscape's replication architecture was to provide sufficient directory capacity for its directory-enabled applications, such as messaging and calendaring. [Figure 24.3](#) shows the overall replication architecture.

Figure 24.3. Netscape's Replication Architecture



There are several important points to note about the replication architecture:

- Each of the main directory-enabled applications (messaging, calendaring, and phone book) are co-located with a directory server running on the same host. These applications are configured to query the local directory server first and then fall back to another server if the local server is unavailable (which is extremely rare). Co-location of a directory replica with each application-specific server improves response time and protects against network outages.
- At the time of the original deployment, the Netscape Directory Server software supported only a single-master replication scheme in which one server is designated as the updatable master and all other servers are read-only. (The latest version of the Netscape Directory Server software does support multiple masters, and the directory team is now revising its replication architecture.) Because Netscape's plans called for a relatively large number of replicas (11 replicas provide service to end users and applications), it was undesirable to try to feed all the replicas from the one updatable master. Instead, the updatable master server mainly feeds a replication hub, which is then responsible for distributing the changes to most of the remaining replicas. This arrangement has the advantage that the updatable master is freed from the task of updating many replicas; it needs to update only three replicas. The result is better update performance for clients.

This particular design leverages the strengths of Netscape Directory Server—namely, its ability to store many entries while still providing excellent performance. If your directory server software limits you to a small number of entries per partition relative to how many entries you will have, a more complicated replication architecture may be required.

On the other hand, the presence of the replication hub is a concession to the single-master nature of Netscape's Directory Server software prior to version 6. Because there is a single

updatable server, it is highly desirable to keep the server's load as light as possible—which is why a replication hub is used. If your directory server software supports multimaster replication, you may be able to eliminate the concept of a replication hub from your architecture. Replication hubs do improve the directory service's ability to distribute data widely and also help accommodate network architectures that limit the ability of applications to access the master directory servers directly.

Privacy and Security

The security design for Netscape's internal directory service deployment focused on two major issues: controlling access to directory data and protecting against certain types of security attacks.

The data contained in Netscape's internal directory is composed primarily of public information that can be viewed by any employee. However, several attributes of person entries are potentially sensitive, such as the employee home address and telephone number.

To fully understand the security issues associated with each directory attribute, Netscape's IS department developed a security matrix. For each type of object stored in the directory, the matrix describes the attributes present in that entry and the security restrictions in effect for those attributes. For example, the electronic mail address for a given employee is considered public information; it is essential for other employees to know a given employee's e-mail address. On the other hand, the employee's home address is considered sensitive information and must be protected in order to address employee privacy concerns.

During the development of this security matrix, Netscape's Legal department offered advice about which attributes might be considered sensitive information. For each attribute considered sensitive, a policy was developed describing who may view and update it. [Table 24.2](#) shows a portion of the final security matrix.

The update policy also allows an individual employee to decide whether certain information should be published in the directory at all. For example, the authoritative source for the employee's home telephone number is the PeopleSoft database. Unlike most of the other directory attributes, the home phone number is not automatically synchronized from PeopleSoft to the directory. Instead, an employee must place his own home phone number in the directory if he wants it there (the same policy is used for home postal address as well). This kind of approach places complete control of sensitive information in the hands of the employee.

Another part of Netscape's directory security design involved protecting the directory from several types of security attacks. Because Netscape's internal directory is used only by employees and contractors, certain assumptions make it unnecessary to protect against some types of attacks. Employees are trusted to conform to company policy, so disciplinary action can be levied against an employee who perpetrates a directory attack. Such attacks (of which there have been none to date) include denial-of-service attacks and impersonation attacks.

Table 24.2. An Excerpt from the Netscape Security Matrix

Attribute Type	Sensitivity	Readable by	Updatable by
----------------	-------------	-------------	--------------

<code>cn</code>	Low	Everyone	PeopleSoft synchronization process
<code>homeTelephoneNumber</code>	High	Everyone	Self (not automatically placed in the directory service by the PeopleSoft synchronization process)
<code>jpegPhoto</code>	High	Self, campus security personnel	Badging system (run by security personnel)

Nevertheless, other precautions need to be taken because Netscape sometimes uses leased lines provided by a third party. Netscape does not control the physical security of those lines; therefore, unencrypted data cannot be guaranteed safe from eavesdropping. The following precautions are in place:

- Application and directory servers are physically secured in machine rooms where they may be accessed only by operations staff and security personnel. This precaution provides a level of protection against theft or physical compromise of the servers (for example, breaking in by rebooting a server and starting a privileged shell).
- Login to directory and application server machines may be accomplished only via the Secure Shell (SSH) protocol. This precaution prevents network eavesdroppers from intercepting communication from a system administrator to a server.
- For the same reason, an administrator's workstation must connect to the Netscape administration server and the directory server over encrypted Secure Sockets Layer (SSL) sessions (either HTTP-over-SSL or LDAP-over-SSL). This precaution ensures that administrative work is not susceptible to eavesdropping.
- SSL is used to encrypt all replication updates flowing from the master server to the replication hub and from the replication hub to each consumer server. This precaution protects the directory data (especially sensitive user data) from eavesdropping and ensures that the data is not modified in transit. This was especially important for updates that went to a directory replica located in one of Netscape's European offices because the network connection was a leased line owned and operated by a third-party network provider.
- All directory servers support LDAP-over-SSL (LDAPS) in case a particular employee wants to communicate over an encrypted connection. The servers support standard LDAP as well, of course.

In summary, the security architecture for the directory service and the data it contains are tuned to Netscape's particular business requirements and culture. As business requirements change, the security architecture is reviewed and updated as necessary.

Directory Service Deployment

In this section we describe the steps taken to put Netscape's internal directory service into production.

Product Choice

Netscape is somewhat unique in that it is a leading developer of directory technology. Netscape's products include a high-performance directory server (Netscape Directory Server) and client software development kits (SDKs). Furthermore, all of the products in Netscape's suite of server software are directory-enabled. Because of cost, performance, and support issues, and because Netscape's developers knew that using their own products would help find bugs and prove the robustness of software they planned to sell to others, Netscape chose to deploy its own products.

Piloting

The pilot phase of the directory deployment was rather informal. The first directory pilot was conducted by the software engineers working on the 1.0 version of Netscape Directory Server, even before the software was officially released as a product. The developers created a directory that held information about employees (including telephone numbers, office locations, and electronic mail addresses), and they publicized the availability of the directory. The engineers also created an HTML-based interface that allowed employees to search for entries and update their own directory entries.

At roughly the same time, the Netscape client product development team was adding LDAP capabilities to a prerelease version of the Netscape address book. The presence of the pilot directory allowed these developers to distribute prerelease copies of the Netscape client software to employees and obtain valuable feedback on the design and usability of the address book user interface.

20/20 Hindsight: Enabling Schema Checking

When the initial pilot phase began, not much thought had been put into analyzing schema requirements. To make it easier to add new attributes and object classes, schema checking was disabled on the master directory server (which allowed any attribute with any name to be added to any entry in the directory). The developers running the pilot directory found the lack of schema checking convenient because it allowed them to begin adding new attributes and object classes to the entries in the directory without modifying the schema configuration, assigning OIDs, or restarting the server. (Recent versions of the Netscape Directory Server no longer require a server restart in order to add new schemas.)

Unfortunately, disabling schema checking also allowed inconsistencies to creep into the directory data. Because developers in other groups were also using the pilot directory to become familiar with LDAP, some unknown and misspelled attribute types were introduced into the data.

Although these inconsistencies weren't such a serious problem in the pilot phase, they became troublesome when the pilot data was imported into the production service in which schema checking was enabled. The pilot data required a rather extensive cleanup before it could be imported. The pilot data could have been discarded; however, many employees had come to depend on the data stored in

the pilot server, so the developers decided to retain as much data as possible.

In retrospect, to avoid this problem it would have been better to enable schema checking even during the early stages of the pilot.

After initial deployment of the directory, several other pilots focused on new directory-enabled applications that were being developed. One such application was a mailing list manager application, which replaced an older application that used proprietary databases to manage internal and external mailing lists. These lists were migrated into the directory, and a new, HTML-based management interface was developed, piloted, and deployed.

Putting Your Directory Service into Production

The production rollout of the directory server coincided with the rollout of Netscape Messaging Server. These steps were followed:

1. The server hardware was purchased and installed. The directory server and messaging server were both installed on a single host, so memory and disk space were sized appropriately. Hardware accelerator cards were installed to improve performance during the servicing of LDAP connections made over SSL.
2. Network ports (100BaseT Ethernet) were installed, and network adapters were installed in the hosts.
3. Backup procedures were developed for the messaging server. Because the directory that resides on the messaging server host is a replica and therefore can be recovered from the master server, directory data on the replica is not backed up. The directory server configuration files are backed up.
4. An initial test was performed to ensure that the messaging server was able to support access via the Post Office Protocol (POP) and Internet Message Access Protocol (IMAP) and could receive electronic mail via the Simple Mail Transfer Protocol (SMTP). This test also verified that the messaging server was able to retrieve directory data.
5. User accounts were moved from the existing, non-directory-enabled messaging server to the new servers. The service was then in full production.

Directory Service Maintenance

In this section we describe the various procedures used to maintain Netscape's internal directory service.

Data Backups and Disaster Recovery

Directory data is backed up daily via the online backup capabilities of Netscape Directory Server. With this capability, data can be backed up to disk while the server is running and accepting updates; it is not necessary to shut down the server or place it in read-only mode (see [Chapter 4](#), Overview of Netscape Directory Server, for more information on the online backup feature). The backup files are then archived to tape along with the directory server configuration data. The backup tapes are stored in a secure location off-site (along with backup tapes of other critical applications) to protect against data loss in a disaster and to protect the security of the data.

Although tape backups are made, the primary method of restoring a directory server is to obtain recent directory data from a replica. The Netscape software's continuous replication feature is used to keep replicas always closely in sync; therefore, the replicas provide a more up-to-date copy of the directory than the daily backups do. Tapes are still required, however, in the event that directory data is damaged (for example, entries are deleted) and the damage propagates to all of the replicas.

The disaster recovery plan for Netscape's internal directory leverages the extensive disaster recovery plan already in place for the Netscape.com Web site. In a nutshell, the plan provides for continuous operations through a combination of alternate power sources at primary sites and alternate sites that contain replicas of critical data and applications.

Maintaining Data

Netscape's directory needs to coexist with several other data repositories. For some data elements, the external repositories are the authoritative source for the data. For other data elements, the directory itself is authoritative. This section describes the procedures used to maintain the relationships between the external data repositories and the directory, and the procedures used to maintain data for which the directory itself is the authoritative source.

Three main external repositories are synchronized with the directory:

1. The Windows NT domain user and group database
2. Unix Network Information Service (NIS)
3. The PeopleSoft system used for HR data

This section discusses these repositories and the process used to synchronize their data with that stored in the directory service itself.

The Windows NT Domain User and Group Database

A special tool was written to run on Netscape's Windows NT primary domain controller (PDC) and synchronize the NT user database with the directory. Specific attributes in the directory

for NT users, NT accounts, NT passwords, NT directory structures, and NT access control lists (ACLs) are read from the directory, and the Windows PDC information is synchronized to match the directory. This process also ensures that the password stored in the Microsoft Windows NT authentication database matches the password stored in the directory. If it does not, the synchronization tool overwrites the NT password with the authoritative password from the directory. The NT sync process starts up every three minutes, searches the directory for all entries that have changed since the last NT sync run, and then synchronizes them.

To simplify management, all Netscape NT users are part of one NT domain. If Netscape ever splits its NT user and group information into multiple NT domains, a separate synchronization service will need to run on each PDC, and a policy will need to be implemented that maps newly created user entries to a particular domain.

NIS

Netscape's Unix workstations use NIS to distribute user and group information to all workstations throughout the company. Like the NT user database, NIS represents a repository of user information that must be kept in sync with the directory. Custom scripts were developed that read directory data and generate several NIS maps, which are then imported into the NIS master server. These maps include the `passwd` map (user and password information), the NFS `automounter` map files, and the `aliases` map, which the sendmail message transfer agent (MTA) uses to expand mail aliases. The NIS sync process runs every 20 minutes.

PeopleSoft

The PeopleSoft system is the authoritative source for most of the information about employees. It is very important that the directory data be kept in sync with PeopleSoft.

For example, a new employee should immediately be able to access vital services such as Unix login, Microsoft Windows login, and e-mail. Similarly, when an employee leaves the company, access to these facilities must be revoked immediately.

This synchronization is accomplished with a set of Perl scripts that reconcile PeopleSoft data with the directory. Based on PerLDAP (available from the Mozilla Web site at <http://mozilla.org/directory/perldap>), these scripts also validate and clean up data when necessary, such as when the PeopleSoft data lacks attributes required by the directory. The scripts also report any exceptional conditions they encounter, such as entries with missing `manager`, `organizationalUnit`, or `businessCategory` attributes to people who can repair the problems. The PeopleSoft synchronization process runs once per hour. A variant of the PeopleSoft synchronization script that Netscape uses is available on the Web at <http://www.mozilla.org/directory/tools/ldaptools.html>.

Data Whose Authoritative Source Is the Directory Itself

The directory itself is the authoritative repository for some data elements, such as e-mail addresses. Unlike data elements that are synchronized from external sources, it's possible to delegate authority for these directory-mastered data elements to other people using the directory servers' access control capabilities.

One example is the home mailing address for employees. As mentioned earlier in the chapter, values for this data element are stored in the PeopleSoft database, but they are not synchronized to the directory (out of concern for employee privacy). However, employees

are free to add `homePostalAddress` attributes to their directory entry if they want to. Also note that the home address information is not synchronized back to the PeopleSoft database (although someday it might be).

Monitoring

Netscape has deployed an extensive SNMP-based monitoring system that focuses on monitoring network devices such as routers, hubs, and server network interfaces. As in many other organizations, the group that provides this monitoring is distinct from the group that manages the directory service. Coupled with the fact that the earliest versions of Netscape Directory Server did not support monitoring via SNMP, the separation between the two groups led the directory deployment team to develop its own set of monitoring tools that check whether the following conditions hold:

- All directory servers are running and responding to requests.
- All replicas are in sync with the master server.

If any of these tests fail, an alert is raised and an appropriate individual is notified via electronic mail and pager.

In the future, monitoring of the directory may be integrated with the other network monitoring functions.

Leveraging the Directory Service

A successfully deployed directory can be leveraged to support new directory-enabled applications. One such example is Netscape's Maps application, which is also called The Locator.

Netscape's Mountain View campus has many meeting rooms. Like many other companies in Silicon Valley, Netscape names these conference rooms after cartoon characters, movie names, famous people, and so on. Although this naming convention is fun and part of the culture, it can make it difficult to figure out where you need to be for your next meeting unless you happen to know, for example, that the Miles Davis conference room is on the ground floor of Building 12. Similarly, employees' office numbers are arbitrary numbers that do not give any indication of the location of the office within the building. Some way was needed to help employees determine the physical location of offices and conference rooms.

To address this need, the Maps application was designed. This Web-based, directory-enabled application allows a person's name or conference room name to be looked up in the directory. If a match is found, a graphical map is displayed in the Web browser window, and the location of the cubicle or meeting room is highlighted. The location (in x,y coordinates relative to the upper left corner of the map image) is stored in an attribute of the directory entry. After the map is drawn in the browser window, a small cross-hair cursor moves across the screen to highlight the location. The Maps application is very popular, and it is used hundreds of times each day.

Many other directory-enabled applications have been developed that leverage the Netscape internal directory service. Here are a few examples:

- **The Phone Book application.** This Web-based application was described briefly earlier in the chapter. In addition to helping employees find contact information about other employees, this application allows employees to update their directory information (for example, they can add a home postal address or an "on vacation" reply notice that is picked up by Netscape's e-mail service).
- **OrgChart, an organization charting tool.** This Web-based application is driven entirely from data that is stored in the directory service. In particular, it uses **manager** attribute values from person entries to construct a graphical view of the Netscape organization. The OrgChart application is also linked to the Maps and Phone Book applications.
- **PKI deployment.** Netscape uses public key certificates for secure e-mail and to protect access to sensitive Web content. The directory service is used to distribute certificates to people and servers and to help employees acquire new certificates and renew existing ones.
- **TinderBox, a source code tracking and continuous build system.** This system, a variant of which is used by the mozilla.org open-source effort, tracks recent source code changes and repeatedly compiles all the software that Netscape's engineers are working on. The TinderBox system pulls some information from the directory to make it easier to contact a software engineer in the event that an error was introduced because of code they added or changed.
- **Protected Web content.** Although most information that resides on Netscape's internal network is freely available to all employees, some sensitive information is not. The servers for the Web sites that store sensitive information all use the

Netscape Enterprise Server software to restrict access, and the Enterprise Server software in turn consults the directory service for authentication and to make group-based authorization decisions.

- **Directory-enabled utilities created by individual developers within Netscape.** Because it is easy to develop directory-enabled applications for Netscape's internal service, over the years many small utilities have been created by individuals within Netscape. For example, a Perl script named utc (which stands for "up the chain") searches the directory to find a person and his manager, and his manager's manager, and so on—all the way up to the top of the company (at the very top of the chain is the chairman of AOL Time Warner, Netscape's parent company). The utc utility and other custom applications are written and shared among Netscape's employees to leverage the directory data and increase their productivity.

Directory Deployment Impact

In what ways did things change for the better after Netscape deployed a directory service? The following are a few of the ways that the directory has improved life for Netscape employees:

- The employee hiring and termination processes are now automatically triggered by changes made to directory data. Before the directory service existed, hiring and termination were tedious processes that required departmental administrators to submit separate work orders for account activation/deactivation, computer system orders, network drop, and so on. The Netscape IS department now treats the directory as an authoritative repository and triggers events such as account activation on the basis of the directory data. This automation has streamlined the processes and provided significant cost savings.

AOL Time Warner Update

Since Netscape's initial directory deployment, the company merged with America Online, and two years later the combined companies merged with Time Warner to form AOL Time Warner. The AOL Internal Computing organization was quick to see the value of the directory service that Netscape had deployed. The contents of the directory were quickly expanded to include entries for all AOL employees and contractors, and during the term of the Sun-Netscape Alliance it also held information about Sun employees who were assigned to Alliance projects such as the development of iPlanet Directory Server. After the merger with Time Warner, an existing Time Warner employee directory was integrated with the Netscape service.

What additional changes have been made to the directory service since Netscape became part of AOL and since AOL merged with Time Warner? New schemas have been added to support AOL's business processes, to support identification based on AOL screen names in addition to Netscape user IDs, and to allow people to be located on the basis of their instant messaging ID. A plug-in for Netscape Directory Server has been developed that supports authentication using screen names and their associated passwords. More directory replicas have been deployed to increase availability and redundancy. Several new directory applications have been deployed, including an electronic purchasing workflow system and a "trouble ticket" system that is used to track software and hardware problems reported by employees and customers.

To support applications that require access to companywide information, AOL Time Warner is deploying a new corporate directory service that includes information on all of its 80,000 employees. The new service is based on the same principles and the same underlying technology that were used in Netscape's original deployment. Meanwhile, the original Netscape directory service will continue to operate to support the specific needs of the AOL division. AOL Time Warner discovered that it would be very difficult to deploy one directory service that would meet the needs of all the AOL Time Warner divisions, simply because the needs differ significantly. Instead, each division will feed data from its own sources to the corporate directory, and each division may choose to run a divisional directory.

- Administration of user and group information is now centralized. Instead of each application requiring its own user and group database, applications obtain this information from the central directory, either directly or via a synchronization process.
- Windows NT domain user and group information is synchronized from the directory, reducing maintenance costs.
- Employees can change all their passwords (Unix, NT, and directory) at one time using a Web-based application called Password Central. The passwords are pushed from the directory into NT and Unix NIS, ensuring that the same password is available everywhere.
- A variety of useful directory-enabled applications are available, such as the Phone Book and Maps applications. And new directory-enabled applications are constantly being developed.

Summary and Lessons Learned

Netscape's directory deployment has been a great success on two fronts. First, the company has benefited from a directory service in the ways you, having read the earlier chapters of this book, have probably come to expect. For example, more information is available to all employees, and business processes have been streamlined, particularly the process of getting the computing environment for a new employee up and running. Second, the deployment has met its goal of being a useful test bed for Netscape Directory Server and the directory-enabled applications that Netscape develops.

