Developing .NET Enterprise Applications

JOHN KANALAKIS

Apress™

Developing .NET Enterprise Applications

ISBN (pbk): 1-59059-046-5

Printed and bound in the United States of America 12345678910

Technical Reviewer: Phil Pledger

Editorial Board: Dan Appleman, Craig Berry, Gary Cornell, Tony Davis, Steven Rycroft, Julian Skinner, Martin Streicher, Jim Sumser, Karen Watterson, Gavin Wray, John Zukowski

Assistant Publisher: Grace Wong

Project Manager: Tracy Brown Collins

Copy Editor: Kim Wimpsett

Production Manager: Kari Brooks

Proofreader: Lori Bring

Compositor: ContentWorks

Indexer: Ron Strauss

Artist: April Milne

Cover Designer: Kurt Krames

Manufacturing Manager: Tom Debolski

Distributed to the book trade in the United States by Springer-Verlag New York, Inc., 175 Fifth Avenue, New York, NY, 10010 and outside the United States by Springer-Verlag GmbH & Co. KG, Tiergartenstr. 17, 69112 Heidelberg, Germany.

In the United States: phone 1-800-SPRINGER, email orders@springer-ny.com, or visit http://www.springer-ny.com. Outside the United States: fax +49 6221 345229, email orders@springer.de, or visit http://www.springer.de.

For information on translations, please contact Apress directly at 2560 Ninth Street, Suite 219, Berkeley, CA 94710. Phone 510-549-5930, fax 510-549-5939, email info@apress.com, or visit http://www.apress.com.

The source code for this book is available to readers at http://www.apress.com in the Downloads section.

CHAPTER 1

Introducing .NET and Enterprise Architecture

An *enterprise application* is one that supports any multidivisional organization with data and services that enable business-specific functions. Such applications typically integrate multiple technologies, such as database access, directory services, mail services, business processes, and more. This chapter defines the components of an enterprise application and then explores the technologies involved in creating one.

Finally, the chapter introduces IssueTracker, the fully functional enterprise application that this book builds with each subsequent chapter. The IssueTracker example application will serve as a central repository for issues within an enterprise, such sales requests, support issues, development bugs, and executive status. As an enterprise-wide application, IssueTracker will interact with multiple server-side resources, such as databases, e-mail servers, process management engines, directory services, Web services, reporting services, and mobile services. Starting with this chapter, each chapter outlines the best practices that yield maximum performance, reliability, and scalability.

Defining Enterprise Applications

An *enterprise* represents a corporate entity with several interacting departments, each managing different types of data and performing different functions. Figure 1-1 models a typical enterprise. It is composed of several departments including Sales, Marketing, Human Resources, Development, Customer Support, and Information Technology. Each department operates independently but shares specific data or services. For example, the Sales department tracks the names and contact details of customers. The Support department refers to the customer information to help customers experiencing problems with the company's product and to enter customer problem information. The Development department reviews the customer problem information to improve the product and update the status of the problem. Finally, the company's management monitors the customer and problem information captured in an effort to improve the company's operations.

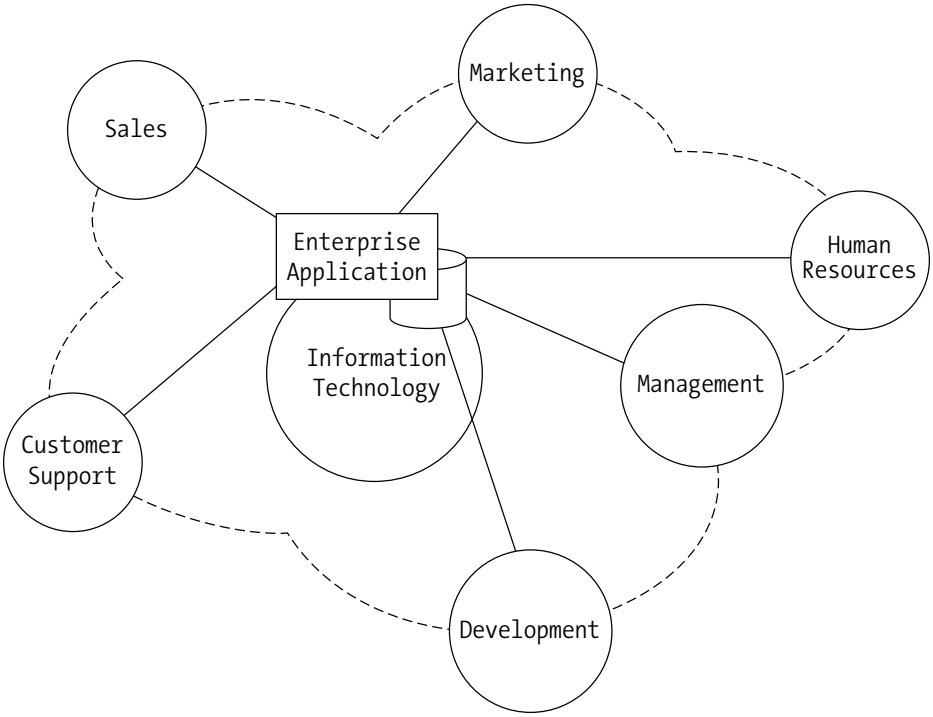

Figure 1-1. Department relationships within a typical enterprise organization

For an enterprise application to support different departments, it needs to manage two important company assets: business data and business processes.

Introducing Business Data

Business data is any data that helps a business operate more effectively. It typically needs to be accessed by different departments within an organization. Business data needs to be reliably accessible, scalable, and secure.

Accessibility refers to the capability of multiple users operating on a variety of platforms and devices being able to interact with the business data. Flexible enterprise applications will have more than one form of user interface. In most cases, one primary user interface client will be developed, and one or more additional clients will be developed. This scenario might include a primary Web-based client and supporting clients in the form of a Windows application, a Pocket PC application, or a cellular phone application. All clients are accessibility tools that help users interact with business data and services.

Scalability refers to the capability of increasing the number of system resources, such as servers, to support an increase in the number of connected users without impacting performance. Typically, smaller database applications,

such as Microsoft Access or Microsoft Visual FoxPro, can reliably support large amounts of data storage but with only a handful of concurrent connections. An enterprise database, such as Microsoft SQL Server, Oracle, or MySQL, has built-in support for handling larger volumes of data with hundreds of concurrent connections, as well as transaction and backup services. However, such databases are significantly more expensive and require specialized maintenance.

To *secure* business data, you offer data access to specific users privileged to access that data. There are many different levels of security. For example, you could fundamentally grant access to employees and block everyone who does not work at the company. Then, within a company, you could grant access to department team members and block everyone else. Finally, you could grant access to certain levels of management and block everyone else. In this scenario, you need to look at the data closely to determine which employee should access which data. Also, you need to determine the type of data access: read-only, read-write, or no access.

Introducing Business Processes

A *business process*, or workflow, is a series of steps that accomplishes a specific business goal. Business processes are typically either internal or external. Internal processes might span multiple departments, and external processes usually span multiple enterprises. Both process types might combine automated tasks with human interaction tasks.

An example of an internal business process is an employee new-hire process. When the Human Resources department enters a new employee's personal data, that data is validated for completeness. Then, the Information Technology (IT) department is notified to create a new user account, create a new telephone account, create a new e-mail account, assign a computer, and then notify the Human Resources department when completed.

An example of an external business process is a supply-chain process. When one company orders merchandise from another, the supplier inventory is checked. If the item being purchased is out of stock, then the supplier orders from the distributor. If the distributor is also out of stock, the product is ordered from the manufacturer.

Business process automation solutions implement processes that combine automated and human interaction tasks and that provide a user interface for graphical modeling.

Introducing the Enterprise Application Components

Developing an enterprise application requires interactions with many components, including data access, business rules, and the user interface. Figure 1-2

illustrates a typical enterprise application architecture and its software components.