In any complex process such as deploying a directory, there is always room for improvement. Here are a few words of advice based on lessons learned during the directory deployment at Netscape:

- Obtain buy-in from senior management early in the planning process to help ensure that you obtain the necessary cooperation from other departments that are crucial to the success of the directory deployment.
- Plan the directory namespace well in advance. It will become increasingly difficult to change the namespace as the directory gains momentum and becomes more and more essential to business processes.
- Pay particular attention to the privacy needs of your directory users. It's best to obtain the opinion of your Legal department when deciding which attributes will be available in the directory. Allow sufficient time for review.
- Even if your directory software allows you to disable schema checking, it's almost always a mistake to deploy the directory in this way. Leaving schema checking enabled improves the quality of your directory data and avoids painful data cleanup work.
- Start early when planning for coexistence with external data sources, especially when those sources are owned by another group. Try to get the other groups to fully communicate the structure of those data sources so that you don't miss important data repositories when migrating data into the directory.
- Do not work in a vacuum. Include as many participants as possible in the initial design phase for directory coexistence, work process changes, resulting information changes, and user interface development. Having a broad base of participation ensures that everything is covered and helps provide buy-in to the project.

Further Reading

An Approach for Using LDAP as a Network Information Service (RFC 2307). L. Howard, 1998. Available on the World Wide Web at <http://www.ietf.org/rfc/rfc2307.txt>.

LDAP Integration Tools for PeopleSoft. Available on the World Wide Web at <http://www.mozilla.org/directory/tools/ldaptools.html>.

Netscape Server Software. Product information is available on the World Wide Web at <http://enterprise.netscape.com>.

PeopleSoft Resource Management Software. Product information is available on the World Wide Web at <http://peoplesoft.com>.

Looking Ahead

In this chapter we have taken a look at a real-world centralized directory service deployment. In [Chapter 25](#), Case Study: A Large Multinational Enterprise, we will examine a much different organization, a large multinational company that has 225,000 employees and a decentralized information services organization.

Chapter 25. Case Study: A Large Multinational Enterprise

- Overview of the Organization
- Directory Drivers
- Directory Service Design
- Directory Service Deployment
- Directory Service Maintenance
- Leveraging the Directory Service
- Summary and Lessons Learned
- Further Reading
- Looking Ahead

In this chapter we look at how a large multinational enterprise might design and deploy an LDAP directory service. Large organizations typically spend more time and money up front to design a directory service than small organizations do. The extra investment is necessary for several reasons:

- Large organizations tend to be more decentralized in terms of geography, political control, and information systems management. It is difficult to bring an organizationwide service like a central directory online because of all the physical, organizational, and political boundaries that separate the different parts of the organization.
- Data about important organizational assets (including people) is typically controlled by a group different from the one charged with deploying the directory service. This separation of control increases the amount of coordination required to obtain the data and deploy the directory.
- The complexity and cost of the components that make up the directory service are significant enough that a large investment in design and piloting is needed to ensure that the service smoothly enters its production phase.
- The cost of a design error is greater than in smaller deployments. Because of the large number of servers rolled out and the complexity of data maintenance and related issues, redesigning a large-scale directory late in the design process causes more pain and expense than redesigning a smaller deployment would.

The case study discussed in this chapter is fictional; it is not based directly on any real-world directory deployments. However, this case study draws from knowledge of several actual deployments with which we are familiar. The large, multinational enterprise we describe is patterned after a manufacturing company such as Ford Motor Company, Boeing, or Intel. The name of our fictitious company is HugeCo. This case study assumes that HugeCo's directory service has been deployed as a production service for approximately six months.

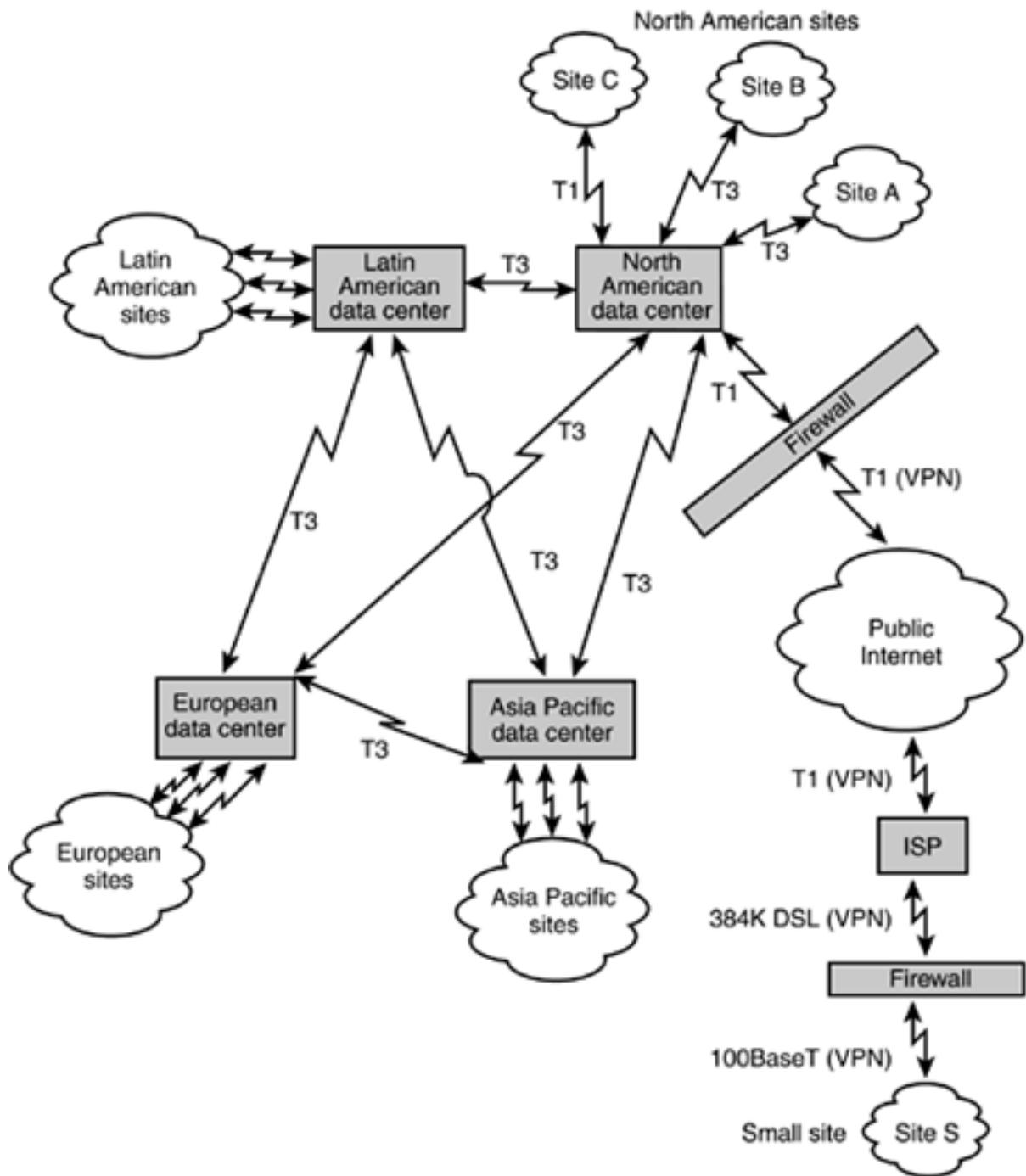
Overview of the Organization

HugeCo is a widely dispersed organization with 225,000 employees and contractors located in 34 countries around the world. Most employees live and work in countries where HugeCo's manufacturing facilities are located: the United States, Canada, Mexico, the United Kingdom, Germany, Japan, and Australia. HugeCo has relationships with hundreds of other companies that serve as suppliers, dealers, and customers for the goods it produces.

HugeCo has a strong, well-funded central information services (IS) organization that operates out of the corporate headquarters site in Chicago, Illinois. Four regional IS departments serve the needs of each of the HugeCo sites in North America, Latin America, Europe, and the Asia Pacific region. The primary role of the central IS organization is to design the network and applications architecture, run several central services (such as a corporate e-mail hub), and coordinate the activities of HugeCo's regional IS departments. The regional IS departments are responsible for installing and maintaining all hardware and software within their regions.

The information technology infrastructure at HugeCo includes a modern, high-speed TCP/IP network. Within each site, a combination of 10BaseT and 100BaseT Ethernet systems is used to connect desktop machines and servers. Larger sites also have an internal 1-gigabit fiber backbone network. Most sites are connected with leased T1 (1.54Mbps) or T3 (45Mbps) lines to regional data centers that are connected to each other with leased T3 lines. Some of HugeCo's smallest sites are connected via the public Internet via IPsec-based virtual private network (VPN) technology and local Internet service providers (ISPs). [Figure 25.1](#) shows an overview of the HugeCo network.

Figure 25.1. The HugeCo TCP/IP Network



The central IS organization, along with the regional IS departments, provides a rich, robust set of e-mail, discussion forum, group scheduling, and document publishing services to HugeCo's professional employees. Here are the most important applications in use at HugeCo:

- Sun ONE Messaging Server provides e-mail routing, delivery, and reading services for most of HugeCo's sites. Sendmail.org's sendmail message transfer agent (MTA) and the University of Washington's imapd mail store software are used by some sites, but both packages are being phased out. A variety of desktop e-mail clients are used, including QUALCOMM's Eudora, Netscape Communicator 4.x and 7.x e-mail, and Microsoft's Outlook clients.
- Lotus Notes servers and clients are used to support group discussion forums and certain collaborative applications developed in-house. Notes is used only by HugeCo's executive staff and their assistants. The Notes users also use the Notes client to read their e-mail.
- Netscape Enterprise Server is used to provide document publishing services and to support a variety of custom Web-based applications. Microsoft's Internet Explorer browser is the most widely used Web client, but Netscape's Navigator browser is still

used by about one-third of HugeCo's employees.

Most applications are hosted on Sun Enterprise servers running the Solaris 8 Unix operating system. The notable exceptions are the servers for Lotus Notes, which are Dell PowerEdge servers running Microsoft Windows 2000 Server.

The central IS organization also employs several dozen software engineers who develop and maintain a variety of custom Web-based applications. These software engineers also assist employees in the regional IS departments and within HugeCo departments that are developing their own applications.

HugeCo benefits from strong leadership that makes it possible for the central IS organization to dictate which software and hardware are used throughout much of the company. Still, there is a high degree of independence within regions, especially with respect to how services and data are administered.

Team LiB

◀ PREVIOUS

NEXT ▶

Directory Drivers

To serve its internal customers better, HugeCo's central IS organization decided to design and deploy an organizationwide LDAP directory service. The motivation to create a comprehensive corporate directory was driven by the following goals, which addressed some immediate and some long-term needs:

- **To improve internal communication.** HugeCo's executive staff handed down a mandate that internal communication should be improved. The IS organization decided that a good way to streamline internal communication would be to make it easier to locate and share information about people and shared resources such as conference rooms. Before the arrival of the organizationwide LDAP directory, most applications had their own database of users, groups, and resources, which added to end-user confusion and created a high administrative burden. IS managers believed that less time would be wasted if the quality, consistency, and timeliness of data that employees use to initiate communication with other employees improved. In addition, if a shared, organizationwide directory were deployed, the elimination of redundant information could lower the data management costs.
- **To make it easier to develop and deploy Web applications.** As more Web-based applications were being developed and deployed, it became clear that shared authentication and a shared group data source were needed. A common directory service used by all the custom applications would allow HugeCo to provide a form of single sign-on and decrease the costs of developing, deploying, and maintaining the custom applications. It would also lower the cost of entry so that smaller departments without the resources to develop and maintain their own infrastructure could develop and deploy their own custom applications.
- **To increase security and privacy.** Over the next two years, HugeCo plans to issue public key certificates that employees can use to authenticate to e-mail and workflow applications. Deployment of the necessary public key infrastructure (PKI) is a time-consuming task, but it is made easier by the presence of a directory service. In the short run, the HugeCo directory provides a single point of management for passwords and distribution of role-based access rights used by Web-based applications. In addition, some departments within HugeCo have started to use Netegrity's SiteMinder product to control user access to their custom Web-based applications.

The security and privacy of the directory data itself are important issues because of the wide geographical dispersion of HugeCo's employees and because some of the corporate traffic is tunneled through the public Internet with VPN technology.

- **To improve communication with dealers and suppliers.** HugeCo's management team knows that the company needs to maintain close ties to its dealers (sales offices) and suppliers to stay competitive. Because these entities operate independently of HugeCo, they do not share any information technology infrastructure. Without exception, HugeCo has a more highly developed infrastructure and more expertise than its dealers and suppliers. At the present time, most communication outside the company is exchanged via simple file transfers and fax machines. Although we do not discuss it in this chapter, HugeCo hopes to leverage the knowledge gained from deployment of its corporate directory service to create a directory to link it closely and securely with its suppliers and dealers. A directory deployment motivated by those needs is discussed in [Chapter 26](#), Case Study: An Enterprise with an Extranet.

Directory Service Design

The first phase in HugeCo's directory service venture was to analyze the company's needs and design the service itself. This task was accomplished over four months by a team of five people from the central IS organization. An IS employee from each of the four regional IS departments was asked to participate in a weekly conference call and occasional face-to-face meetings to review the detailed directory design as it was drafted.

HugeCo's chief information officer (CIO), who oversees all IS activities within the company, was involved early in the decision to develop an LDAP directory service. To bring the CIO up to speed and expose her to the short- and long-term benefits that a directory would provide, the IS group arranged a private presentation and demonstration for her. An entire week was spent preparing for the meeting with the CIO, but the goodwill and support it generated helped the directory deployment process immensely. Having the CIO's support lowered the political barriers that occasionally threatened to block progress of the design and deployment of HugeCo's directory service.

Needs

When HugeCo's senior IS managers formed the directory design team, they clearly spelled out the reasons for creating the directory. The ready availability of this information, along with the fact that the overall computing environment was already well known and documented by the central IS organization, made the directory needs analysis a relatively easy step.

The following applications were identified as good candidates to take advantage of a directory service immediately:

- Sun ONE Messaging Server (used for e-mail transfer and delivery)
- A variety of e-mail clients (used by end users to manage their e-mail)
- Netscape Enterprise Server (used to publish Web documents and to support some custom applications)
- Netegrity SiteMinder (used to control access to some custom applications)
- Lotus Notes (used for group discussions)

This list is in order of priority, although the explosion of Web-based applications that was under way made Netscape Enterprise Server and Netegrity SiteMinder increasingly important. Integration of the new LDAP directory service with Lotus Notes was a relatively low priority because of the small user base of Notes (limited to HugeCo's executive staff). In contrast, almost all of HugeCo's professional employees use e-mail daily.

The directory design team also thought it would be cost-effective and useful to replace printed departmental employee phone lists with a Web-based phone book application. This application could even be used by directory administrators and Help Desk staff to perform simple directory maintenance tasks, such as correcting an employee's contact information or resetting a password.

Because of the wide geographical distribution of HugeCo's facilities, it was clear from the start that the directory service would need to be available to end users and applications 24 hours a day, 7 days a week. The IS staff was already required to provide similar levels of service for most of the shared network infrastructure and important applications such as e-mail.

The design team also took time to develop a directory project plan. This plan included

several goals, each with an associated target date:

- Publish a directory service requirements document (within 1 month).
- Complete and publish the initial directory design (within 3 months).
- Complete a directory pilot involving at least two regions of HugeCo (within 6 months).
- Deploy a production-quality directory service used for phone book lookups, e-mail routing, and at least one important Web-based application (within 12 months).

Everyone agreed that this was an aggressive set of goals, but the team thought all of them could be achieved. The initial draft of the requirements document was created and published soon after the needs analysis stage was complete. The leader of the design team shared the goals with IS upper managers and other IS employees—but not before padding the target dates by a few months to give the team more room to meet them.

Data

As soon as the list of applications was in hand, the HugeCo design team worked with an existing data architecture team within the IS organization to craft a directory data policy statement. The core tenets of the statement included the following points:

- Data shared by more than one application should be stored in the directory service.
- The authoritative source for each data element stored in the directory must be defined.
- Extremely sensitive or private data should not be stored in the directory service unless there is an operational reason to do so. This kind of data includes payroll and employee benefit information.
- All legal requirements, including country and regional privacy laws in effect where HugeCo operates, must be followed.

IS quickly produced a list of data elements needed by each application. This was a relatively easy task; many of the applications supported LDAP already and therefore came with a set of well-documented data elements and schemas. For HugeCo's custom Web-based applications, a Netegrity SiteMinder role-based access control system used by some of the newest applications was adopted.

Roles at HugeCo include `helpDesk`, `emailAdmin`, `systemAdmin`, `payrollClerk`, `seniorManager`, `lineManager`, and dozens of others. The Web-based applications use a combination of the roles that a person is allowed to adopt, along with information about the department the person is in, to grant or deny access to application functions. Many of the newest applications manage workflow tasks in which the appropriate flow of information is especially critical. The directory design team decided that in addition to a standard department number data element, a list of roles should be stored with each person entry in the directory service to facilitate the use of role-based access control by applications. Because Netegrity SiteMinder itself uses LDAP directories and can pass custom attributes along to applications, the role information can be used by SiteMinder-based applications as well as other applications.

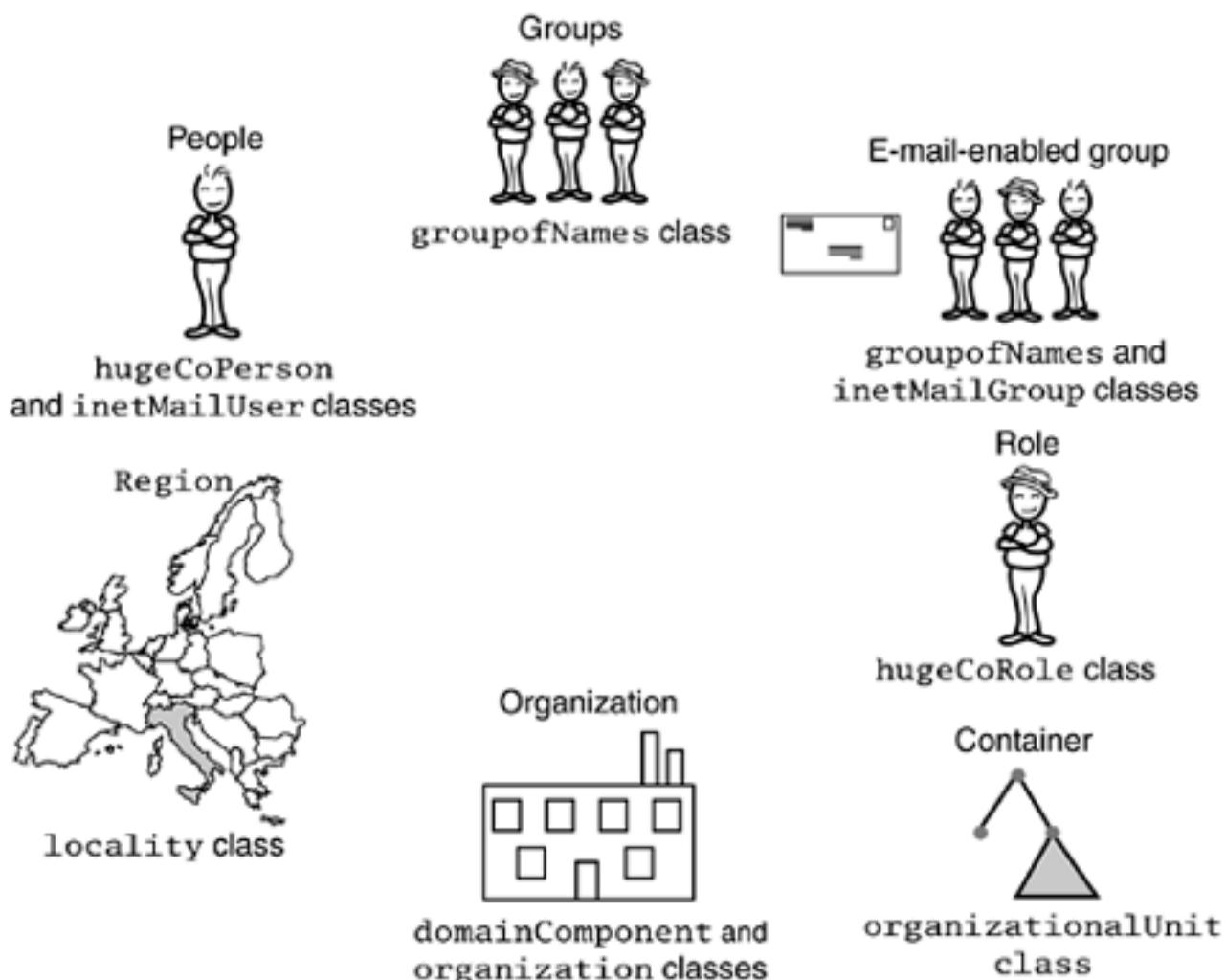
Next the team created a table of all the data elements that would initially need to be stored in the directory service. As recommended in [Chapter 7](#), Data Design, the table included a comprehensive set of characteristics for each data element. The team discovered that most of the data elements were shared by several applications and would not contain any sensitive information, which fit well within the data policy statement. The team also noted that many of the data elements had to accommodate data values of several different character sets and languages because of the multinational makeup of HugeCo. Although most official communication is in English, most of the employees at sites located in non-English-speaking countries communicate internally using their native language.