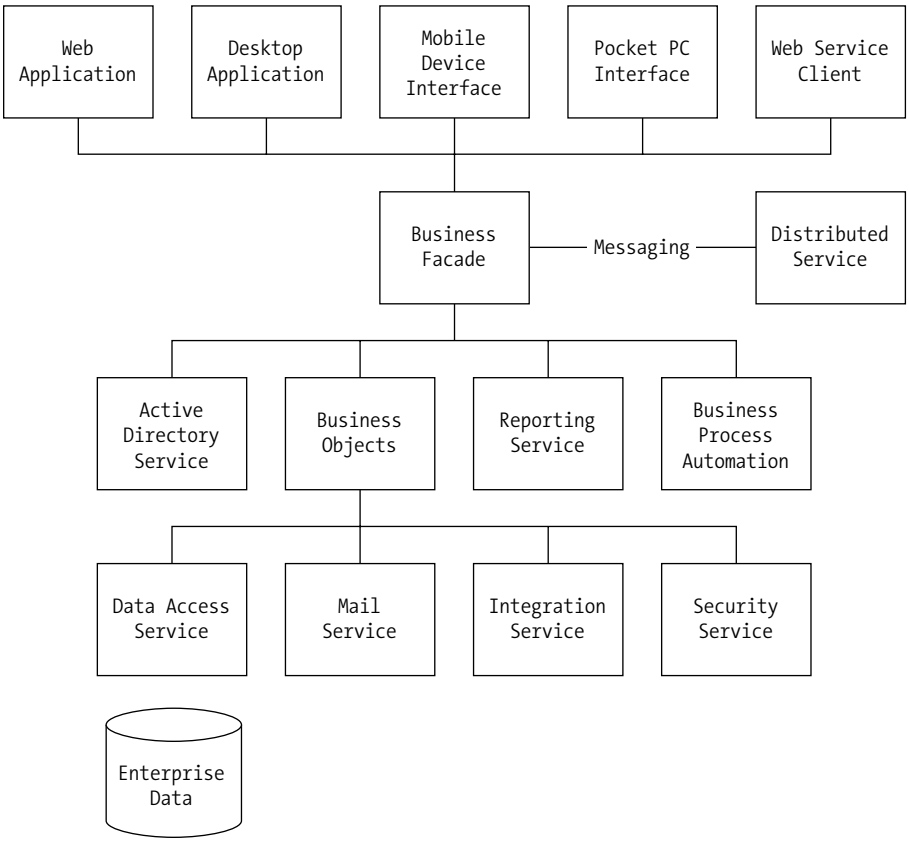

Figure 1-2. Enterprise application components

As Figure 1-2 illustrates, multiple technologies work together to implement an enterprise application:

Database access services: Database access technology is fundamental to any application regardless of whether it is an enterprise application. The purpose of any application is to interact with data by storing, retrieving, and searching for specific values. The .NET Framework provides the Active Data Objects (ADO) framework as the primary toolset for reading from and writing to databases. As explained in Chapter 2, "Accessing Data in ADO.NET," ADO.NET provides different data access mechanisms optimized for different types of data sources.

Directory services: Directory services provide a single point of entry into a networked application. With Active Directory, users can be validated and then access networked resources such as directories, printers, and so on from a single login. Applications can leverage directory services to search the network for system resources, such as files and printers. Active Directory also supports third-party directory service solutions found in many enterprises, such as the Novell Directory System (NDS) and the Netscape Directory Server. Chapter 3, "Using Networking and Directory Services," covers directory services and their role in an enterprise application.

Messaging services: Messaging services provide a mechanism by which applications can reliably send and receive messages between applications within the enterprise or across multiple enterprises. Message content can vary between applications and can initiate a process on another system. Messages are also reliable in that if they fail to reach their destination, they are queued and resent when the destination is accessible again. Chapter 4, "Applying Reliable Messaging," covers messaging services and the Microsoft Message Queue (MSMQ).

Mail services: Mail services enable applications to send and receive standard e-mail messages between applications. Because many enterprise applications link multiple users together, e-mail-based notification systems can be an effective means of notifying users when a server-side process ends or when human interaction is necessary to complete a workflow. Support for mail services is undoubtedly a cornerstone technology that helps an enterprise application tightly integrate into the organization. Chapter 5, "Integrating Mail Services," covers integrating mail services into the application framework.

Business process automation: Business process automation enables users to define a series of repeatable tasks that helps accomplish a specific business goal. You can start these processes by hand, by an application-triggered event, or by the presence of a document within a specified location. Processes can combine automated tasks, such as routing documents from one place to another, and human interactive tasks, such as setting a manager's approval. Business processes are valuable to enterprise applications in that they streamline business operations so that tasks are accomplished faster and with fewer resources. They also help link different parts of the enterprise. Chapter 6, "Automating Business Processes," covers how to define business processes and incorporate them into an enterprise application.

Web applications: Web applications are software solutions that reside largely on a remote server. Users interact with these remote servers with a thin client, such as a Web browser. Although Web applications are easily deployable to multiple operating system platforms, the result is a simple user interface that offers slower performance. For many enterprise developers, choosing to develop a thick desktop client vs. a thin Web client is a difficult decision. Chapter 7, "Building Web Applications," covers developing Web applications that tap into core business functionality.

Desktop applications: Desktop applications reside largely on each user's computer. All application logic executes locally but typically connects to a remote database. Desktop applications often have a much richer user interface than their thin Web application counterparts. Desktop applications also execute much faster in terms of screen refreshing. Chapter 8, "Developing Desktop Applications," covers building deployable desktop applications that integrate core business functionality.

Web services: Web services provide a framework for platform-independent data exchange based on published open standards. Web services implement or wrap specific business functionality and return platform-independent documents structured in the Extensible Markup Language (XML) format. Web services are ideal for building inter-enterprise application integration solutions. Some think of a Web service as a Web site without the user interface. Chapter 9, "Using XML and Web Services," includes a detailed look at Web services and how they can enhance accessibility in an enterprise application.

Reporting services: Reporting services provide a mechanism for generating structured reports in tabular or graphical format. Reports communicate status or statistical information relating to the enterprise application's data. Although you can also implement reporting within a dynamic Web page or a dynamically rendered Windows form, using a reporting technology such as Crystal Reports streamlines the report-building process and makes it easier to create and deploy new reports. Chapter 10, "Integrating Reporting Services," covers building a reporting layer that integrates with an enterprise application.

Wireless services: Wireless services expose enterprise application data and services to mobile devices, such as Personal Digital Assistants (PDAs) or Web browser–enabled cellular phones. Wireless services help extend an enterprise application's reach to employees outside of the office, such as a field service representative who is at a customer's site and needs to check back with the office. Enterprise applications supporting wireless extensions offer additional value to customers in need of application data access anywhere and anytime. Chapter 11, "Deploying .NET Applications on Wireless Devices," covers building a wireless application layer that targets PDAs and Web-enabled cellular phones.

Integration services: Application integration binds different applications together to work as a single application that solves a single problem. There are many different techniques for accomplishing application integration that depend on the type of applications being integrated. Chapter 12, "Integrating .NET Applications," outlines different approaches as well as the advantages and disadvantages to each approach.

Security services: Securing enterprise data and services is an increasing concern to most consumers of enterprise applications. Chapter 13, "Understanding .NET Security and Cryptography," explores potential security holes within an enterprise application and outlines implementation strategies to ensure that an application is designed and built with security in mind.

Deploying an enterprise application into a customer environment might be the last step, but you should not take it lightly. The deployment process is often the first impression that an application makes to its user. If that process is too lengthy, too complicated, or requires too much user involvement, users might be turned off to the application before they even use it. Chapter 14, "Installing .NET Applications," dives into the details and best practices for building easy-to-use deployment projects.

Leveraging the .NET Servers

In some respect, you can think of Microsoft's .NET platform as the next generation of Windows DNA, an earlier platform for developing enterprise applications. Windows DNA included many foundation technologies still found within the .NET Framework, such as Microsoft Transaction Server (MTS), COM+, MSMQ, and Microsoft SQL Server. The .NET Framework incorporates these technologies and adds a Web service framework around them.

The .NET servers offer developers the flexibility to develop scalable enterprise applications. Enterprise applications can leverage the .NET servers to build scalable applications quickly and reliably. The .NET servers include the following:

- **Application Center Server:** This server manages and monitors clusters of application servers and is primarily for managing application scalability.
- **Application Center Test Server**: This server tests and measures the scalability of enterprise applications by scripting and simulating concurrent connections and measuring response times.
- **BizTalk Server**: This server integrates applications and enterprises by graphically modeling business processes. Processes can trigger system resources and integrated applications or wait for human interaction.
- **Commerce Server**: This server supports the development of e-commerce applications with tools and templates for building Web components and services.
- **Exchange Server**: This server offers messaging and collaboration services that include sending and receiving e-mail, scheduling appointments, managing tasks, and keeping journals.
- **Host Integration Server**: This server offers integration with legacy applications, such as SAP, PeopleSoft, and Siebel Systems, with specialized data adapters.
- **Internet Information Server**: This Web server distributes static and dynamic Web content and services to remote Web-based clients.
- **Internet Security and Acceleration Server**: This server offers firewall and Web caching services.
- **Microsoft Message Queue Server**: This server manages and routes messages between applications within and between enterprises.
- **SharePoint Portal Server**: This server provides a functional portal supporting content management, search, and crawling, along with subscription management.
- **SQL Server**: This server manages enterprise data and accessibility to that data with support for caching, indexing, transactions, stored procedures, and backup/restore services.

Understanding the Enterprise Development Process

The success of any enterprise application depends upon how well its development can follow a rigid process. This process typically maps into six specific steps: capturing the requirements, prototyping, designing, developing, testing, and releasing the application.

Capturing the Requirements

The most critical step in the development process is capturing the requirements. Product managers need to listen closely to customer needs and expectations. They need to document these requirements with clear descriptions of what data is entered, what data is returned, and how the data is processed. In addition to application requirements, product managers should discuss any type of printed reports expected by the customer. Specifics include the type of report, formatting and graphic preferences, and performance expectations.

Prototyping the Application

Prototyping essentially helps an enterprise developer validate their understanding of the product requirements. Typically, a prototype will be a lightweight and featureless version of the application. With the help of Visual Studio .NET, you can build rich user interfaces for desktops as well as Web applications fairly quickly. You often present these prototypes to key customers for feedback to ensure that the customer needs are fulfilled. Committing a long development effort to a product that doesn't meet customer expectations will be nearly impossible to sell. Prototypes also help to better estimate the total development effort and, in turn, the total cost of development.

Designing the Application

The design step should be the longest in the development process. You cannot build an enterprise application ad-hoc with a general concept in mind. You need to have detailed discussions about security, performance, and scalability. The first step is to outline the high-level framework, as illustrated earlier in Figure 1-2. Next, determine which elements you can purchase and which components you need to build. Then, looking at the product requirements, you need to design the database schema—one that can capture the necessary application data with flexibility for expansion. Finally, you need to address a component-level design, identifying which classes you will need to create, what their public and private

interfaces will be, what inheritance relationship they will have to each other, and what data members will be private and public. When you have clearly defined the application from a high-level picture down to a database schema and class definition, it is time for development.