The final data design task was to examine an existing inventory of HugeCo's data sources. The team's goal was to determine which data elements that were to be stored in the directory were already available in existing data stores. The team discovered that many of the employee-related data elements were already present in HugeCo's PeopleSoft human resources (HR) system. A series of meetings was planned with the PeopleSoft experts within the IS organization to determine how best to manage the relationship between the HR database and the directory service.

Schema

To accommodate the data elements, the HugeCo team chose to use standard schemas and schemas provided by the application vendors when available. In some cases the team members found that they needed to subclass standard schema elements to meet their needs. [Figure 25.2](#) shows some of the LDAP object classes selected for use in the HugeCo directory.

Figure 25.2. HugeCo Object Classes



To allow the storage of values for a few custom attributes in each employee entry, the standard `inetOrgPerson` object class from RFC 2798 was subclassed to create a `hugeCoPerson` object class. The first three custom attributes defined as part of the `hugeCoPerson` class were

1. `hugeCoEmployeeRole`, to store the list of roles described earlier

2. **hugeCoBuildingNumber**, to store a number that uniquely identifies the building where an employee works

3. **hugeCoBuildingFloor**, to store the floor or building level where an employee works

[Listing 25.1](#) shows the definition of the **hugeCoPerson** object class.

Listing 25.1 The `hugeCoPerson` Object Class

```
(  HUGECPERSON-OID

    NAME  'hugeCoPerson'

    DESC  'HugeCo person object class'

    SUP  inetOrgPerson

    MAY ( hugeCoEmployeeRole $ hugeCoBuildingNumber $ hugeCoBuildingFloor )

)
```

HUGECPERSON-OID must be replaced by a real OID (one was assigned by the HugeCo directory deployment team). Subclassing a standard LDAP object class meant that maximum compatibility with existing directory-enabled applications could be maintained, and company-specific extensions could be made cleanly and easily in the future.

For the routing of e-mail, the **inetMailUser** auxiliary object class that is recommended by Sun for use with its messaging server was the natural choice. For e-mail and access control groups, the standard **groupOfNames** object class was used with the **inetMailGroup** auxiliary class added to e-mail-enable entries as needed.

A new structural object class called **hugeCoRole** was created to allow the entire list of available roles to be stored as a set of directory entries. [Listing 25.2](#) shows the design of the **hugeCoRole** object class.

Listing 25.2 The `hugeCoRole` Object Class

```
(  HUGECOROLE-OID

    NAME  'hugeCoRole'

    DESC  'HugeCo role object class'

    SUP  top

    STRUCTURAL

    MAY  description

    MUST  cn
```

)

The `cn` attribute is used to hold the name of the role itself. `HUGEGEROLE-OID` must be replaced by a real OID (one was assigned by the HugeCo directory deployment team). [Listing 25.3](#) shows a sample role entry.

Listing 25.3 A Sample `hugeCoRole` Entry

```
dn: cn=helpDesk,ou=Roles,ou=Global,dc=hugeco,dc=com
objectclass: top
objectclass: hugeCoRole
cn: helpDesk
description: people who handle questions sent to the IS help desk
```

Storing all the available roles in the directory service allows directory-enabled administration tools and Web-based applications to easily obtain and present the complete list of roles to their users if needed. Although some directory server products support a roles feature, to be as vendor-neutral as possible HugeCo chose to use its own schema for roles.

To accommodate international data, the language attribute subtypes (as defined in RFC 2596) and Unicode (UTF-8) character set specified by LDAPv3 were adopted. By convention, an ASCII-only value that is not tagged with any language is always stored for all attribute types that have values. This practice provides maximum compatibility with directory-enabled applications that cannot handle Unicode data.

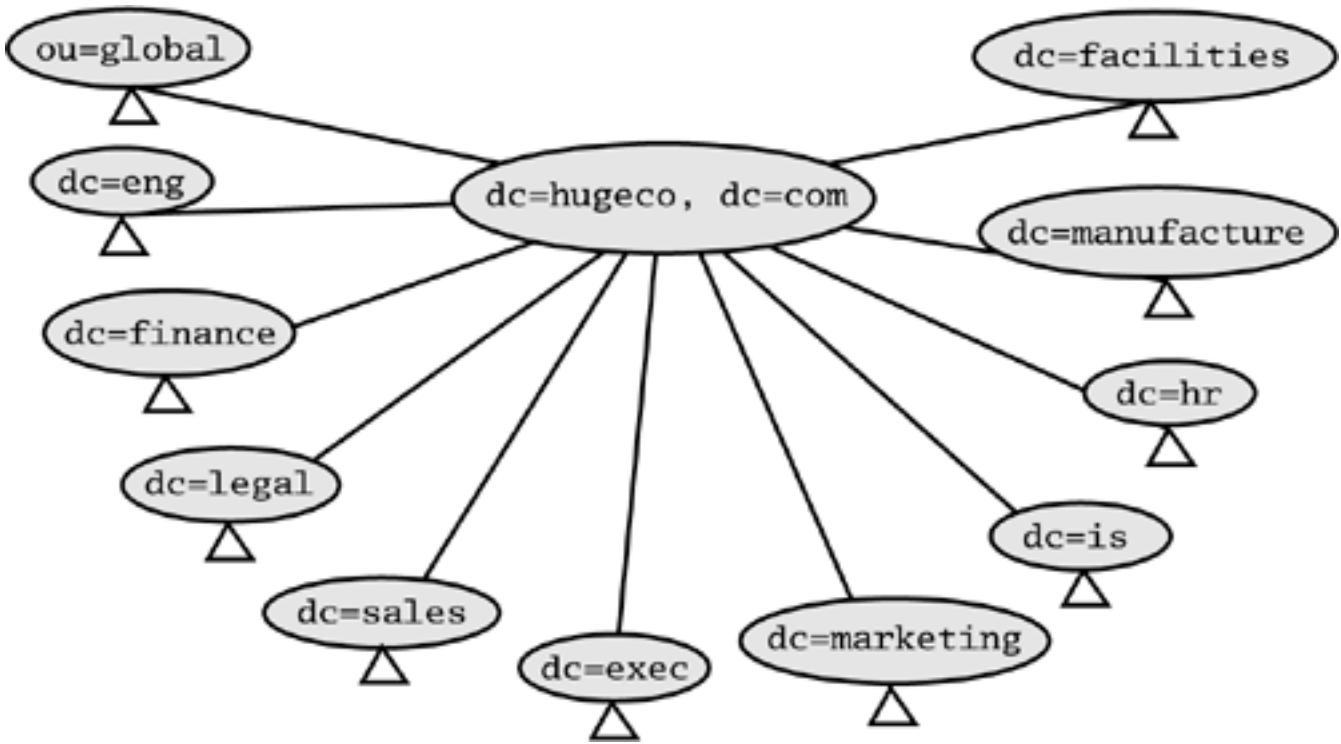
Namespace

The HugeCo team made an early decision to follow a domain component (`dc`) naming scheme. Using this scheme allowed the team to leverage the existing Domain Name System (DNS) names for the top part of the namespace. The top-level suffix of `dc=hugeco,dc=com` was simply derived from the company's existing Internet domain name (`hugeco.com`).

The HugeCo design team wanted to keep the directory namespace fairly flat to reduce the overhead of managing distinguished name (DN) changes. A hierarchy was incorporated into the namespace to allow for delegation of responsibility and provide subtree roots to facilitate replication.

[Figure 25.3](#) shows HugeCo's initial directory namespace design. The small triangle shown beneath each node in the figure represents the subtree that exists below the node.

Figure 25.3. The Initial Design of HugeCo's Directory Namespace



Splitting the directory tree along DNS subdomain lines produced a useful hierarchy of directory entries. This scheme fit well with the way e-mail accounts are managed within HugeCo. It allowed different access control to be placed at the top of each `dc` container, enabling a rough mapping of authority based on departments.

However, this namespace design did not mesh well with HugeCo's need to replicate its directory information in all locations. The main reason was the complexity of managing and setting up replication among 11 directory partitions. The HugeCo team decided to revise the namespace design and separate the tree along regional lines instead.

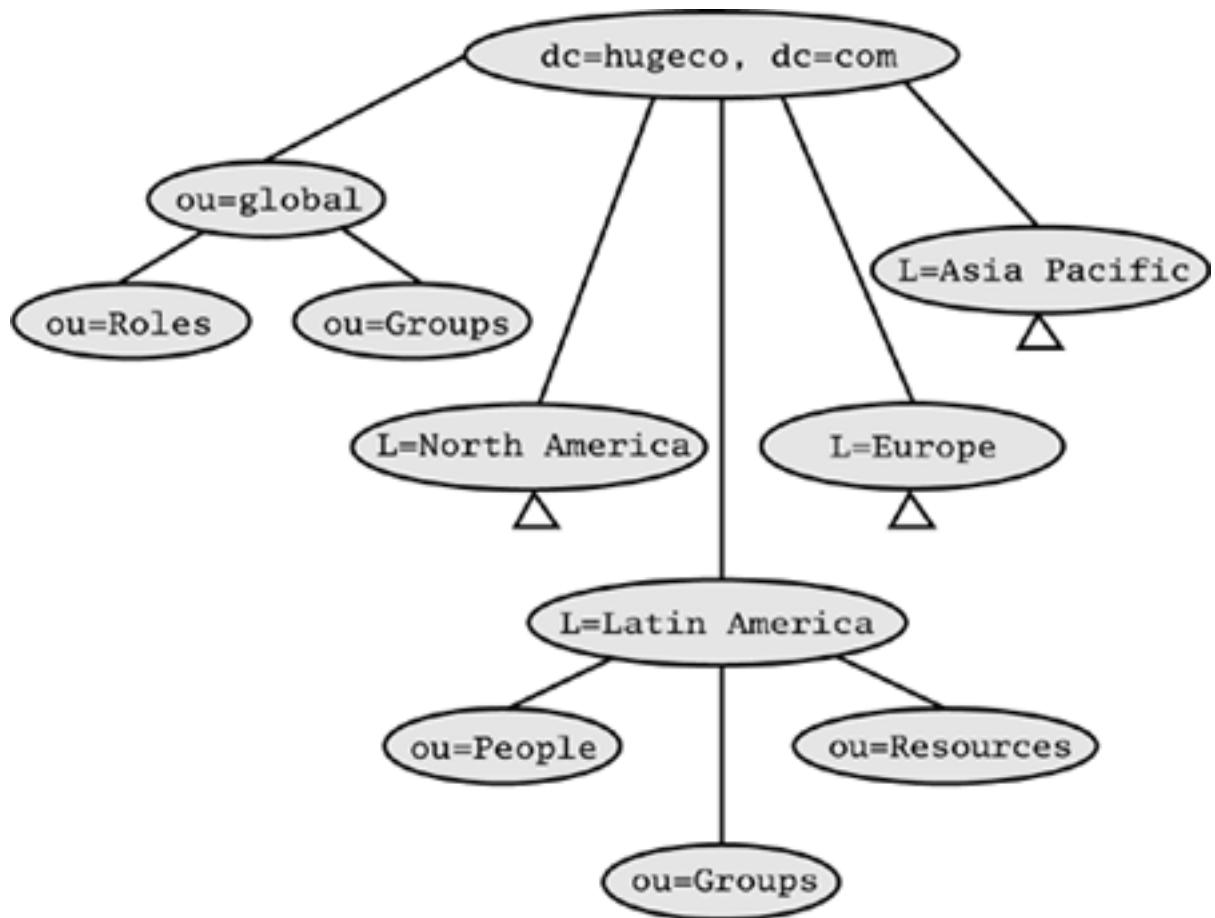
[Figure 25.4](#) shows the final, revised namespace design. In the final design, one `locality` entry is used to represent each region. The relative distinguished name (RDN) of these entries is formed from the `L` (locality) attribute. [Listing 25.4](#) shows the entry for the Asia Pacific region.

Listing 25.4 A Sample HugeCo `locality` Entry

```

dn: L=Asia Pacific,dc=hugeco,dc=com
objectclass: top
objectclass: locality
L: Asia Pacific
description: includes all sites in Japan and Australia
  
```

Figure 25.4. The Final Design of HugeCo's Directory Namespace



There are three consistently named subtrees under each of the four locality entries: `ou=People`, `ou=Groups`, and `ou=Resources`, as the Latin America node in [Figure 25.4](#) illustrates. The namespace is flat under each of these containers. An `ou=Global` subtree rooted at the same level as the regional entries is used to store groups that are global to the entire company, along with the master list of HugeCo roles.

Limiting the number of major directory partitions to only five makes replication easier to manage. In the new namespace design, access control rules that refer to values stored in specific directory entries rather than ones that refer to the structure of the tree provide for delegation of authority. For example, a departmental directory administrator group can be granted authority over its own employees' directory information by creation of an access control rule that refers to the `departmentNumber` attribute within each employee's entry.

Every HugeCo employee and contractor is assigned a unique, nonrecycled identification (ID) number by the HR department. The directory team decided to use these ID numbers to form the RDNs of all people entries. [Listing 25.5](#) shows a typical employee entry.

Listing 25.5 A Sample `hugeCoPerson` Entry

```

dn: employeeNumber=12397,ou=People,L=North America,dc=hugeco,dc=com
objectclass: top
objectclass: person
objectclass: organizationalPerson
objectclass: inetOrgPerson

```

```
objectclass: hugeCoPerson  
objectclass: inetMailUser  
  
cn: Babs Jensen  
  
cn: B Jensen  
  
sn: Jensen  
  
employeeNumber: 12397  
  
uid: babs  
  
telephoneNumber: +1 810 555 1234  
  
roomNumber: 42  
  
manager: employeeNumber=9837,ou=People,L=North America,dc=hugeco,dc=com  
  
mail: babs@hugeco.com  
  
mailhost: mail-1.manufacture.hugeco.com  
  
mailQuota: 25000000  
  
hugeCoBuildingNumber: 747  
  
hugeCoBuildingFloor: 3  
  
hugeCoRole: lineManager  
  
...
```

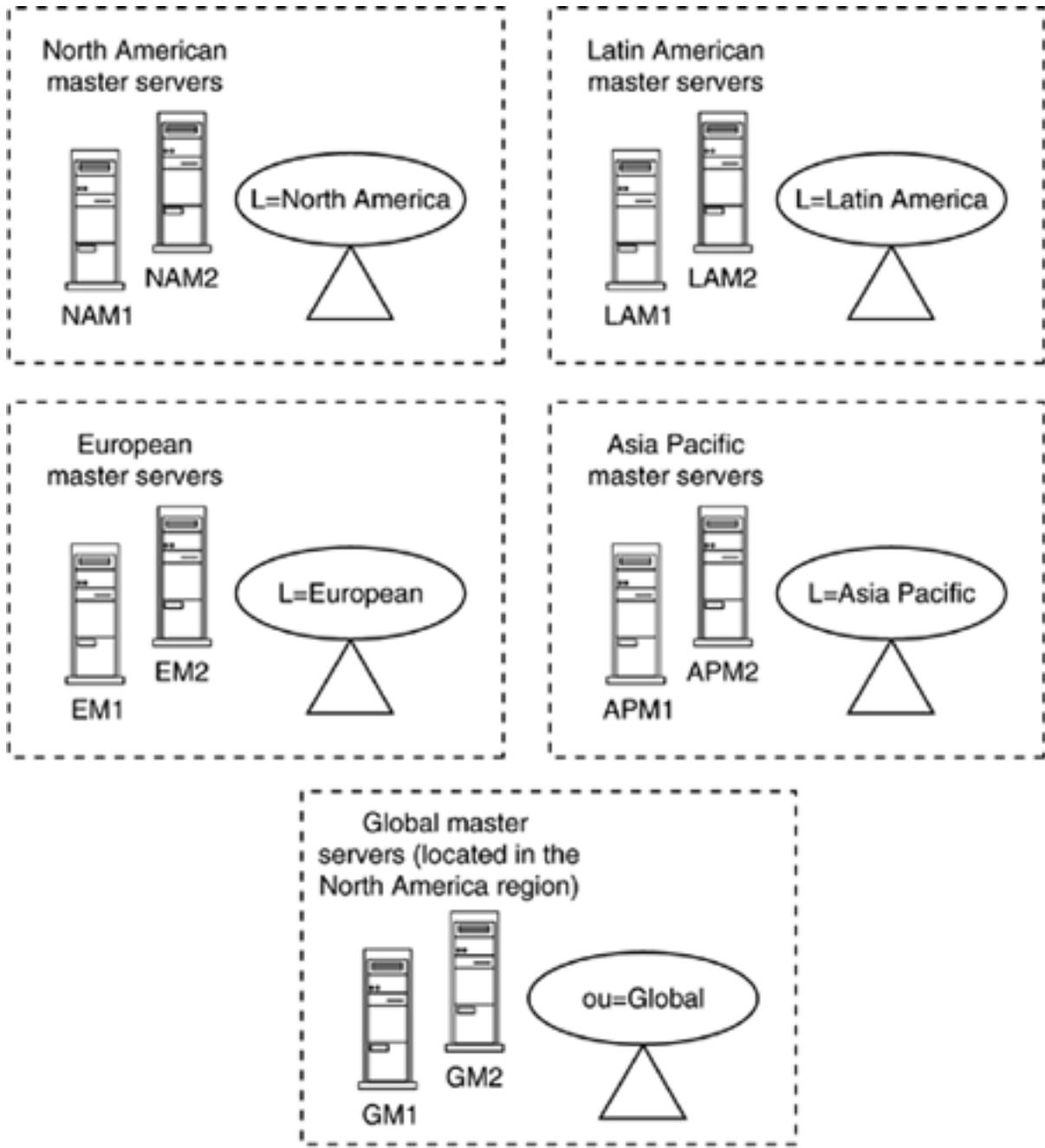
As we have seen already, the four location entries are named with the `L` (locality) attribute. The `cn` attribute is used to form the RDNs of all other entries—for example, groups and roles.

Topology

The HugeCo IS organization's general strategy for managing important, shared services such as e-mail is to deploy a small number of relatively large server machines within each site. The company followed this same philosophy when designing its directory topology. For directory-enabled applications that have high search and read performance requirements, the design allows a directory replica to be located on the same network segment as the application. Because HugeCo was already leaning toward choosing directory server software that can support more than one million entries in one server database, there was no need to partition the data across servers for scaling reasons. Partitioning along regional lines to accommodate HugeCo's replication strategy is discussed in the next section.

Special thought was given to the configuration of the read/write master directory servers where updates are made by LDAP clients. To avoid introducing any single points of failure, the HugeCo designers chose to use directory server software with multimaster replication capabilities to reduce the risk of total failure. The master servers are deployed in pairs, with one pair in each region, except for the North America region, where a second pair is deployed to hold the non-region-specific `ou=Global` subtree. Data for each region is mastered within that region only. [Figure 25.5](#) shows the five pairs and the data held by each server.