Developing the Application

Application development should be one of the shortest stages of the entire process. This is largely because the design phase should clearly spell out everything that needs to be coded. Also, the application framework should provide enough templates that trivial and repetitive development tasks are minimized. The requirements have already been captured and translated into a complete design. All that is necessary is to translate design drawings and class definitions into code. In addition to application development, you also need to create the installation and configuration code. Although the .NET Framework supports multiple development languages to coexist within a single solution, it is ideal to select one language and work with it exclusively. This helps build internal skills with a specific language and keeps maintenance costs lower.

Testing the Application

Like the design step, the testing step should be one of the longest in the development process. Ensuring application quality should come first with all application development. If application development begins slipping past schedule, it is a smarter strategy to drop a handful of product requirements than to trim back testing time. Customers have repeatedly communicated that they can live with a product that falls short of expectations much more easily than a product filled with software bugs.

During this step, you should also develop product documentation, such as user and administrative guides. In some cases, documentation occurs during the development step. However, it is far too common for functional changes, labeling changes, or even flow changes to occur, which forces rewrites of the product documentation. Also, keeping the documentation effort in parallel with the testing effort helps ensure a longer testing stage. Ideally, you should write product documentation with a tool that easily generates online help as well as printed documentation. Some enterprise providers attempt to produce only online help and then find that corporate IT managers insist on printed documentation.

Deploying the Application

The final step in the enterprise application development process is packaging the application and labeling the CD. You need to include all printed documentation, including license agreements. An organized release package often sends a strong message to enterprise customers. This message expresses that the application has been well organized and professionally developed, stemming from a solid enterprise application design.

Understanding Enterprise Application Design

As mentioned earlier, the application design step is essential to the success of the application's development. This step requires creating a high-level framework, which includes determining what components will be bought vs. built, creating a database schema, and creating a component-level design.

Understanding Multitier Architectures

The most common enterprise application design revolves around a multitier architecture. A *multitier* architecture distributes core functionality across different servers to maximize overall application performance. The four major components of any application include presentation, workflow, business logic, and data access. The presentation layer captures user input and displays the functional output. The workflow layer typically manages the flow of execution from one block of code to another. The business logic implements specific functionality that acts upon input data. The data access keeps user input persistent by reading from and writing to a database.

As Figure 1-3 illustrates, classical desktop applications typically package all four application layers into a single binary file, the .exe file. A two-tier application separates the data access layer to a different process. This allows for greater scalability because you can introduce an additional server to offload work. A three-tier application separates the business logic layer, as well as the data access layer, to different processes. And in an n-tier application, all major application layers are separated into their own processes for the greatest scalability.

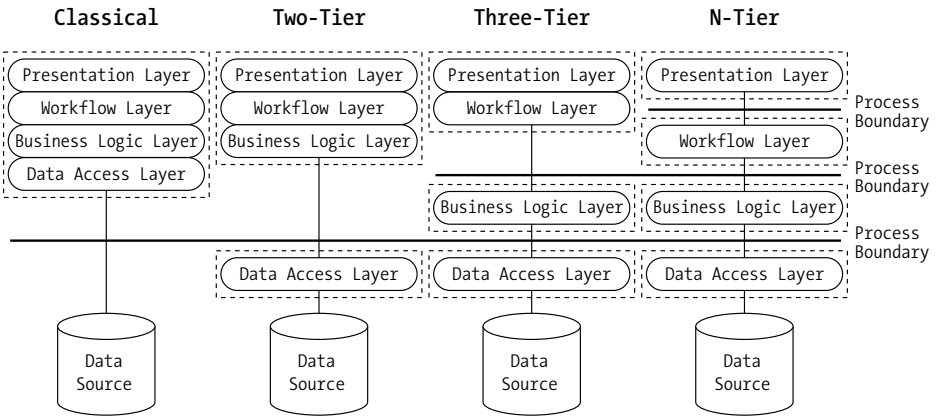

Figure 1-3. Multitier architectures distribute work to different servers.

Determining Buy vs. Build

Most developers would enjoy the opportunity to develop an enterprise application completely from the ground up. However, this may not be very cost effective. It terms of dollars, it is likely to cost more in developing and testing than to purchase a third-party solution. In many cases, such as database services, process automation, messaging, and directory services, the decision to leverage existing software solutions should be easy.

Licensing Microsoft solutions such as SQL Server, BizTalk Server, MSMQ, and Active Directory can be more cost effective than building equivalent systems from scratch. However, other solutions, such as document management, are not as obvious because you can develop a lightweight version relatively quickly. Figure 1-4 illustrates an enterprise architecture with the purchased technologies shaded. The blocks that are not shaded need to be developed internally. You need to investigate early on and decide whether to buy or build specific application components. This includes evaluating the effort required to integrate the component as well as the features it supports.