Figure 25.5. HugeCo's Paired Master Servers

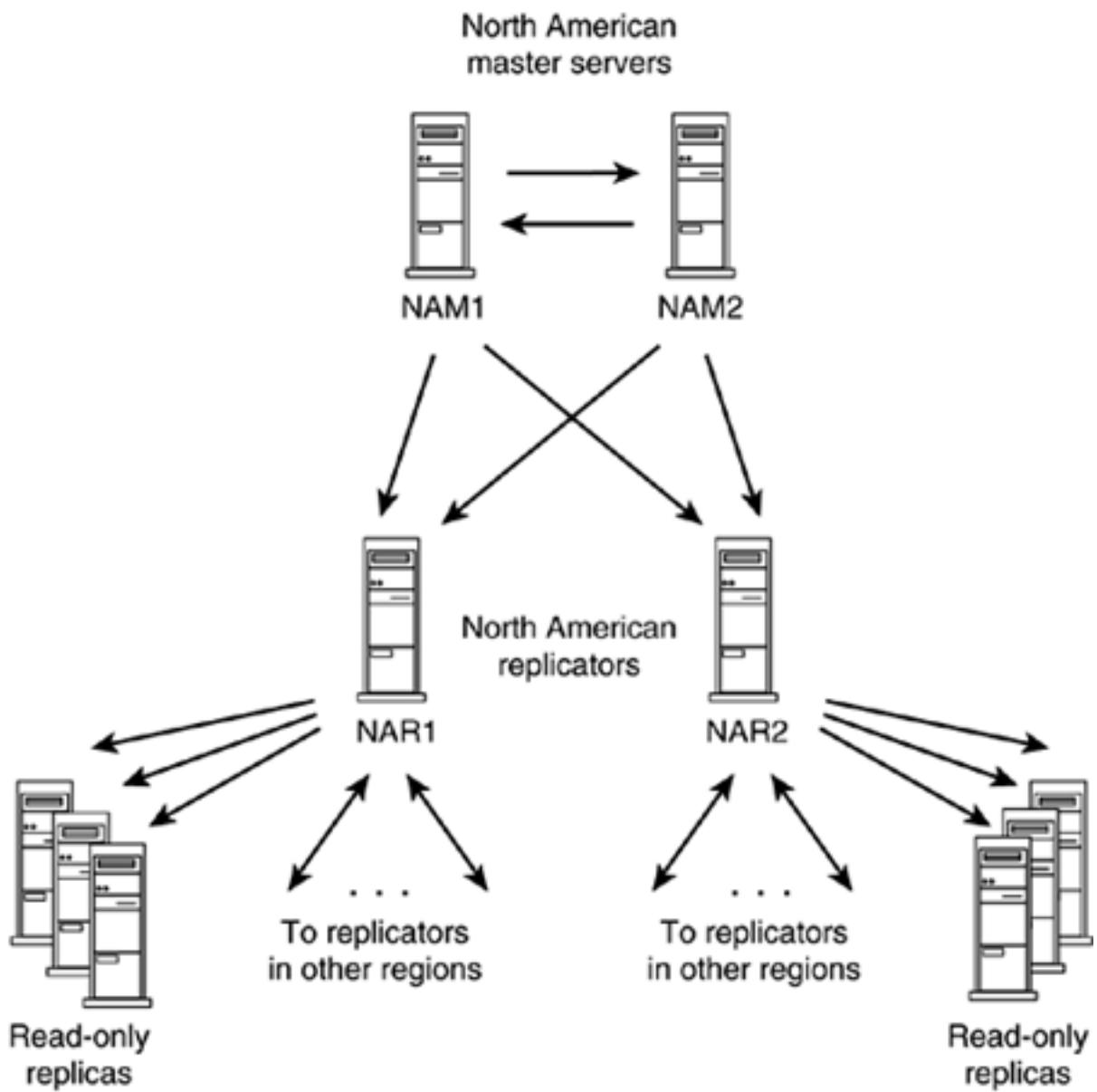


Replication

The main role of replication in the HugeCo design is to allow increased directory search and read capacity to be provided in an incremental way. To provide adequate capacity, all of HugeCo's directory data is copied to as many replicas as necessary. In addition to the paired read/write master servers described in the previous section, a read-only replica is located near each important directory-enabled application, such as a large mail server, a busy phone book service, or a Web-enabled workflow application.

HugeCo chose a two-tiered replication scheme in which each pair of master servers supplies two replicator servers, each of which supplies all the other replicas. [Figure 25.6](#) shows the details for this two-tiered replication schema for a sample region.

Figure 25.6. HugeCo's Two-Tiered Replication Scheme



The two-tiered scheme was selected to reduce the load on the master servers in an effort to maximize update performance. Aside from handling approximately half the update traffic for the entries it masters, each master server needs to supply only two replicator servers. Each of the replicator servers holds a complete copy of the HugeCo directory tree, which is kept up-to-date by the master servers.

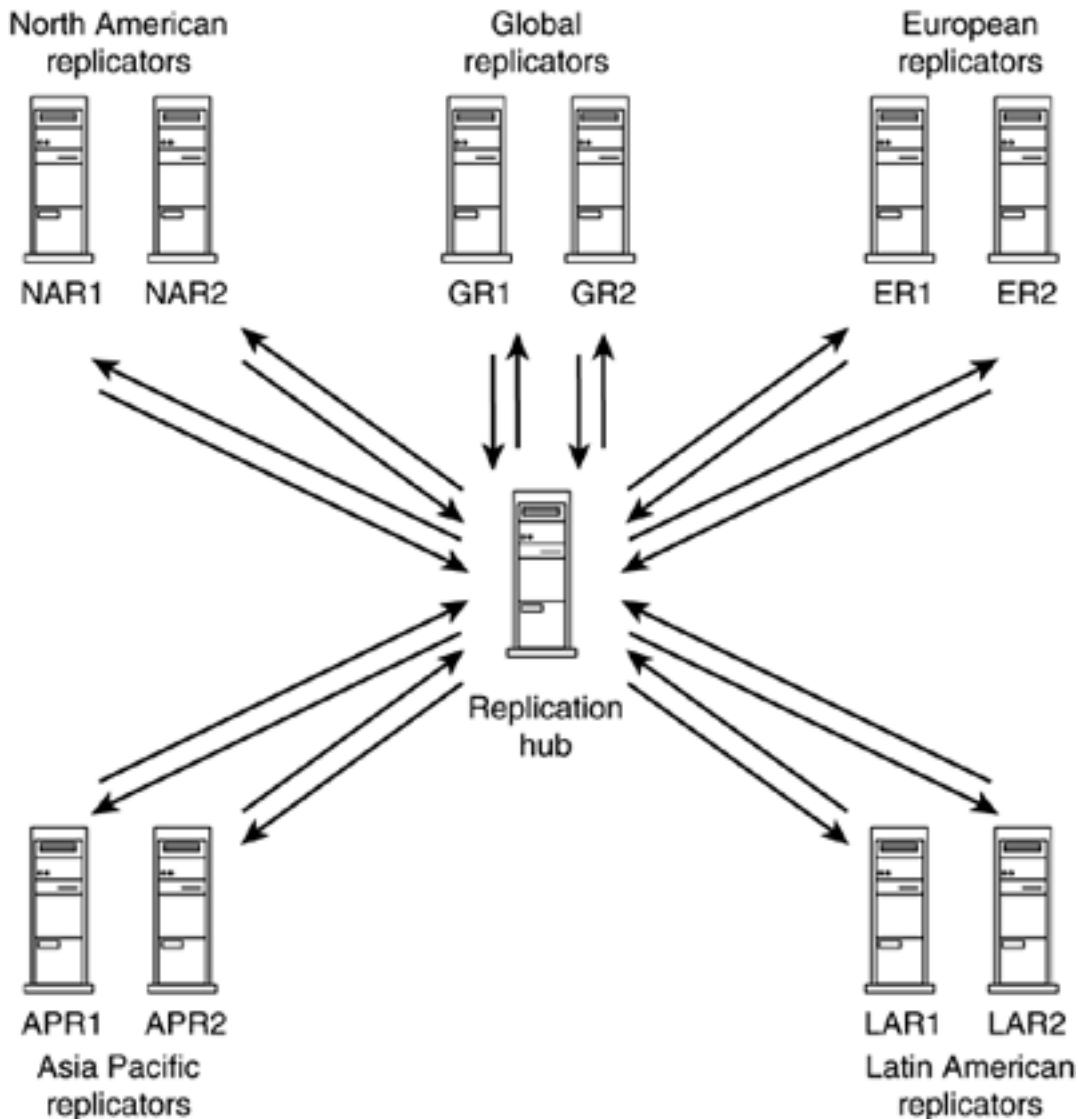
Because many directory server products require that the unit of replication be defined as an entire subtree, the topmost entry in the HugeCo directory namespace (the `dc=hugeco, dc=com` entry) cannot be automatically replicated. Instead, each server has its own copy of that entry, and access control rules are used to ensure that no changes are made to it.

20/20 Hindsight: Awash in Replication Agreements

As HugeCo moved beyond the directory pilot and rolled out its production service worldwide, it found that setting up the large number of server-to-server replication agreements was a large chore. Now that the servers are set up, maintenance of the replicated servers is easy, but there are still many replication connections to monitor. For example, each of HugeCo's 10 replicator servers must supply all the other replicators, with the exception of the replicator within its own region. That means that each of the 10 replicator servers must supply the 8 peers located in the other regions, resulting in a total of 80 replication agreements (one agreement is needed in each direction of replication).

To reduce the number of replication agreements needed, a new design that includes some central replication hubs is being considered. As shown in [Figure 25.7](#), a simple star topology can be used to replicate data across all regions while requiring only 20 agreements. To avoid creating a single point of failure, a second central replication hub will need to be introduced. Replication agreements between the pair of replicator servers within each region could be set up, and then an agreement between each regional replicator and one of the central replication hub servers could be established.

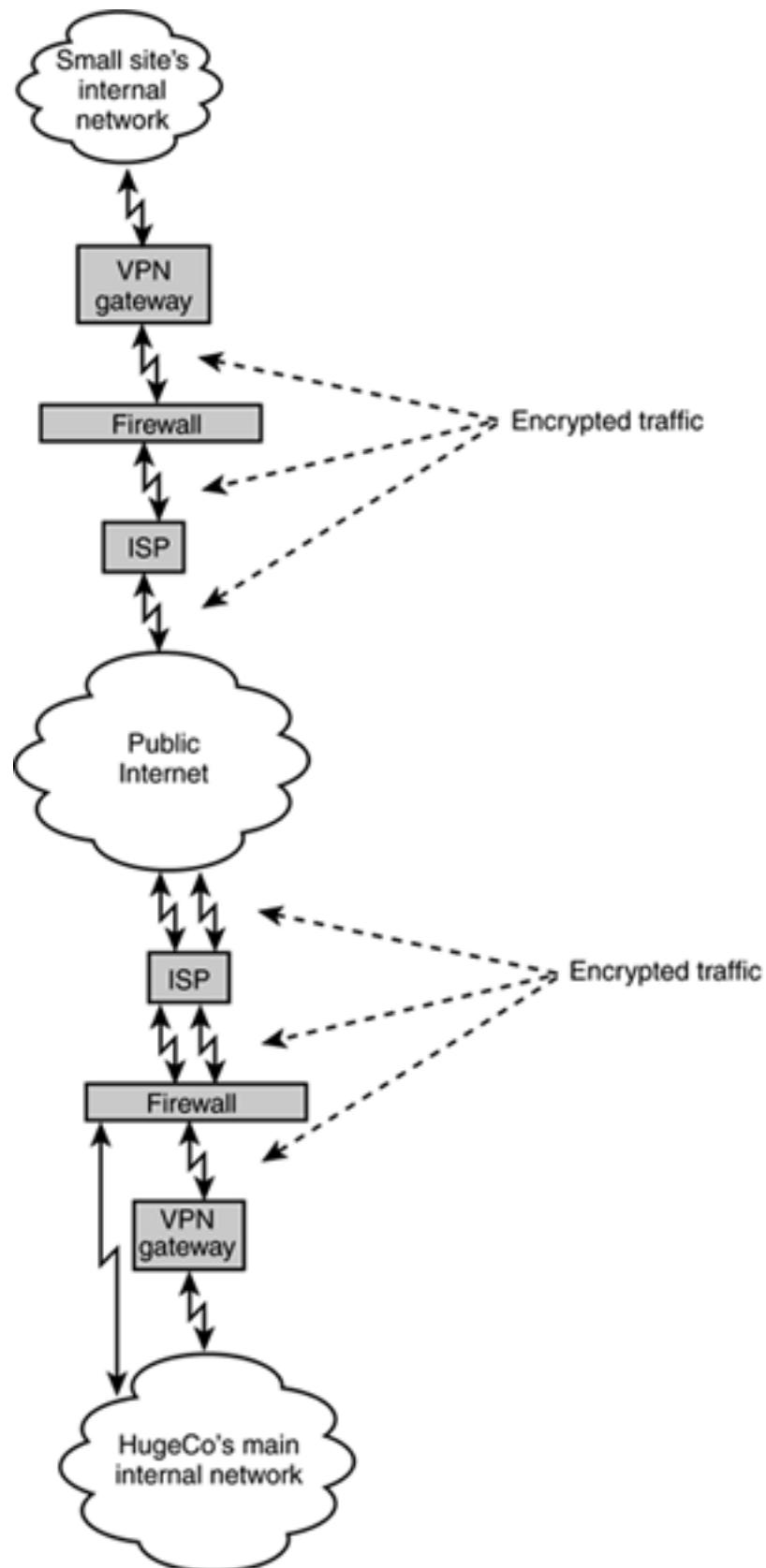
Figure 25.7. Future Replication Topology



Privacy and Security

HugeCo generally trusts its own employees, but it is wary of outside security threats. Because no directory data is currently published outside HugeCo's corporate firewall, the threat from outsiders is minimized. However, some small HugeCo offices are connected via IPsec-based VPNs, which originate at a local ISP, tunnel TCP/IP traffic across the Internet through an encrypted pipe, and enter HugeCo's corporate network through its Internet firewall (see [Figure 25.8](#)).

Figure 25.8. The VPN Used to Connect Small Sites



The VPN encryption technology is believed to be uncrackable. To be safe, however, directory access control was set up to disallow any changes to directory data from connections that come in through a VPN. A manual process (usually handled through e-mail) is used by employees at small sites to request updates to the directory data. These change requests are handled by directory services data administrators located in the IS department, who screen the requests carefully.

To protect data against naive or misguided employees who might try to download large lists of employees from the directory service, two precautions were taken. First, all server-to-server replication transfers are done over 128-bit encrypted Transport Layer Security (TLS) channels. Second, conservative size and time limits were specified for all directory servers to make it inconvenient to collect large numbers of entries (a process called *trawling*). Searches initiated by regular users are limited to at most 200 returned entries, and a maximum of 30 seconds is spent by a directory server on each search. Higher limits are given to directory administrators and applications such as the e-mail servers.

There is also a lot of concern at HugeCo about keeping employee data private, and the CIO believes strongly in personal privacy. To meet employees' privacy needs, the directory design team developed a conservative access control scheme that allows only an employee's manager to see the employee's home address and telephone number. In addition, the access control for groups was designed so that only the owners (maintainers) of a group can see the complete list of members.

20/20 Hindsight: Giving Individuals Greater Control over Access to Their Personal Information

After the HugeCo team rolled out its production directory service, it discovered that employees wanted finer control over the access granted to their personal information. The most common request was for home addresses and telephone numbers to be accessible to more than just a person's manager.

To meet legal requirements, and because not all employees want to share their home information, a small redesign is under way to allow each employee to choose whether his home information can be read by everyone. To provide this option, access control rules will be defined that allow anonymous access to the `homePhone` and `homePostalAddress` attributes if a Boolean attribute called `hugeCoHomeInfoIsPublic` within a person's directory entry is set to `TRUE`. The phone book front end is being altered to provide an **Allow Everyone to See My Home Information** check box that employees can toggle to change the value of the `hugeCoHomeInfoIsPublic` attribute, thereby granting access to other employees as they desire.

Directory Service Deployment

While the directory design was being completed and reviewed, HugeCo formed a directory deployment team. The team included all the people who participated in the design process, plus system administrators responsible for the actual rollout and for running the service on a day-to-day basis. An IS employee who had expertise in network monitoring and problem escalation procedures was also added to the team.

Product Choice

Before making a final choice of LDAP server software, HugeCo performed an extensive in-house evaluation. After talking to many directory server vendors, HugeCo narrowed its choice to three products: Netscape's Directory Server, Critical Path's CP Directory Server, and Novell's eDirectory. Evaluation copies of each of the three products were obtained, and each was subjected to a thorough hands-on evaluation that involved installing the products, configuring them with HugeCo's schema, setting up replication, and conducting performance and scalability testing.

In the end, the team selected the Netscape Directory Server product for the following reasons:

- Best performance and scalability, as observed during performance tests
- Support for TLS, as required by HugeCo's replication and security design
- Flexible, powerful access controls whose use did not significantly reduce performance
- Good support for international data
- Comprehensive management tools that included a Java GUI, command-line scriptable tools, and an HTML-based delegated administration tool
- Support by all of HugeCo's important directory-enabled applications

HugeCo also evaluated several LDAP software development kits (SDKs), including Netscape's C and Java SDKs, a few LDAP Perl modules found on the Internet, Microsoft's ADSI, and JavaSoft's JNDI. The team found that all these SDKs were functional but decided to focus on Netscape's SDKs and the PerLDAP Perl module for most of its own development projects. The team members recommended the Netscape SDKs primarily because they felt confident that these SDKs would work well with the Netscape and Sun server products already selected. Availability of source code for the SDKs was considered a bonus.

Piloting

An extensive directory service pilot was conducted to prove the directory design, become familiar with the directory software, and determine the level of effort required to roll out and maintain the production service. HugeCo's North America and Asia Pacific regions participated in the pilot, which was conducted over four months. During the pilot, the directory service was deployed in a limited number of physical sites within each region, and only one master and one replicator server were used for each portion of the directory data.

The directory-enabled applications used in the pilot included the following:

- Sun ONE Messaging Server to provide e-mail routing and delivery for end users.
- A simple workflow application to allow employees to request vacation time and other time off. This application was hosted by a Netscape Enterprise (Web) Server in conjunction with Netegrity SiteMinder and included an interface for employees (used to request time off) and an interface for managers (used to approve time-off)

requests). The application uses the `hugeCoEmployeeRole` and `manager` attributes within employee entries to route time-off requests appropriately and verifies that a specific manager is allowed to approve or deny an employee's request.

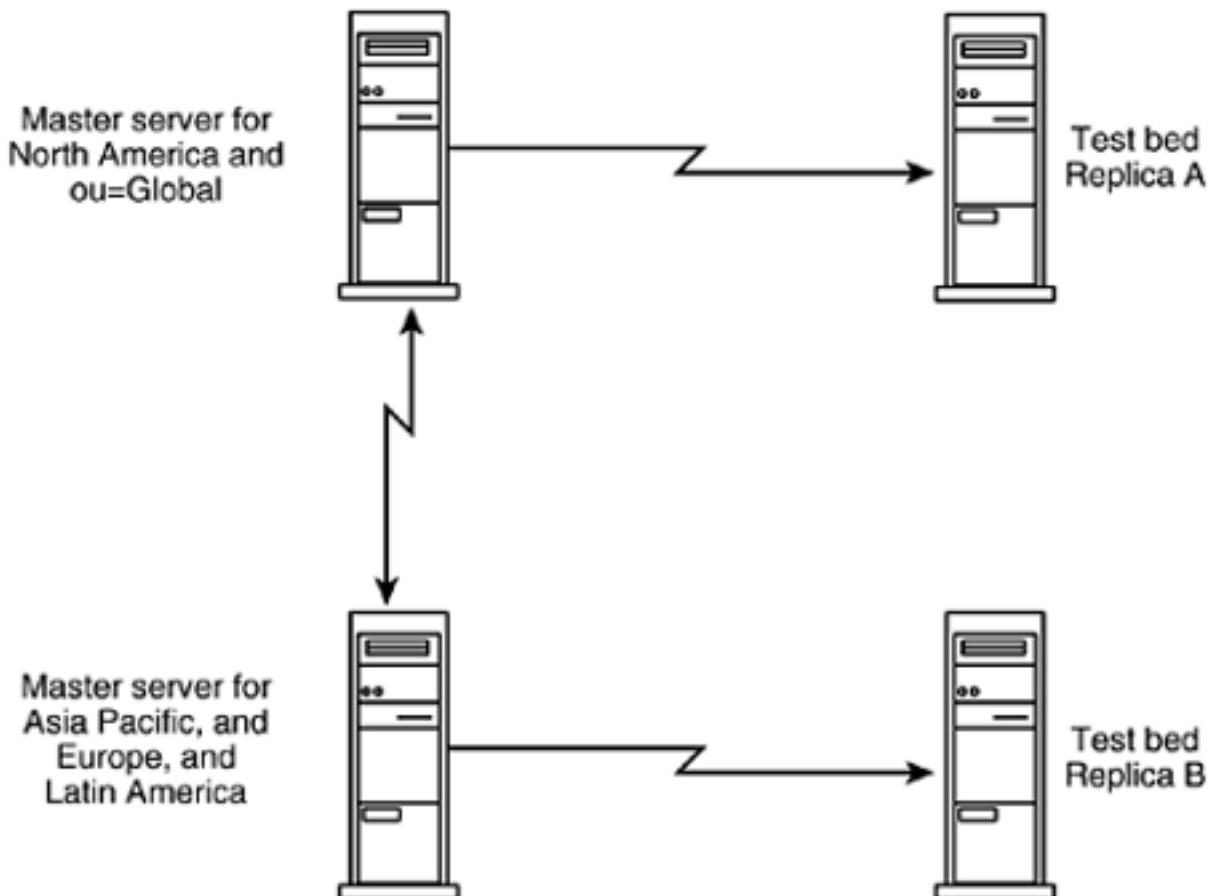
- An employee phone book to support anonymous directory lookups. This was created by the central IS programmers using PerLDAP.
- A directory administration application, based on the Netscape Delegated Administrator tool. This application supports directory administration at the global and departmental level, as well as employee self-service activities such as setting passwords or changing home telephone numbers. The directory administration application is accessible from the employee phone book via a hypertext link.

Apart from testing the directory-enabled applications, an important goal of the pilot project was to obtain feedback on the directory service from end users and system administrators. To collect feedback from end users, the directory phone book was modified halfway through the pilot to occasionally display a simple survey form before providing access to the phone book itself. Face-to-face and telephone interviews were conducted to collect feedback from system administrators of directory-enabled applications and the directory service.

The pilot showed that most of HugeCo's directory design choices were sound. One major redesign was done halfway through the pilot after the team experienced the pain of managing a replication topology that included many partitions. As discussed earlier in this case study, the directory namespace was redesigned to use a simpler structure based on regions rather than DNS subdomains.

After the pilot project was complete, most of the hardware used was incorporated into the production directory service. A few servers were reserved to form a test bed for future experiments with new applications, new directory server software, and directory design changes. [Figure 25.9](#) shows the test bed topology.

Figure 25.9. The HugeCo Directory Test Bed



Normally none of the servers in the test bed are connected to the production directory service, although sometimes they are temporarily incorporated into the production topology to prepare for software upgrades or obtain data for testing purposes. One limitation of the HugeCo test bed is that it does not match the replication topology used in the production service. As older machines become available, the HugeCo directory team plans to make them part of the test bed and improve it by adding replicator servers and by pairing up the master servers.

Analyzing and Reducing Costs

HugeCo tried to minimize the ongoing cost of its directory service by saving money in the following ways:

- All routine directory administrative tasks were automated, including nightly backups, service monitoring, creation of entries for new employees, and deletion of entries for terminated employees.
- Pilot hardware was reused to deploy the production service and form a directory service test bed.
- A small number of larger, more expensive server machines were used instead of many smaller machines. This cost-cutting measure followed the principle that personnel costs are more significant than hardware costs, and it fit well with HugeCo IS's general approach toward service deployment.

HugeCo has not conducted a thorough analysis of directory costs and has no immediate plans to do so.

Putting the Directory Service into Production

Because HugeCo's directory deployment involved many sites, servers, and applications, the production rollout was a complex undertaking. The key to success was to roll out the service in five phases:

Phase 1: Roll out directory servers and the phone book application in one region (North America).

Phase 2: Roll out directory servers and the phone book application in the remaining three regions using the same server configuration and enlisting the assistance of IS staff members who deployed the North American service.

Phase 3: Deploy the administrative application across the entire service. This phase was handled by the central IS staff, with testing performed by IS staff within each region.

Phase 4: Deploy directory-enabled e-mail services in each region. This phase was handled by the regional IS staff with help from some directory experts in the central IS organization.

Phase 5: Deploy other directory-enabled applications, including the Web-based workflow applications and the Netegrity SiteMinder access control product on which they depend.

In conjunction with the production rollout, training sessions were conducted within each region for IS system administrators and Help Desk staff. The IS communication group spread the word about the directory service by publishing a series of how-to articles in the

employee newsletter and through a "Do you know where your directory entry is?" poster campaign. Posters were placed in every HugeCo building to encourage employees to try the phone book application and to use the self-service feature to update their own directory entries. The poster campaign raised awareness of the new service and improved the accuracy and completeness of employee information in the directory.

Team LiB

◀ PREVIOUS

NEXT ▶

Directory Service Maintenance

Ongoing maintenance of HugeCo's large directory service requires a lot of attention from IS system administrators. This is especially true at the present time because the service is still evolving as new directory-enabled applications are being integrated. All basic maintenance is handled by automated procedures that are similar to those used for other systems that the IS organization manages. The following sections provide specific information on each aspect of directory maintenance within HugeCo's deployment.

Data Backups and Disaster Recovery

As discussed earlier in this chapter, there are two master servers for each portion of the HugeCo directory namespace. Once a month, one of the master servers is taken down, and the system is tested to ensure that everything can still function well with only one master.

The master servers are backed up to disk nightly via the directory server's "hot backup" feature and archived to tape via digital linear tape (DLT) drives. Twice a week, each region sends a set of backup tapes to another region for off-site storage. The backup procedures are largely automated and similar to those used for all the services that HugeCo's IS organization supports.

HugeCo outsources all its disaster recovery planning and services to IBM Business Continuity and Recovery Services, which maintains cold sites in each of HugeCo's four regions. So far HugeCo has not experienced a disaster that required use of the cold sites.

Maintaining Data

The IS organization spends a lot of time and money on data maintenance across all of HugeCo's systems. Corporate data is held in a variety of databases, and keeping the data up-to-date is largely a manual process. One goal of the directory service team was to increase the overall data maintenance burden as little as possible. The team managed to minimize maintenance demands by automating some processes and distributing data maintenance responsibilities.

To integrate with its PeopleSoft HR database, HugeCo contracted with America Online's Professional Services organization to create a directory synchronization tool. The synchronization tool runs once per hour to transfer changes made in the HR database to the directory service. Basic information about employees is synchronized, including name, contact information, ID number, and location. The synchronization tool takes care of creating new **hugeCoPerson** entries in the directory service when employees join HugeCo, and it disables user accounts by altering passwords after an employee leaves the company. The synchronization tool, written in Perl, operates on text extracts generated from the PeopleSoft database, and it uses the PerLDAP module to access the LDAP directory.

To distribute the data maintenance responsibilities, the HugeCo team defined the following categories of directory data managers:

- **Directory administrators**, who are granted complete access to all the data in the directory service
- **Departmental administrators**, who are granted nearly full access rights to the people and group entries for their department (but are not allowed to change any attributes managed by the HR database synchronization process)
- **Help Desk staff**, who are permitted to set passwords for all people entries

- **End users**, who are allowed to change home contact information, URLs, descriptions, and a few other fields within their own entries

For access control purposes, groups are maintained in the directory for each category of data administrators. The one exception is the end-user category: End users are identified by the absence of group membership. Access control rules were placed in the directory to give people in each category an appropriate level of access. Because departmental administrators and end users are allowed to manage some of their own information, the data management burden carried by the IS employees (directory administrators and Help Desk staff) is minimized.

20/20 Hindsight: Improving Data Quality

As an increasing number of HugeCo employees found out about the new directory service and began to examine their own data, the IS Help Desk started to receive reports of erroneous information. To understand the problem better and determine the cause, the central IS organization is developing an e-mail survey tool that will extract information from both the PeopleSoft HR database and the directory service. Surveys will be sent to a random sample of 5,000 employees in an effort to determine how widespread the data quality problems are. The results will be checked against directory audit logs to determine the source of the incorrect information, and the data gathered by the survey will be used to decide where to focus future data quality improvement efforts.

Monitoring

The overall HugeCo strategy for network monitoring revolves around HP OpenView, a commercial network management system (NMS). Each regional IS department runs an HP OpenView system that monitors the network and the applications located in that region. In addition, the central IS organization runs an HP OpenView system that monitors the global network and centrally managed applications such as the PeopleSoft system.

A combination of techniques was used to integrate the Netscape Directory Server software and important directory-enabled applications into the NMSs. First the Simple Network Management Protocol (SNMP) support built into the server software was used to provide basic service and performance monitoring. Then a set of Perl scripts was developed with the PerLDAP module to probe all the critical directory servers from several locations on HugeCo's network. Finally, indirect monitoring of the directory service was started through extensive observation of critical directory-enabled applications, including the e-mail servers, the PeopleSoft synchronization process, the phone book servers, the Netegrity SiteMinder servers, and the Web servers that support critical applications. As much as possible, probes mimic the operations that end users and applications frequently perform.

20/20 Hindsight: The Value of Indirect Probes

About a month after the HugeCo directory service was first rolled out worldwide, the IS e-mail team received a complaint from one of the executive vice presidents in the Latin America region. E-mail was being delayed for up to 30 minutes before reaching all the intended recipients. This was puzzling because most messages were routinely delivered by the network of Sun ONE message transfer agent (MTA) servers within 5 minutes.

After an afternoon of investigation, the e-mail administrators discovered that all the delayed messages had been sent to a dynamic group (one in which membership is determined by a search of the directory). They quickly brought some directory experts over to look at the problem. In the end, the root cause of the problem was traced to a missing index in the configuration of the directory replica servers used by the MTAs. Although easily corrected, the problem had gone unnoticed for almost a month (much to the chagrin of the IS staff).

This incident prompted the IS employees to design and implement a series of indirect directory probes that closely emulate the behavior of important applications such as the messaging servers. By proactively monitoring the performance of the system as experienced by end users, the HugeCo IS staff hopes to detect problems earlier in the future.

When a problem is detected by HugeCo's OpenView monitoring system, the following automated notification methods are used to bring the problem to the attention of the appropriate system administrator:

- Text pager messages are sent when an urgent system outage is detected.
- E-mail messages are used to send weekly directory activity summaries and to notify administrators immediately about problems such as reduced performance of a directory application.
- IS staff and end users can access a continuously updated Web page that lists information about all known outages.

Overall, the directory service and associated applications have proven to be reliable. So far, there has been no need to automate such actions as restarting failed directory server processes or machines.

Troubleshooting

HugeCo's IS organization maintains a well-documented set of escalation procedures stating that members of the IS staff with increasing seniority will be called in over time to address critical problems. Directory-specific procedures were developed during the directory pilot deployment and refined over time to ensure that problems are addressed quickly by the right people.

Leveraging the Directory Service

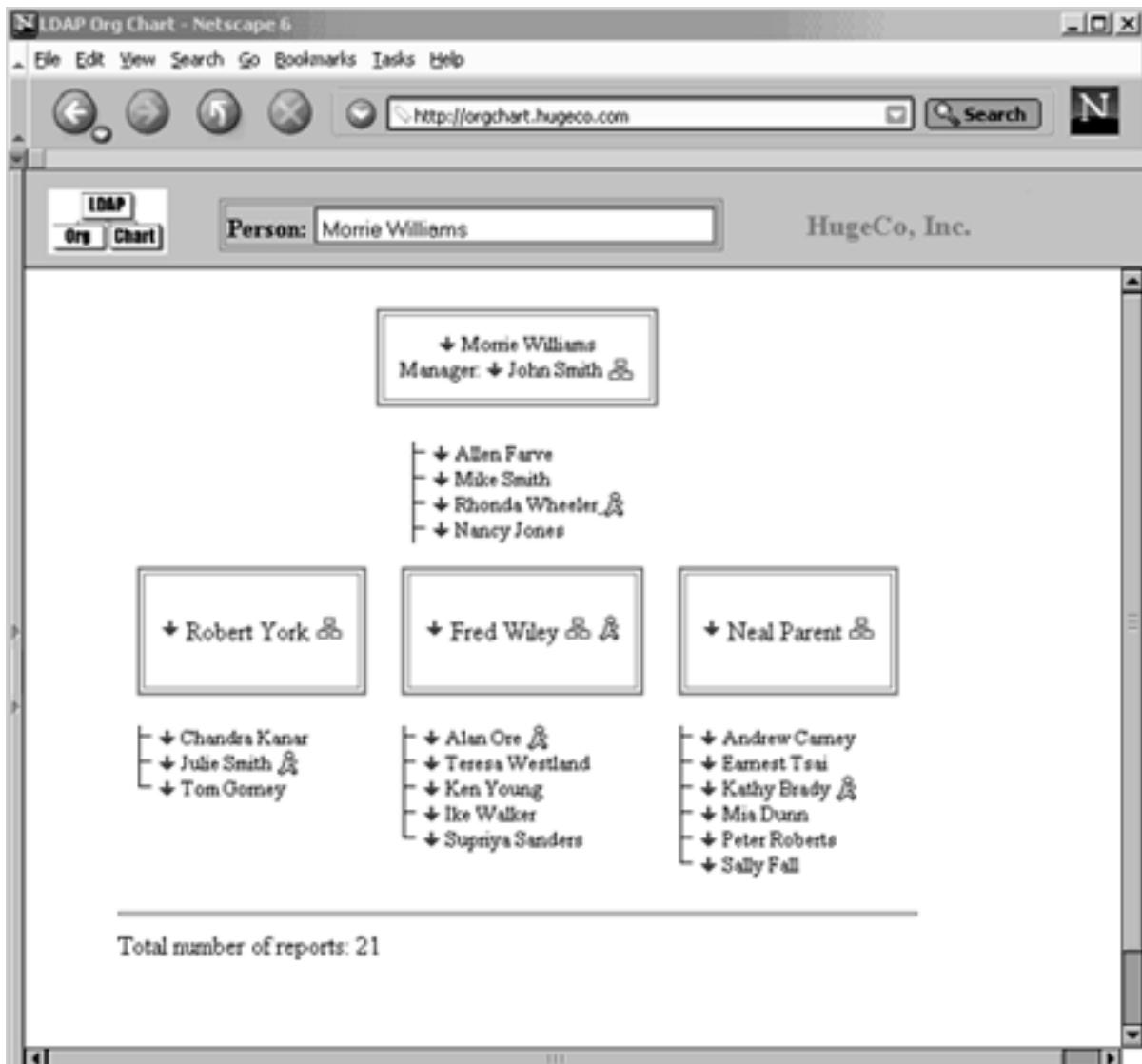
Now that the LDAP directory service has been successfully deployed worldwide, HugeCo is finding new and interesting ways to leverage the directory. In this section we describe applications that are in the process of being developed and integrated with HugeCo's directory, and we look at how things have changed at HugeCo since the directory was deployed.

Applications

The central IS organization, the regional IS departments, and several independent departments within HugeCo are all leveraging the directory service to enhance existing applications and develop new ones. Efforts under way include the following:

- Well-informed end users quickly discovered that many of their favorite e-mail clients have LDAP search features built in. The IS departments picked up on this discovery and a series of articles was published in the employee newsletter that explained how to configure the Eudora, Outlook, and Netscape Messenger e-mail clients to access the corporate LDAP directory service. Before the introduction of LDAP, it was impossible for clients from different vendors to share one directory.
- Pleased with the success of the employee time-off request application, the HR department is working with IS programmers to create a variety of Web-based, directory-enabled workflow applications. The goal is to automate common employee processes that are used when an employee joins or leaves the company, enrolls in employee health benefit plans, and more. These applications are being developed with HTML, JavaScript, and Java, and most are leveraging Netegrity SiteMinder for application-level access control.
- The North American product engineering group has embarked on an ambitious project to tie its existing database of engineering drawings into the directory. The integrated system will make it easier to locate the persons responsible for an existing design. Eventually the directory service will be used to build a workflow system to automate the design approval process.
- A pilot project under way within the IS and HR departments will require public key certificates for authentication to e-mail servers and the time-off request workflow application. The directory service is being used to manage personal certificates, perform revocation checking, and streamline the process of issuing and renewing certificates.
- The central IS department is developing an enhanced employee phone book application that will provide access to information about people, conference rooms, office equipment (such as computers and fax machines), and building locations. The new phone book is an extension of the existing one and is being written in Perl. Location information pulled from the directory service will be integrated with building maps. Directory information will also be used to create organizational charts (org charts) from the employee data. [Figure 25.10](#) shows a prototype of the org chart module.

Figure 25.10. Prototype of the HugeCo Org Chart Module



Without exception, it is now easier and much more practical to create these applications than it was before the HugeCo directory service existed.

Directory Deployment Impact

After the directory was deployed and people started to use it, the HugeCo directory team observed that the service rapidly became indispensable to many employees. Some of the team's observations are reported in this section.

Richer services are now available to end users. It is much easier to find timely information about other employees, so communication among employees has increased. Paper lists of phone numbers are a thing of the past, now that many employees have come to rely on the directory-enabled phone book application. Employees can more easily update information about themselves, and new automated services such as benefit request systems are coming online.

A richer set of functionality is available to application developers. Application writers can easily access and update information about a variety of important HugeCo resources without incurring the burden of managing their own directory or database. Simple authentication and attribute-based role information can be used to make applications more secure. The directory-based Netegrity SiteMinder deployment is easily leveraged by developers of Web-based applications.

The HR and IS organizations are using the directory service to build closer relationships with

their customers and to place more control (and more of the work) in employees' hands. Timeliness and manageability of important information has improved dramatically.

Finally, the HugeCo directory design and deployment team has greatly increased its knowledge of directory service design and is now well positioned to tackle additional directory projects. First up will be the design and deployment of an extranet directory that will store information about HugeCo's dealer network. This new directory service, which is described in [Chapter 26](#), will allow the deployment of a new class of secure, Web-based information delivery applications. The extranet directory design team will draw on lessons learned during the internal HugeCo directory deployment.

Team LiB

◀ PREVIOUS

NEXT ▶

Summary and Lessons Learned

Now let's step back and take a critical look at the HugeCo directory design and deployment process. HugeCo made a large investment to ensure that its directory service would be a success—and it is. As always, we can learn from the company's experience, successes, and failures.

Although HugeCo's directory design team clearly did its homework, it still occasionally found problems with the design. In some cases a redesign was required before deployment of the production service. Some problems are yet to be resolved, and more redesign and redeployment activities are on the horizon. Interesting design problems that HugeCo's team encountered include the following:

- The original namespace design proved too cumbersome and had to be redesigned during the directory pilot.
- Setting up and managing many replication agreements was burdensome, and alternative designs are being examined to reduce the number of agreements required.
- Individual employees made their voices heard in asking for more control over who can access their personal information.
- Several data quality problems must be investigated and remedied.
- The initial plan for monitoring the directory service did not include any indirect probing. This facet of monitoring was added only after an embarrassing directory configuration error was unearthed when a vice president complained about poor e-mail service.

As more applications are integrated with the directory service, new challenges will no doubt surface.

Overall, HugeCo's directory design and deployment process was well conceived and well executed. By conducting a serious pilot, the team members discovered some critical problems before they deployed the entire service. They were not afraid to revisit design decisions to improve the service. And although deploying a directory service for their large organization was a big task, the service is now being leveraged in some interesting and unexpected ways.

Further Reading

AOL Strategic Business Solutions Professional Services. Information is available on the World Wide Web at <http://enterprise.netscape.com/custsvcs>.

IBM Business Continuity and Recovery Systems. Information is available on the World Wide Web at <http://www-1.ibm.com/services/continuity/recover1.nsf/documents/home>.

Netscape Directory Products from AOL Strategic Business Solutions. Information is available on the World Wide Web at <http://enterprise.netscape.com/products>.