Defining the User Interface

The user interface definition is an important starting point in the application design process and results in the creation of the user interface definition document. Two levels of detail make up this document: the user interface mockup and the user interface details.

The user interface mockup can vary from hand-drawn representations of Windows forms and Web pages to screen captures of a semifunctional prototype. Its primary purpose is to represent how the finished application will look and broadly define what functions the application will perform.

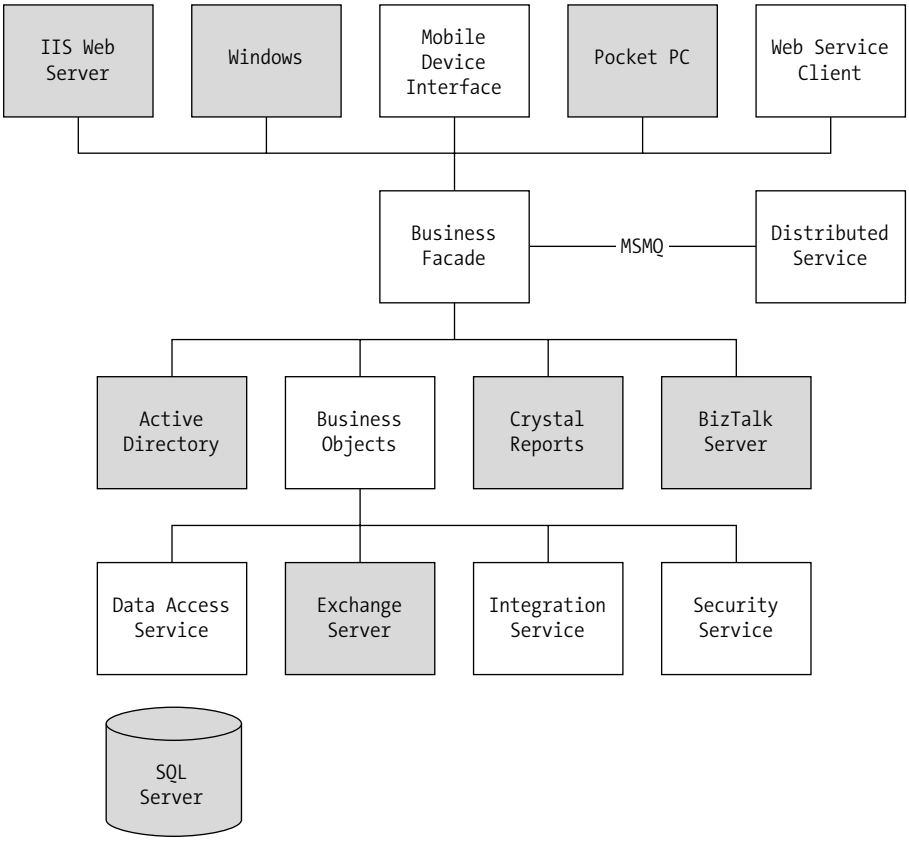

Figure 1-4. Architectural diagram with the purchased components shaded

The user interface details provide the next level of technical information. This specifies exactly what user controls will be displayed within each Windows form or Web page. This specification also includes additional details for each control, such as what events each control responds to and what business functionality is executed. Finally, the specification indicates data formatting and data validation. Data formatting outlines how a label or textbox should represent data, such as percentages vs. currency. Data validation outlines the acceptable ranges for textbox controls, such as restricting letters from a number entry field.

Creating a Database Schema

The database schema is one of the most important elements of the enterprise application design. The difference between a good schema and a bad one can determine the scalability and performance of the application. It is important to structure the schema for extensibility because new features will certainly work

their way into the product. However, abstracting data into too many tables will result in poor performance because all the tables will then need to be joined. Figure 1-5 illustrates the IssueTracker data schema.