Netegrity SiteMinder. Information is available on the World Wide Web at <http://www.netegrity.com/products>.

PerLDAP, an object-oriented LDAP Perl module. Available on the World Wide Web at <http://mozilla.org/directory/perldap.html>.

Sun ONE Messaging Server. Information is available on the World Wide Web at http://wwws.sun.com/software/products/messaging_srvr/home_messaging.html.

Looking Ahead

In [Chapter 26](#), Case Study: An Enterprise with an Extranet, we will continue our parade of LDAP directory deployment case studies with an example of an extranet directory service.

Chapter 26. Case Study: An Enterprise with an Extranet

- Overview of the Organization
- Directory Drivers
- Directory Service Design
- Directory Service Deployment
- Directory Service Maintenance
- Leveraging the Directory Service
- Summary and Lessons Learned
- Looking Ahead

In this chapter we examine how a hypothetical company might extend its intranet-based services to its external trading partners. Such an arrangement, called an *extranet*, is an effective way to leverage existing systems and expertise.

In such a deployment, a directory service is usually already designed and deployed. To expand its service, however, the directory must be extended beyond the organization's internal network (and outside any firewalls). This constraint creates additional requirements, especially in the area of security and access control, that might necessitate a reassessment of the basic directory design.

The same fictional company that was introduced in [Chapter 25](#), HugeCo, is our model in this chapter. In this case study HugeCo extends its existing directory service to include services for its trading partners.

Overview of the Organization

HugeCo is a supplier of a wide range of custom-built home renovation products, from cabinetry and replacement windows to completely prefabricated homes. Its Home Renovation Products (HRP) division produces cabinets, counters, windows, and other products built to customer specifications and shipped to local home improvement retailers and contractors in the United States and Canada.

Instead of maintaining a direct sales force, HugeCo partners with home improvement retailers and contractors, who become certified retailers of HugeCo products. Certified retailers place orders for custom-built items, which are manufactured at HugeCo's main factory and shipped to the retailer or directly to the customer site. Retailers can become certified on any of several different product lines, and they may sell only those products for which they are authorized.

HugeCo currently has about 800 authorized retailers in the United States and Canada. To support each retailer, each month HugeCo ships them updated product literature and price lists. Because the ordering process is quite complex and varies from product to product, the literature includes worksheets that describe the options available for each product and any measurements, which must be supplied before the manufacturing process can begin.

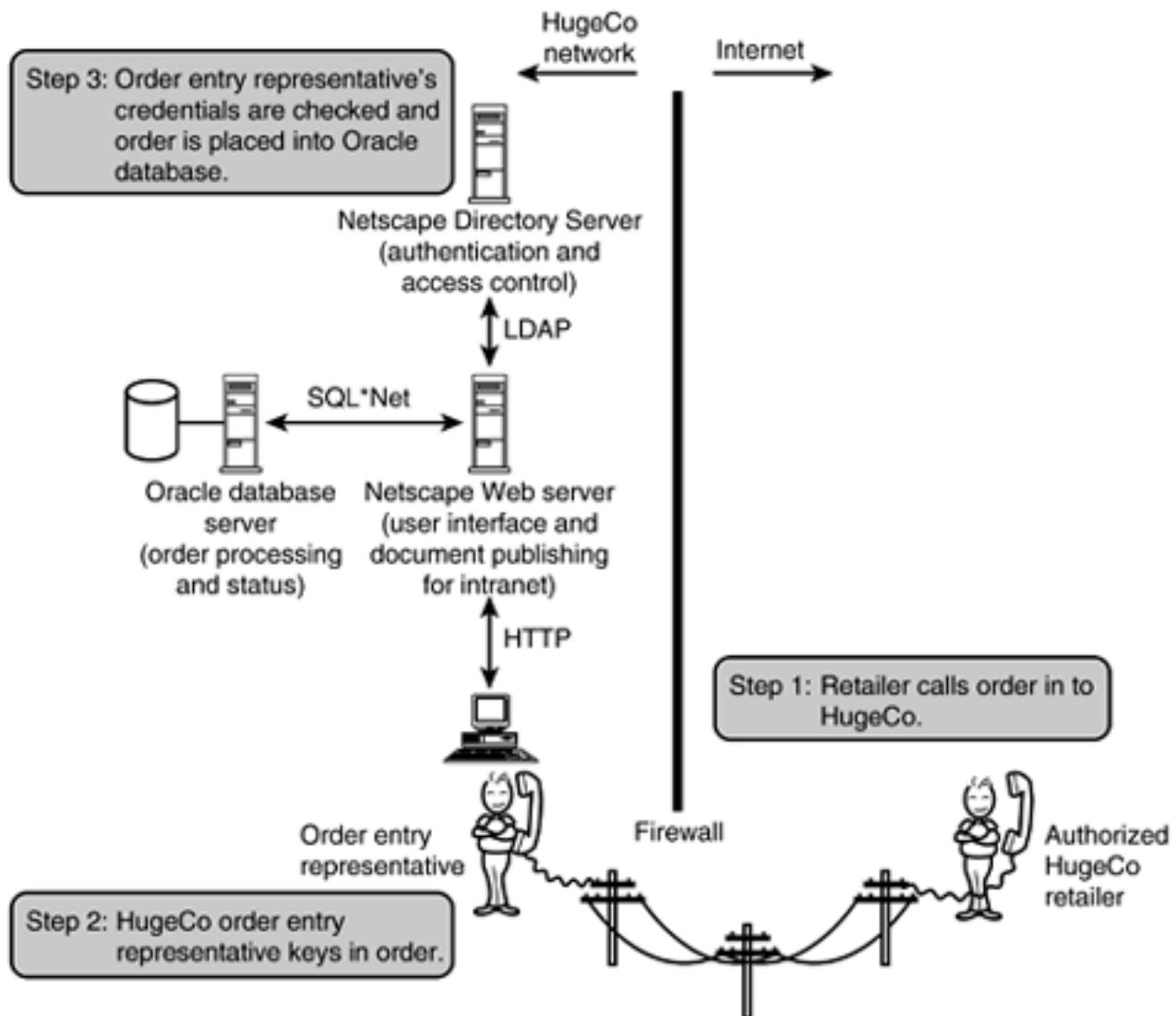
An early adopter of database technology, HugeCo uses a large Oracle installation to track the progress of orders through the system, from initial order to final shipment. Although the order tracking system had provided large gains in productivity at lower cost compared to the older system, its user interface was somewhat lacking. With the advent of World Wide Web technology, HugeCo's IS staff developed a Web-based interface for the system, making it easier to use, and therefore lowering training costs.

Spurred on by the initial success of the order management system, HugeCo established an e-mail link between the Manufacturing staff and the Sales staff. Over this link the craftsman responsible for building a custom set of cabinets, for example, can send electronic mail to the sales representative who originally took the order to request clarification on a particular aspect of the work order.

Recently, HugeCo management started a new project: a corporate extranet that aimed to extend the benefits of HugeCo's intranet to its authorized retailers. The primary motivations behind this extranet deployment were to improve service to retailers and customers and to lower costs. This case study describes the planning and execution of the directory component of this project.

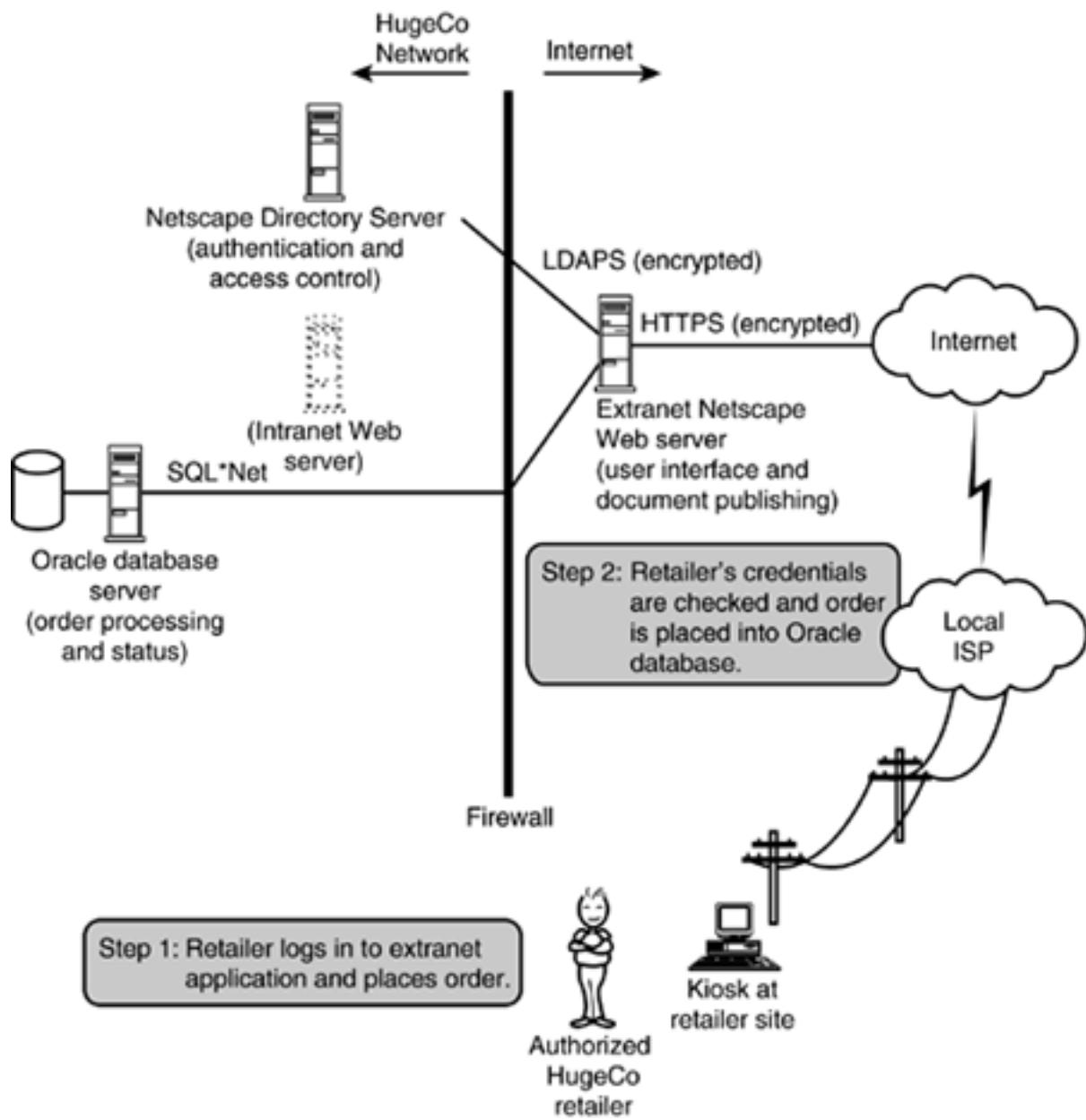
The way that authorized retailers previously placed orders and learned about status was inefficient and error-prone: The telephone was the medium for all interactions. The original order was called in to a data entry operator, who keyed the information into the order entry application. This transcription step was a source of errors. In addition, if a retailer needed to check on the status of an existing order, a call had to be placed to the central office during business hours. This order management system architecture is shown in [Figure 26.1](#). As the figure shows, the Web server provides an easy-to-use front end to the order entry and tracking database, which is hosted by an Oracle database server.

Figure 26.1. The Old Order Entry System Architecture



A much more attractive alternative would be to allow employees at authorized retailers to place their own orders directly into the Web-based order system, bypassing the data entry operators and improving accuracy. Such a system would also allow retailers to check order status directly 24 hours a day, 7 days a week, resulting in better customer service. Such a system is depicted in [Figure 26.2](#). Note that there is a much closer coupling between the retailer and the craftsperson in this system. Such a system should significantly improve responsiveness to retailer needs and higher customer satisfaction.

Figure 26.2. The New Order Entry System Architecture



New product information and ordering procedures were distributed along with regular product literature updates shipped to retailers every eight weeks. Because new product notices were distributed so infrequently, a long time could pass between the time that a new product was available and the time that orders for it could be placed.

An alternative approach was to publish notices about new products and procedures to the extranet. When a new product is available or a new procedure is released, each retail employee can be informed via items on a personalized start page. Information about products that the retailer is authorized to sell can be highlighted, and information about products not authorized can be hidden. New product literature and price sheets can be published to the extranet Web server, making the new product immediately available for ordering.

Directory Drivers

In planning for these extranet applications, having several drivers led the designers to conclude that a directory service was essential to the deployment. The following requirements led them to this conclusion:

- **Extranet applications must be manageable through delegation.** Because there are over 800 retailers, each of which has several employees authorized to sell HugeCo products, there are expected to be at least several thousand user entries in the database at any time. To complicate the management of this user information further, each retailer is an independent business with its own employees. Clearly, to make the management of user information possible, the creation, deletion, and maintenance of user account information must be delegated to the individual retailers.
- **User information must be shared among multiple applications.** Several different extranet applications (order entry, personalized product notice generation, and so on) would constitute the extranet suite, each needing to authenticate users and store customization information. Because a directory provides a standard method for storing and accessing this information over the network, it would make sense to place shared user information in a directory.

Directory Service Design

To support its new extranet applications, the HugeCo corporate directory design had to be revisited and altered. HugeCo's IS staff began the redesign process by analyzing the needs of the new extranet applications.

Needs

Several needs drove the direction of the directory design for the HugeCo HRP extranet:

- The applications demand a flexible access control model that supports delegation of user account maintenance. Conceptually, two roles would be defined for the applications: manager and employee. One or more user administrators would be defined per authorized retailer, and those persons would be responsible for creating, maintaining, and deleting entries in the directory for all the employees allowed to use the HugeCo extranet applications. The directory must support this capability.
- The extranet applications were anticipated to have roughly the same performance requirements as the equivalent intranet applications. For example, the extranet Web servers that support the order tracking system would place the same load on the directory as the existing intranet servers did.
- The directory must be highly available and deliver a high level of service. This means that the directory needed to be replicated for fault tolerance.
- The directory must support configuration of customized schemas to support the specialized schema requirements of these extranet applications.

Data

Because the extranet would be a set of new applications that leveraged an existing directory installation, HugeCo needed to review its existing data policy statement to see if it was valid for the new extranet applications. HugeCo's existing data policy stated the following:

- Data that is shared by more than one application should be stored in the directory service.
- The authoritative source for each data element stored in the directory must be defined.
- Data that is extremely sensitive or private should not be stored in the directory service unless there is an operational reason to do so. This kind of data includes payroll and employee benefit information.
- All legal requirements, including country and regional privacy laws in effect where HugeCo operates, must be followed.

After review, it was apparent that this data policy statement was suitable for the new extranet, and HugeCo staff began to define the new data elements needed to support these applications, to determine if the data was already available in the directory or other HugeCo databases, and to identify the authoritative data sources for each item.

Building on the concept of roles, the development team defined two new roles. The `hugeCoHrpRetailEmployee` role represents an employee at a retailer authorized to sell HugeCo's home renovation products, and `hugeCoHrpRetailManager` represents a manager at an authorized retailer.

Newly identified data elements necessary to support the extranet applications are discussed in the following section.

Schema

HugeCo chose to subclass standard object classes to represent the new objects that would be stored in the directory. Two new object classes were defined: `hugeCoHrpRetailer`, which represents an authorized retail outlet; and `hugeCoHrpRetailEmployee`, which describes an employee of a HugeCo home products retail outlet. The `hugeCoHrpRetailer` object class is defined in [Listing 26.1](#).

Listing 26.1 The `hugeCoHrpRetailer` Object Class

```
objectclass hugeCoHrpRetailer  
superior top  
oid 1.2.3.4.5.6.8  
required  
hugeCoHrpRetailerID  
allowed  
hugeCoHrpProductAuthorized
```

The `hugeCoHrpRetailer` object class is designed to be mixed in with the standard `organization` object class, and it augments the attributes from that object class with two additional attributes. The `hugeCoHrpRetailerID` attribute, used to name the entry, contains the unique authorized retailer ID assigned to that retailer. The `hugeCoHrpProductAuthorized` attribute contains a value for each HugeCo home renovation product line that the retailer is authorized to sell.

A sample `hugeCoHrpRetailer` entry might look like the one shown in [Listing 26.2](#). This entry describes an authorized retailer of HugeCo products. Located in Tucson, Arizona, the retailer is authorized to sell four different HugeCo product lines.

Listing 26.2 A Sample `hugeCoHrpRetailer` Entry

```
dn: hugeCoHrpRetailerID=882-501, ou=hugeCoHrpRetailers, ou=Extranet,  
dc=HugeCo, dc=com  
objectclass: top  
objectclass: organization  
objectclass: hugeCoHrpRetailer  
hugeCoHrpRetailerID: 882-501  
o: Jensen Home Improvement Products, Inc.
```

```
postalAddress: 123 Anystreet $ Tucson, AZ $ USA 94432
st: AZ
postalCode: 94432
telephoneNumber: +1 520 555 0499
facsimileTelephoneNumber: +1 520 555 3882
hugeCoHrpProductAuthorized: A733
hugeCoHrpProductAuthorized: J812
hugeCoHrpProductAuthorized: J813
hugeCoHrpProductAuthorized: J814
```

The `hugeCoHrpRetailEmployee` object class is defined in [Listing 26.3](#). This object class is designed to be mixed in with the standard `inetOrgPerson` object class, and it augments the attributes from that object class with four additional attributes:

1. **hugeCoHrpRetailerID**. Contains the retailer ID of the authorized HugeCo dealer that employs this person. This attribute is required; it makes no sense if a user has the `hugeCoHrpRetailEmployee` object class but is not associated with a retailer.
2. **hugeCoHrpLastLogin**. Contains the date and time that the employee last logged in to the system. The initial page shown to each employee contains a list of new product notices, promotions, and alerts, and the `hugeCoHrpLastLogin` attribute allows the Web server that generates the start page to show only those notices posted since the last time the employee logged in.
3. **hugeCoHrpEmployeeExpireDate**. Contains the expiration date for this entry (which is removed on or after that date).
4. **hugeCoEmployeeRole**. Contains the value `hugeCoHrpRetailEmployee` if the entry describes an employee at a retailer, or the value `hugeCoHrpRetailManager` if the entry describes a manager at a retailer.

Listing 26.3 The `hugeCoHrpRetailEmployee` Object Class

```
objectclass hugeCoHrpRetailEmployee
superior inetOrgPerson
oid 1.2.3.4.5.6.9
required
hugeCoHrpRetailerID
allowed
```

```
hugeCoHrpLastLogin  
hugeCoHrpEmployeeExpireDate  
hugeCoEmployeeRole
```

A sample `hugeCoHrpRetailEmployee` entry might look like the one shown in [Listing 26.4](#). This entry contains the attributes you would expect to see in a typical `inetOrgPerson` object, such as `cn` (common name), `sn` (surname), and `uid` (user identifier). However, this entry can also contain two additional attributes because it is of the `hugeCoHrpRetailEmployee` object class. The `hugeCoHrpRetailerID` attribute shows that this person works for retailer number 882-501, the `hugeCoHrpLastLogin` attribute shows the time that the employee last accessed the system, and the `hugeCoHrpEmployeeExpireDate` attribute gives the date on which the employee record will expire if it is not renewed by the manager. More information on the employee expiration policy is given later in this chapter, in the section titled Directory Service Maintenance.

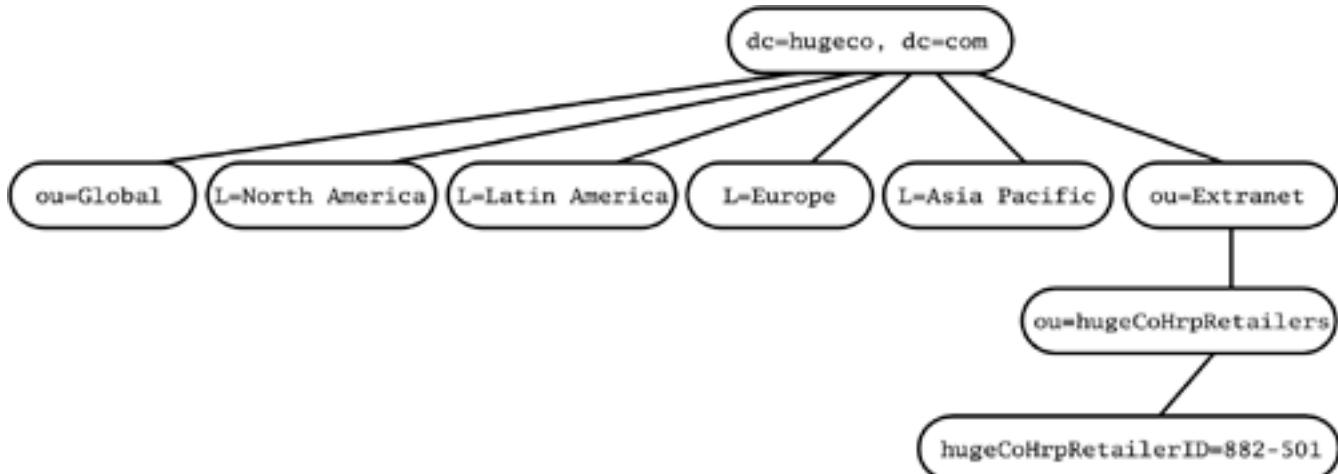
Listing 26.4 A Sample `hugeCoHrpRetailEmployee` Entry

```
dn: uid=hreming, hugeCoHrpRetailerID=882-501, ou=hugeCoHrpRetailers,  
ou=Extranet, dc=HugeCo, dc=com  
  
objectclass: top  
  
objectclass: person  
  
objectclass: organizationalPerson  
  
objectclass: inetOrgPerson  
  
objectclass: hugeCoHrpRetailEmployee  
  
cn: Harold Remington  
  
sn: Remington  
  
uid: hreming  
  
...  
  
(other inetOrgPerson attributes)  
  
...  
  
hugeCoHrpRetailerID: 882-501  
  
hugeCoHrpLastLogin: 200108300061302Z  
  
hugeCoHrpEmployeeExpireDate: 20030127  
  
hugeCoEmployeeRole: hugeCoHrpRetailManager
```

Namespace

To satisfy the delegation requirements of the extranet application framework, HugeCo decided to place the information about authorized retailers and their employees in a separate subtree. The complete HugeCo directory namespace design is shown in [Figure 26.3.](#)

Figure 26.3. HugeCo's Directory Namespace Design, Including Extranet Data



Placing all the information about authorized retailers into the `ou=hugeCoHrpRetailers`, `ou=Extranet,dc=hugeco,dc=com` subtree allows separate access control policies to apply to this information. It also allows this information to be replicated selectively across the firewall without requiring that all other corporate data in the directory also be made available outside the firewall.

20/20 Hindsight: Namespace Design and Fan-out

The initial design for HugeCo's Home Renovation Products extranet directory namespace called for a single container named `ou=Retailers,dc=hugeco,dc=com` that would contain all the HRP division retailers. Soon after completing the initial design, the extranet application developers realized that, if successful, their extranet application would encourage the development of other extranet applications within other HugeCo divisions. The original namespace design was clearly inadequate; placing all retailers underneath a single subtree would present problems as soon as the thousands of retailers of other HugeCo products were placed there.

To address this issue, the HRP extranet developers opted to create a container named `ou=Extranet` that could be subdivided as more extranet applications were developed. Within the `ou=Extranet` container, additional containers could be created to accommodate each new extranet application.

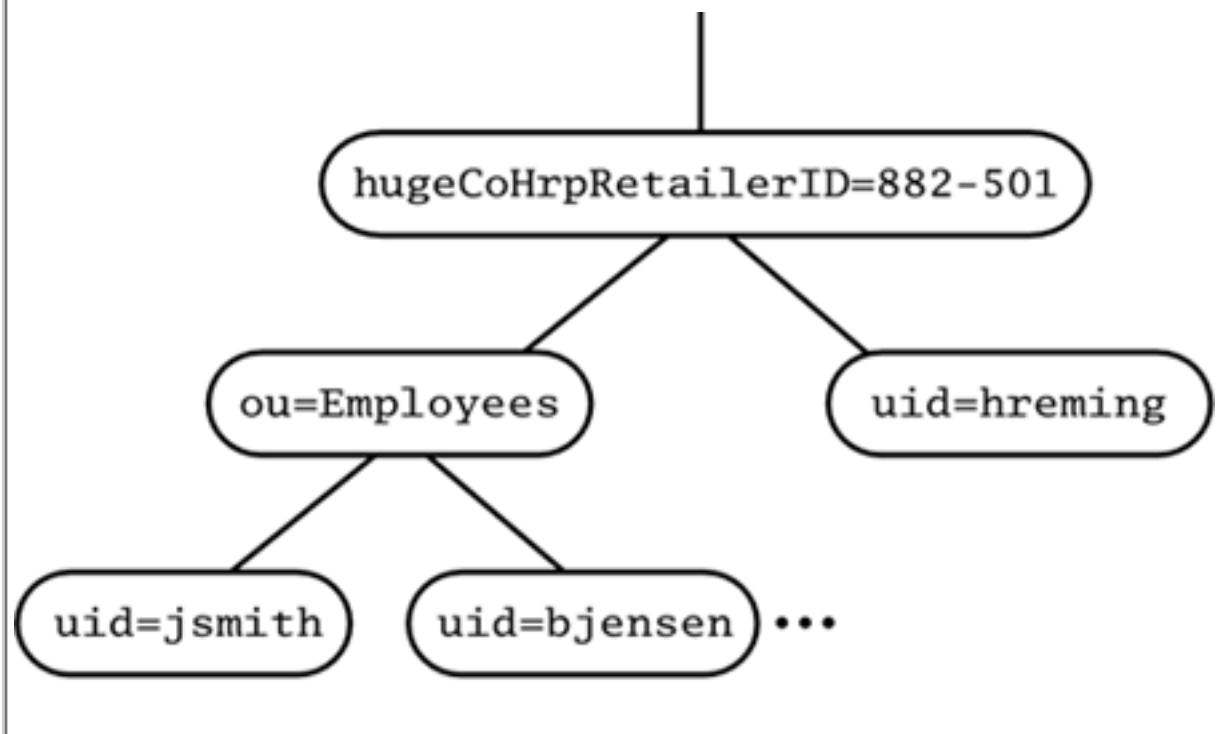
Underneath the retailers subtree is one tree for each retailer. The entries at the top of each tree are of the `hugeCoHrpRetailer` object class and contain information about the actual retailer (see the Schema section earlier in this chapter for more information about the `hugeCoHrpRetailer` object class). Within each retailer subtree is an entry for the primary contact person (usually the department manager) and a container named `ou=Employees`. This container holds information about additional employees, if any, who are authorized to place and

track orders.

The intent of placing data about each retailer in its own subtree is to make it easy to delegate administration of that information. For example, the manager at any particular retailer could be delegated the ability to update information about the retailer and create new employee entries, but only within his or her particular retailer subtree. Early in the design process, it was unclear whether it was a good idea to allow such delegation, but the designers decided to construct a directory namespace that could support delegation if needed. Delegation can be enabled by placement of the appropriate access control directives in the `hugeCoHrpRetailer` entries.

You may be wondering why HugeCo decided to add the `hugeCoHrpRetailerID` attribute to entries representing employees. On the surface it seems unnecessary because all the employees of a given retailer are held underneath the entry describing the retailer (see [Figure 26.4](#)). However, if you want to find all the employees of a given retailer, this construction allows you to perform a subtree search rooted at the retailer entry with a filter of (`objectclass=hugeCoHrpEmployee`).

Figure 26.4. A Portion of HugeCo's Directory Representing a Single Retailer and Its Employees



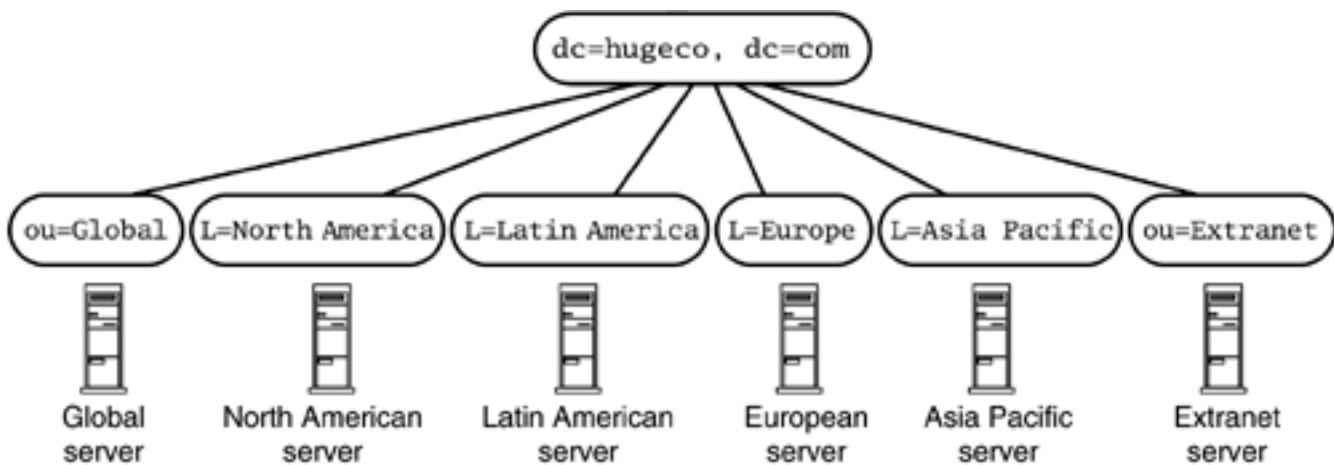
So why is it desirable to include the retailer ID in each person entry? Sometimes it is necessary to map from an employee's entry back to his or her retailer. Although it would be possible to walk up the directory tree until an `hugeCoHrpRetailer` entry was located, storing the retailer number in the entry allows the corresponding retailer record to be located with a single search operation. (For an example of an application that performs exactly this operation, see the discussion of how personal start pages are generated in the Directory Service Deployment section of this chapter.)

Topology

HugeCo's directory is partitioned along regional lines, with each major locality mastered in a different server and replicated to all the other servers via a central replication hub (refer to [Figure 25.7](#) in [Chapter 25](#)).

When adding the extranet subtree to the directory tree, HugeCo's directory designers opted to create an additional directory partition holding all the data at or below the `ou=Extranet` entry. This approach, which is consistent with the existing partitioning scheme, is depicted in [Figure 26.5](#).

Figure 26.5. HugeCo Directory Partitions, Including Extranet Data



The designers initially used referrals in each of the regional servers to link the extranet data into the directory information tree (DIT). Whenever any of the regional servers receive an LDAP operation that needs data from the `ou=Extranet` subtree, a referral to the extranet server is generated.

20/20 Hindsight: Search Performance across All Corporate Directory Data

One negative consequence of linking the extranet directory data into the HugeCo main directory tree is that searches across the whole organization (for example, searches rooted at `dc=hugeco, dc=com`) require the LDAP client to chase a referral and then combine results from the two servers before presenting them to the client or directory-enabled application. Performance is slower than it would be if all entries were held within a single server.

In addition, some server capabilities become less useful in a distributed environment. For example, server-side sorting and paging through result sets works only within a single server. If a client application needs to contact two servers to satisfy a user's search request, and the user has specified that the results should be sorted, the client application needs to merge the entries returned from each server into a single list before presenting it to the user.

After some more thought, HugeCo staff realized that the HRP extranet applications didn't really need to be aware of the intranet directory data, and the intranet applications had no real need to be aware of the extranet data. Therefore, the designers decided to remove the referrals from the regional servers, essentially making the extranet data a separate, unconnected directory—which is acceptable for the time being. In the future, if the extranet and intranet directory data need to be combined, the replication architecture can be altered so that extranet data is replicated to each of the regional servers.

Replication

The primary goal of replication in HugeCo's existing directory deployment is to increase the read and search capacity of the directory in an incremental fashion. HugeCo decided to stick with this approach as it extended its applications to the extranet.

Extranet data can be replicated as needed to increase capacity for the extranet applications. HugeCo planned initially to deploy a master server and two read-only replicas, thereby increasing the reliability and scalability of the extranet directory data. The client connection load is balanced across the two read-only servers with a Nortel Networks Alteon Load Optimizer load balancer. The virtual IP address of the load-balanced directory servers is known by the domain name hrpdirectory.extranet.hugeco.com. If additional replicas need to be added, they can be deployed and load-balanced in the same fashion.

Privacy and Security

The security design for HugeCo's internal directory server deployment focused on two major issues: access control on directory data to support delegated administration, and protection against attack.

Access Control

Before we discuss the access control policies in effect for the directory, we should clarify that the order entry and tracking application has its own set of access control policies that are separate from the directory access control policy. The directory is used to authenticate users, produce the personalized start page for retail employees, and determine whether a given user may access a given URL within the order entry application. The directory is used merely as a repository for the information that the Web application uses to make its access control decisions.

The access control policy in effect for the directory itself controls who may access the actual directory content. HugeCo has a restrictive access control policy in effect for its directory data. Employees may update only a few fields in their own entries.

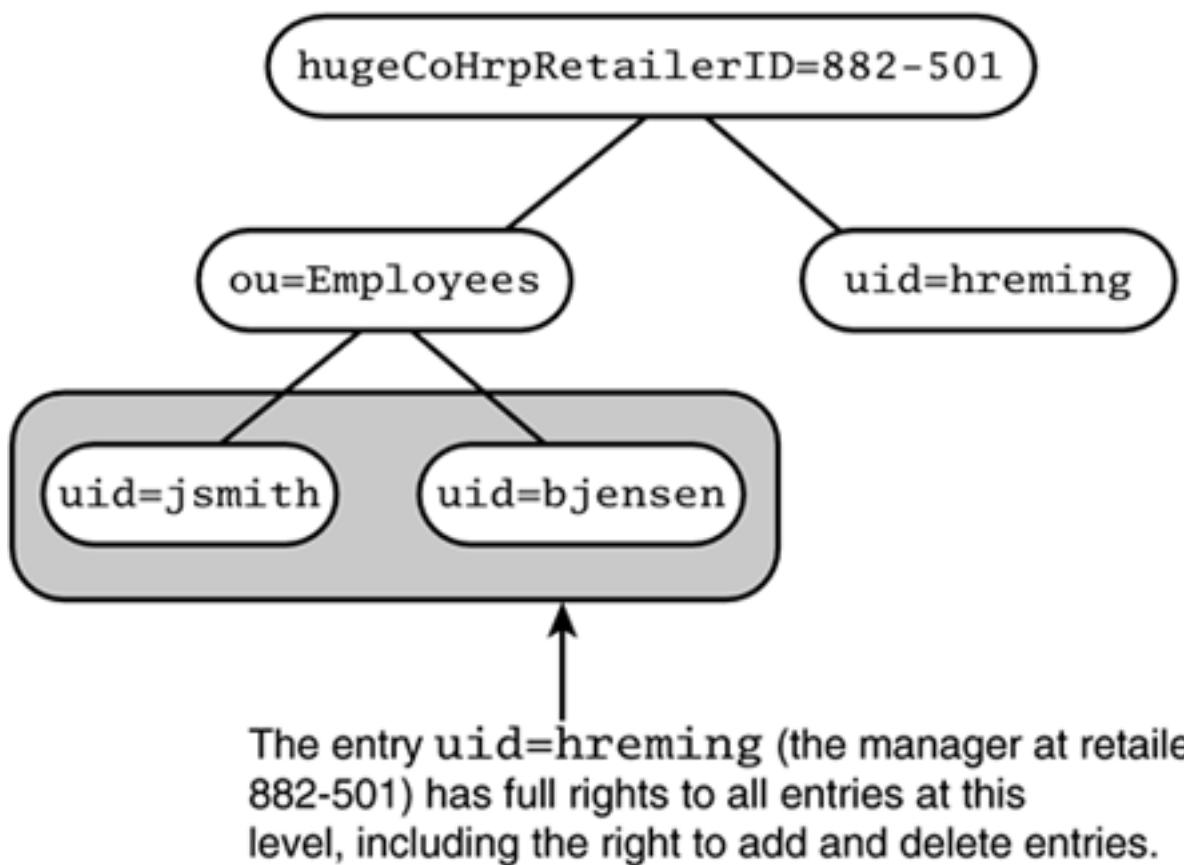
Within the subtree used to hold information about employees of authorized retailers, a different set of access control policies is in force. The goal of these policies is to delegate administration of the individual retailer information to a responsible person at each retailer. This approach not only reduces costs for HugeCo, but is absolutely necessary: Employees of the retailers are not HugeCo employees, so HugeCo has no knowledge of when these employees are hired or terminated.

To meet this requirement, the following access control policies were developed:

- Each retailer must designate one or more points of administrative contact. These administrators are responsible for adding and deleting local users and updating certain information about the retailer.
- The administrative contacts may create and delete users as they see fit.
- Users created by an administrative contact may not update their own entries.
- Only HugeCo personnel may add or delete the entry representing the retailer.
- Only HugeCo personnel may add or delete the entry representing the administrative contact.

See [Figure 26.6](#) for an illustration of this delegated access control policy.

Figure 26.6. The Delegated Access Control Policy



Protection against Attack

Connections from HugeCo's authorized retailers to the Web-based application and to the directory traverse the public Internet. To protect against eavesdropping and man-in-the-middle attacks, HugeCo's security design team decided to use Secure Sockets Layer (SSL) session-layer encryption to protect sensitive data during transit. With SSL in place, even if data were intercepted in transit, the eavesdropper would not be able to make use of the encrypted data without a great deal of effort.

Dangers of Delegation

When you're delegating privileges, it's also important to consider the capabilities that those privileges grant, especially with respect to directory-enabled applications. For example, what would happen if administrative contacts at retailers were able to create entries with the same user identifier (UID) as an existing entry somewhere else in the directory? If the new entry were within a separate directory tree, its name wouldn't collide with the other entry; however, applications that expect UIDs to be unique across the directory (authentication applications behave this way) may malfunction when they discover multiple matching entries.

To prevent this problem, applications should check before adding new entries to make sure that no undesirable attribute collisions occur. If, for example, someone attempts to add a new entry with the UID `jsmith` to a directory, but there is an existing `jsmith` somewhere else in the directory, the application could reject the

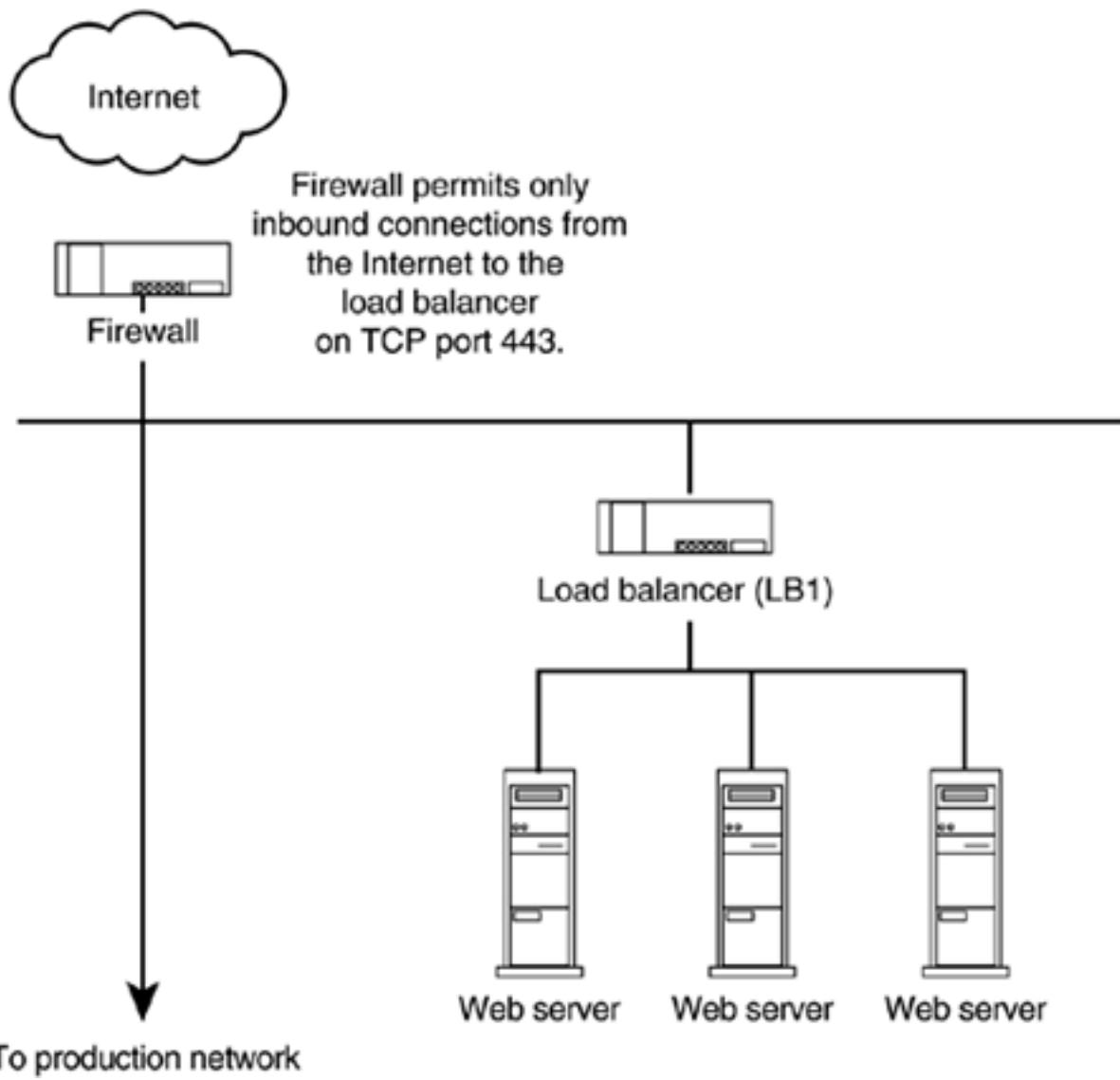
addition with a helpful error message. Similarly, the application could check whether modifying or renaming an entry is legal and does not cause a collision.