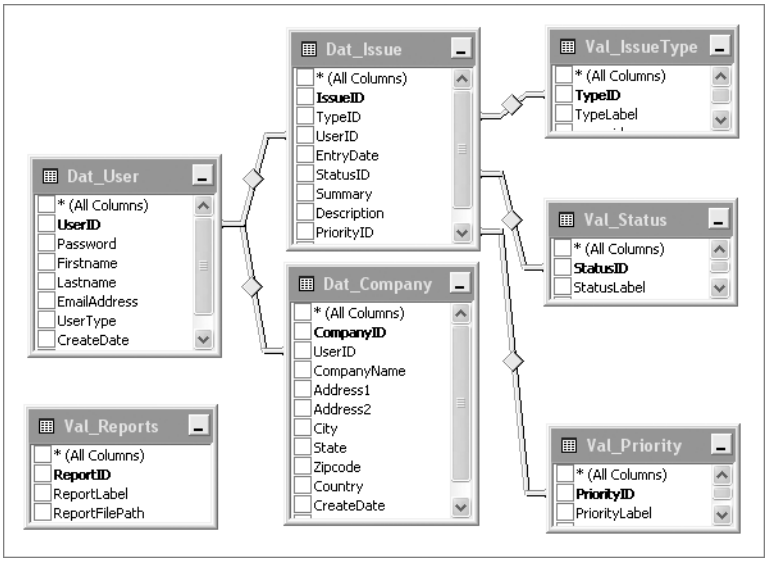

Figure 1-5. The IssueTracker database schema

The person responsible for the schema design needs to understand what the application does and how it will access the data. This includes knowing which tables will be accessed most and how often data will be accessed. During development, the schema designer will also be responsible for creating primary keys, indexes, and stored procedures. There will always be minor changes to the database schema; however, you should complete the overall schema design before the component-level design begins.

Creating the Component-Level Design

The component-level design is where most of the real design time goes. Starting with the high-level architecture, you need to expand each block into detailed class definitions. Each class definition will have private and public data as well as private and public methods. Implementing the principles of object-oriented design will improve overall reusability and extensibility.

Establishing an Application Development Framework

An application development framework is very important to the success of any large-scale development project. By nature, enterprise application development requires multiple full-time developers working on various implementation tasks. Ensuring quality and consistency across the entire application can be nearly impossible without a well-defined and well-structured application framework.

The application development framework outlined in this book relies heavily upon clear design documentation and foundation design patterns. The design documentation includes those items previously described: the user interface definition, a database schema, and a component-level design. The foundation design patterns include uniform data access, interface definitions for business objects, templates for differently behaving forms, style sheets, and reusable user controls. Pulled together, an application framework can significantly accelerate the speed of development, ensure user interface consistency, reduce the potential for bugs, enhance overall security, and apply best practices for functionality reuse.

The most difficult aspect of establishing an application development framework is determining how much functionality should go into it and how much should be implemented in application-specific code. In general, you should probably add any functionality to the framework that you expect to be used in at least 80 percent of the applications. If there is no clear use of a specific technology beyond a specific project, then you should implement that functionality only for that application. Otherwise, time will tend to clutter the application framework and make it less useful and less robust. Determining whether specific functionality should be implemented within the application framework or within a specific project will be a constant challenge for enterprise developers, and it deserves careful thought and planning.

The application development framework in this book is flexible enough to be applied to multiple domains and is composed of design patterns reflected across all application tiers. As Figure 1-6 illustrates, the framework begins with a collection of stored procedures optimized for quick and secure data access. These stored procedures interact with a middle tier that manages a DataSet for cached validation data as well as business objects that manage in-memory representations of interactive data. These business objects implement framework interfaces that define how the data is interacted with, especially for data storage and retrieval, data value validation, and help descriptions. These business objects leverage data binding to populate Windows forms or Web pages that render the application user interface to the user. Windows forms inherit from predefined form classes, and Web pages are laid out based on predefined Hypertext Markup Language (HTML) page layouts. Style sheets also provide predefined visual characteristics, such as fonts and colors, for Web pages. Finally, a collection of predefined user controls provide common functionality, such as page headers and footers, role-based menus, and common event handling, such as save and cancel functions.

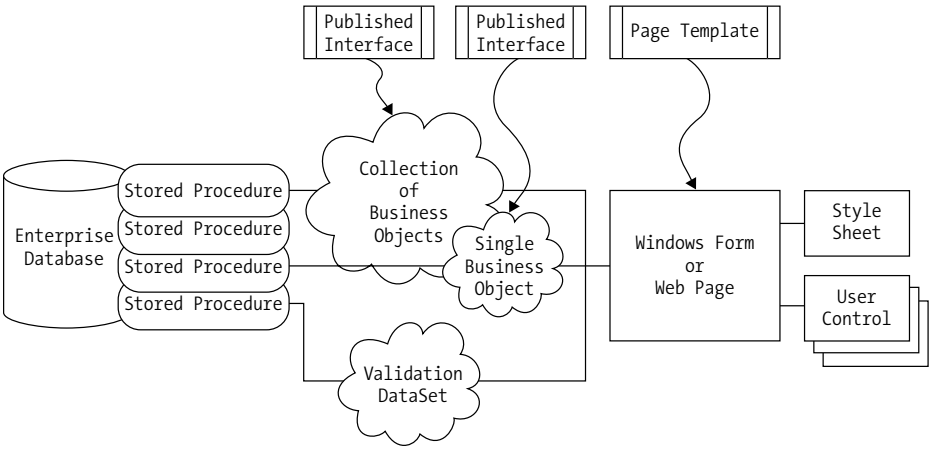

Figure 1-6. A flexible application framework composed of interfaces and templates

Each of the following chapters focuses on and develops a specific part of the framework as it fits into the IssueTracker application. You can leverage the best-practice design patterns from these examples and build a variety of enterprise applications including supply-chain solutions, e-commerce solutions, and enterprise resource planning systems.