However, this approach requires that you force all updates to the directory through a single application such as a Web front end. Thus you lose the ability to deploy general-purpose clients, and there may be no practical way to prevent someone from using one of these general-purpose clients.

A much more foolproof solution is to enforce the attribute uniqueness constraints on the server itself. Netscape Directory Server (version 4.0 and later) includes a general attribute uniqueness plug-in that allows the server to reject any update to the directory that causes an undesired attribute value collision. For example, the plug-in can be configured to reject any modifications to directory data that would cause two entries to have the same UID. Of course, it is still up to the user interface to provide a helpful error message in the event of a naming collision. But using this plug-in means that no matter how clever or determined a person is, he or she cannot bypass the uniqueness constraints you have configured.

In addition, firewall technology is used to reduce exposure to hostile parties on the Internet. The Web servers in the application architecture are protected by a firewall that permits only inbound connections to the HTTPS (HTTP-over-SSL) port (443). Inbound connections to any other port are prohibited, and all outbound connections are prohibited. [Figure 26.7](#) illustrates this part of the architecture.

Figure 26.7. The Web Tier of HugeCo's Extranet Architecture

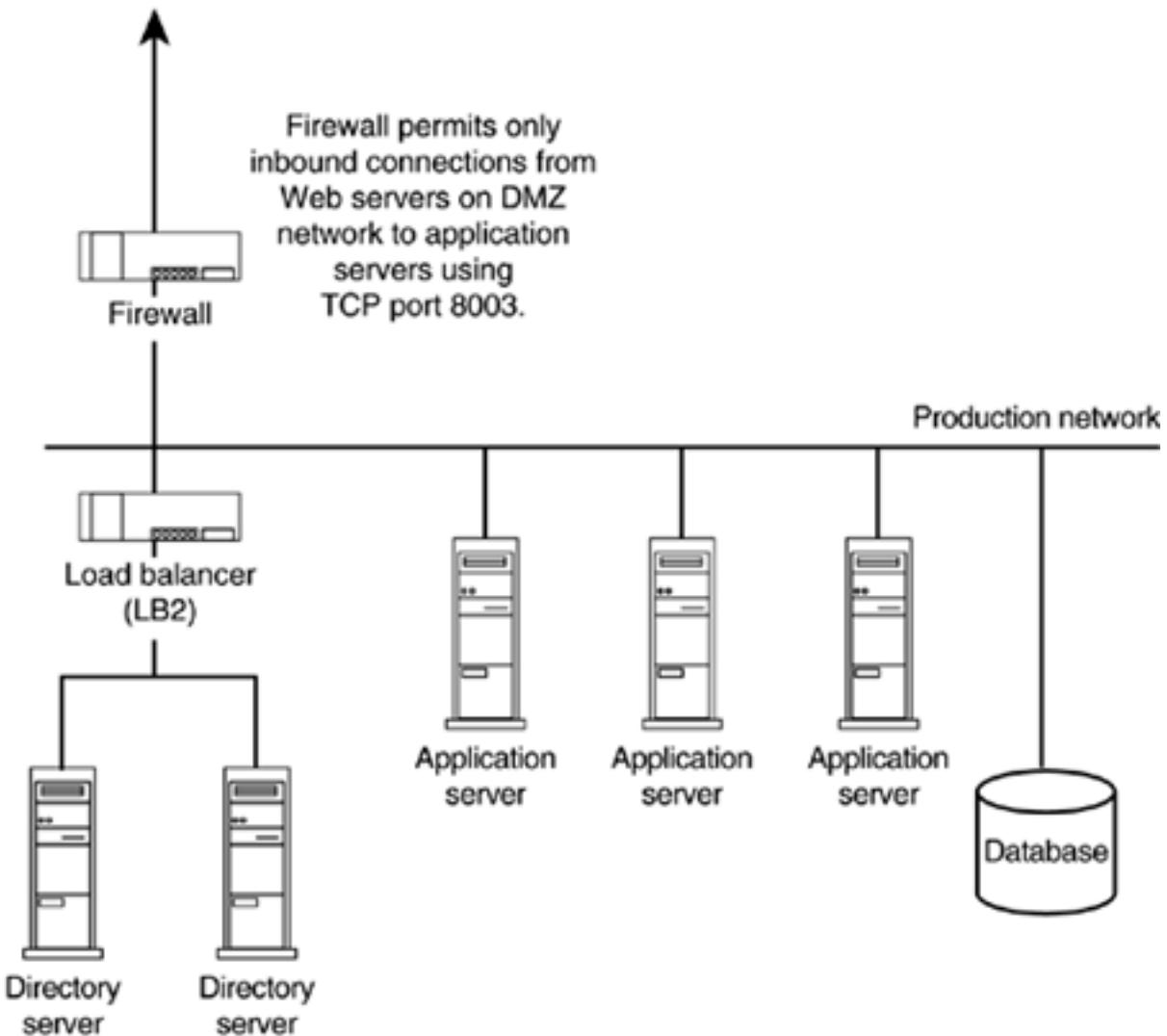


Preventing outbound connections keeps your servers from being used as springboards for further attacks if they are compromised. You can also use an intrusion detection system to detect when outbound connections originate from your Web servers to external addresses. Such outbound connections almost certainly indicate that your servers have been compromised.

The application servers that support the business logic of the application, the directory servers, and the database server are further protected by an additional firewall that permits traffic to pass only from the Web servers to the application servers, and only on TCP port 8003, which is the port that the Web servers and application servers have been configured to use. [Figure 26.8](#) depicts this layer of the application architecture.

Figure 26.8. The Application Tier of HugeCo's Extranet Architecture

To DMZ network



Inserting an additional layer of firewall protection adds further protection for the sensitive data stored on the directory and database servers. Even if the Web server is compromised by an outside attacker, the only further attack that can be mounted is one that originates from one of the Web servers and is directed against the application servers using TCP port 8003. Although the additional firewall doesn't make it impossible for your data to be compromised, it makes it much more difficult and offers a much higher level of security than would be possible if the firewall were not present.

Directory Service Deployment

After HugeCo completed the design of the extranet applications and directory, it embarked on the deployment phase of the project. HugeCo used a phased approach: Products were chosen, an extranet pilot project was deployed, and then the production service was rolled out to all retailers.

Product Choice

Because HugeCo had already deployed Netscape Directory Server for its corporate intranet, continuing to use it for the extranet applications made sense. Nevertheless, HugeCo analyzed the requirements of the extranet directory to verify that the Netscape product was suitable. This analysis detailed the types of operations that the extranet applications would perform against the directory. The following operations were included:

- **User authentication and authorization.** Whenever a user logs in to the order entry and tracking system, his or her user ID and password are sent to the application server via the Web server, which verifies the information and looks up the corresponding entry in the directory. A browser cookie is returned to the client, which associates further activity with the user. Because no further interaction is required with the directory until the user logs in again, user authentication and authorization place a very light load on the directory.
- **User start page generation.** A user's personalized start page includes a list of relevant product bulletins or announcements that the user has not yet viewed. To generate this list, two directory entries are retrieved. First the user's entry is retrieved and its `hugeCoHrpRetailerID` attribute examined. Then the corresponding retailer entry is retrieved from the directory, and the `hugeCoHrpProductAuthorized` attribute is extracted. When these pieces of information have been retrieved, the application has enough information to display the new product notices for only those products that the retailer is authorized to sell. When a new order is placed, the order entry and tracking system consults the retailer's directory entry to verify that the retailer is authorized to sell the ordered product.

The types of queries required by these applications are well within the capabilities of Netscape Directory Server. Hence, HugeCo decided simply to install additional instances of the Netscape server to support the extranet application. If, on the other hand, features required by the application were not provided by the Netscape server, HugeCo would need to evaluate other directory servers for suitability, or redesign the applications to work in the absence of those features.

Piloting

As with the original directory deployment, HugeCo rolled out the extranet applications and directory in an incremental fashion. The process began with a small-scale pilot that provided valuable feedback and validation of the directory design. The pilot involved three different types of retailers—a home improvement superstore, a small-town hardware store, and a homebuilder—in different parts of the United States. Selecting these three types of retailers provided a set of users with a wide range of computer experience. The large home improvement center, for example, has its own computer consultant (shared with the other stores in the region), whereas the homebuilder serves as his own computer consultant.

HugeCo staff visited each pilot site. They assisted the retailer at each site with hardware and

software installation, created entries and issued passwords for the users, and then stayed for three days as the pilot users began to use the extranet system. During this time, any difficulties the users encountered while using the system were noted and fed back to the system developers at HugeCo. Of particular interest to the developers were the experiences of the local administrators as they created and removed accounts for their staff using the administration utility hosted on the extranet Web server.

Also of interest to the extranet developers was the quantity of directory load placed on the extranet directory servers. During the pilot the developers kept logs for the extranet directory servers and observed how each individual retailer generated load on the directory. Extrapolating from the pilot usage data allowed the developers to validate their assumption that three directory servers (a writable master and two read-only replicas) would be sufficient for the needs of the extranet applications. Of course, if any new applications are deployed in the future, the load on the directory might change significantly, so this assumption must be revisited periodically throughout the lifetime of the extranet.

After a pilot site was installed, the HugeCo staff moved on to the next pilot site but remained available by telephone for assistance. After all the pilot sites were installed and operational, the pilot staff returned to each pilot site in order and interviewed the pilot users. The collected feedback from all the pilot sites and the usage data from the extranet directory servers were reviewed by the developers, and changes were incorporated into the system. Most of these changes involved alterations to the user interface of the order entry system.

Putting Your Directory Service into Production

After the pilot was completed and applications were updated to incorporate feedback, HugeCo planned its production rollout of the service. Although HugeCo believed that the application would be easy to use, it decided to be conservative in the number of sites to be brought online, just in case the retailers needed a lot of telephone support. To avoid swamping the Help Desk with calls, only 25 retailers were initially brought online. One week after the retailers went live, follow-up calls were placed to make sure that they were successfully using the application.

After several weeks of enabling 25 retailers per week, HugeCo was confident that it could handle a larger volume. This conclusion was based on the frequency of support calls and the usage patterns on the extranet directory servers, which were increasing as expected but were well within the capacity limits of the servers. Volume was increased to 100 retailers per week, and all retailers were brought online within ten weeks.

To encourage use of the new application, HugeCo tracked the usage of the extranet applications and the telephone order center. Initial results were disappointing—the call center volume actually increased for a period of time—but the help calls eventually began to decline as retailers became more comfortable using the system.

Data was collected and analyzed during the entire process to determine if the directory was in danger of becoming overloaded. On the basis of the experience they had gained while deploying HugeCo's corporate intranet directory, staff members felt confident about their abilities to perform capacity planning for the new extranet applications. Basic performance metrics, such as average search response time, were developed and measured throughout the directory deployment phase, and directory load remained well within expected levels.

Directory Service Maintenance

In this section we describe the various procedures used to maintain HugeCo's extranet directory.

Data Backups and Disaster Recovery

Backups of the extranet directory servers are handled in the same manner as backups of the other directory servers. A digital linear tape (DLT) drive is attached to the master extranet directory server, and backups are performed nightly at 0200 hours (2 A.M.) local time via the Unix cron daemon. Specifically, at 0200 hours the Netscape Directory Server `db2bak` utility is run to generate a database backup on the local disk. Immediately after the `db2bak` utility completes, the backup directory is copied to the backup media via the Unix tape archive (tar) command, and verified.

After the tar file has been copied to disk, the cron job removes any backup files older than three days. This means that the most recent three days' worth of backups are always on the server's disk and can be quickly restored in an emergency (for example, if an errant update procedure deletes important data from the directory). The tapes are transferred off-site twice a week and stored in a secure facility.

Disaster recovery services for extranet applications were added to the contract that HugeCo maintains with a disaster recovery vendor. The disaster recovery approach for the extranet application provides for a cold standby site. In the event that a disaster renders HugeCo's corporate data center unusable, the cold site will be brought online, the directory server software installed, and data restored from off-site backups. The use of a cold standby saves a considerable sum of money, but it means that recovery time for the application will be measured in days, not hours.

This recovery time was deemed acceptable, given that HugeCo maintains a hot standby for the main internal order entry system. In the event of a disaster, the main database and order entry application will fail over to the hot site. The extranet order entry site will be unavailable, but retailers will be able to place orders via the HugeCo call center and track those orders via the extranet application once it is brought back online.

Maintaining Data

Maintenance of extranet directory data involves two data sources: the Oracle database that tracks information about the authorized HugeCo retailers, and the managers at the individual retailers. Each of these data sources is considered authoritative for certain directory information.

The Oracle database is considered the authoritative source for information about the individual retail outlets. The retailer name, telephone and fax numbers, mailing address, retailer number, and list of authorized products are all synchronized from the Oracle database into the directory on a regular basis via a set of Perl scripts developed by HugeCo's IS staff. An established procedure is used to add or remove retailers from the Oracle database, and these changes propagate to the directory via the synchronization scripts.

The entries corresponding to individual employees at the retailers, on the other hand, are owned by the manager at each particular retailer. When a new employee is to be granted access to the extranet applications, the manager uses a special Web-based application to create a new entry in the directory; this application creates the directory entry for the

employee and sets an initial password. Similarly, when an employee leaves a retailer, access must be revoked. The manager can accomplish this by using the same Web-based application.

20/20 Hindsight: Preventing Stale Directory Data from Accumulating

Delegating the creation of new employee entries to the individual retail managers is an effective way to cut down on costs. In fact, it's absolutely necessary because HugeCo's Human Resources division has no record of these employees at all.

However, it's also necessary to take steps to ensure that stale directory entries do not accumulate. When employees are terminated, it's the responsibility of the retailer's manager to remove their records. But what happens if the manager forgets to do this? The initial directory design depended on managers to perform this task, and it did not include any method for alerting the manager to the presence of stale data.

To prevent stale directory entries from accumulating, an automatic expiration system was put in place. Each employee entry in the database (except for the manager's entry) is created with an expiration date six months in the future. One month before expiration, the manager is alerted to the fact that the entry is about to expire. The manager can easily reinstate the employee for an additional six-month period by clicking a button in the user management application; if the employee is not reinstated, the entry is removed from the directory. Behind the scenes, a Perl script (which uses the PerLDAP module) runs nightly and searches for employee directory entries that are about to expire. For each such entry found, the script arranges for the manager of that entry to be notified. The same script removes entries that have expired.

When a manager leaves his or her position with a retailer, HugeCo administrative staff remove his or her entry, and they add a new record when a new manager is hired. HugeCo representatives periodically contact the HRP retailers via telephone as part of a regular administrative procedure, and this is frequently the point when they discover that a manager has left a retailer. If a new manager has been appointed, a new entry is created, and access control lists (ACLs) in the directory are altered to grant appropriate privileges to the new manager.

These steps keep directory data from becoming stale, thereby improving its quality and usefulness.

Monitoring

HugeCo extended its existing monitoring system to monitor the extranet application services in the following ways:

- eHealth SystemEDGE agents from Concord were installed on each of the three directory servers (master and two replicas). These agents verify that the ns-slapd process is running and generate an alert if it is not.
- Custom code written in PerLDAP (see [Chapter 19](#), Monitoring, for similar code) probes the directory on the LDAPS (LDAP-over-SSL) port (636) by retrieving an entry. The response time of the server is recorded, and alerts are generated if the probe fails or does not respond in a reasonable amount of time. The monitoring tools perform this type of probe because it closely matches the type of operation that the directory will service in handling client requests.

Troubleshooting

Procedures were added to the existing HugeCo directory escalation process to accommodate the Web and directory servers that support the new extranet applications.

Team LiB

◀ PREVIOUS

NEXT ▶

Leveraging the Directory Service

With the successful deployment of the HRP extranet, HugeCo directory designers have begun to make plans for additional directory-enabled extranet applications. These applications will continue to leverage the directory to improve service to HugeCo retailers and customers.

Directory Deployment Impact

The HRP retailer extranet has enhanced HugeCo's business in the following ways:

- Product literature and prices are disseminated more quickly and at a lower cost to HugeCo.
- Retailers receive more proactive notification of product announcements, special promotions, and so on via the personalized start page.
- Retailers can communicate more effectively with HugeCo concerning outstanding orders. Orders can be placed and tracked 24 hours a day, 365 days a year, and any clarifications required by craftspersons on the assembly line can be handled electronically.

Summary and Lessons Learned

As with any complex process such as deploying a directory, there is always room for improvement. Here are a few words of advice based on lessons learned during the extranet directory deployment at HugeCo.

During the design, pilot, and deployment phase, HugeCo developers chose to revisit several decisions they had made in response to the following situations:

- The original namespace design, in which all extranet application data was held within a single `ou=Retailers` subtree, would not have scaled well as additional types of extranet applications were added. Many additional container entries would have needed to be created at the `dc=hugeco,dc=com` level of the DIT to accommodate the applications. By placing an additional `ou=Extranet` container at this level, HugeCo's developers gained additional flexibility to arrange the extranet namespace in a more scalable fashion.
- The original server topology, which tied the extranet and intranet directory data together via referrals, had negative performance implications. The developers chose to keep the intranet and extranet directories separate for the time being because no applications needed to use both sets of directory data. This decision might be revisited in the future if intranet and extranet data needs to be shared between applications.
- Maintaining the quality of the retailer employee information was delegated to the managers at each authorized retailer, but there was initially no way for the manager to find out about stale directory data. A system was developed in which entries automatically expire unless they are reinstated by the manager, who is notified of the impending expiration.
- Associating entries with one another on the basis of location in the DIT proved to be troublesome. It is possible to locate the retailer entry for any given employee by moving up exactly two levels in the DIT. However, what would happen if the layout of directory entries changed? What if all the employee entries were moved beneath another container within the retailer subtree? Retailer entries would then be three levels above, instead of two. A better choice, implemented in the HugeCo HRP extranet, is to place an attribute in an employee's entry that associates it with a particular retailer. The `hugeCoHrpRetailerID` attribute serves this purpose and decouples the method of locating a retailer's entry from the DIT structure.

As new extranet applications are designed and deployed, some of the design decisions will no doubt need to be revisited. The process of incrementally adding new directory-enabled extranet applications is constantly evolving and being refined.

Looking Ahead

Overall, the HugeCo HRP extranet was an excellent first step into the world of extranet applications, and it expanded on the expertise developed when HugeCo designed and deployed its intranet directory service. The expertise developed should serve HugeCo well as it moves forward and leverages its directory to enable even more interesting extranet applications.