Getting Started with Visual Studio .NET

Three different versions of Visual Studio .NET 2003 are available: Enterprise Architect, Enterprise Developer, and Professional. The examples and templates in this book require Visual Studio .NET Enterprise Architect or Enterprise Developer. Most concepts and examples work with the Professional version but will be incomplete.

Exploring the New Solution Options

In Visual Studio .NET, application source code is grouped into two entities: projects and solutions. *Projects* implement a specific business function, and *solutions* are collections of projects (similar to the workspace concept in Visual Studio 6.0). As each chapter progresses, you will implement a small piece of the overall enterprise application. The source code for each chapter will be packaged as projects within the IssueTracker solution.

To begin, start Visual Studio .NET 2003 and click the New Project button in the Projects tab. The New Project dialog box presents several templates related to creating an enterprise application. Expand the Other Projects tree node and select the Enterprise Template Projects node (see Figure 1-7).

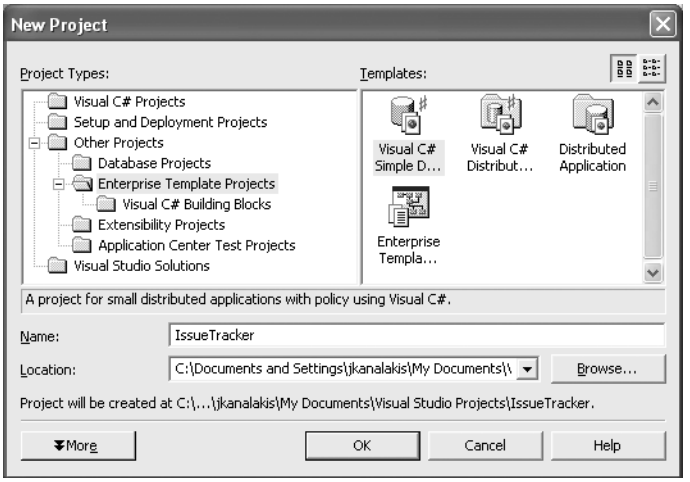

Figure 1-7. The New Project dialog box

The following project templates become available:

Visual C# Distributed Application: This template creates a C# version of the standard Distributed Application solution. This template consists of seven layers, allowing for multiple projects of different programming languages within each layer of the application. This creates either a Windows application or a Web application.

Visual C# Simple Distributed Application: This template is a simpler version of the standard Distributed Application template. This template creates applications that have a simpler structure than the Visual C# Distributed Application template. Each application layer has only a single language project. This creates both a Windows application and a Web application.

Visual Basic Distributed Application: This template is a Visual Basic version of the standard Distributed Application template. This template consists of seven layers, allowing for multiple projects of different programming languages within each area of the application. It includes both a project that produces a Windows user interface and a project that produces a Web user interface.

Visual Basic Simple Distributed Application: This template is a simpler version of the standard Distributed Application template. This template creates applications that have a simpler structure than the Visual Basic Distributed Application template. Each application layer has only a single language project. This template includes a project that produces a Windows user interface and a project that produces a Web user interface.

Distributed Application: This template provides the same structure as the C# and Visual Basic Distributed Application templates but without the initial language projects. This template creates large-scale, mixed-language applications by adding building blocks (projects) to each layer. Conversely, when you select the C# or Visual Basic Distributed Application template, each layer already includes a C# or Visual Basic project.

Enterprise Templates Project: This is an empty template that does not include any language projects or associated policy. Architects can use it to create their own templates.

As mentioned, each enterprise project is composed of multiple layers. Each layer is implemented as a separate project that abstracts a specific enterprise function. These layers include the following:

Business Facade: This project is often used to provide a consistent interface to the underlying business objects and to isolate the client from changes to the underlying business logic. It lives either between the client and the business logic or between the Web service projects and the business logic layers. You abstract application functionality into high-level methods exposed within this layer.

Business Rules: This project contains the business objects themselves and the rules applied to them. They implement the business entities or objects of the system. The business rules of the system are coded within these objects; however, some of the business rules might actually be coded in the database within stored procedures and triggers.

Data Access: This project performs the function of retrieving data from and sending data to the database. It accomplishes this using ADO.NET data access objects and SQL Server stored procedures.

System Frameworks: This project manages the parts of the application that provide system service, system infrastructure functionality, or other shared functionality. This functionality might not be specific to any given application.

ASP.NET Web Service: The Web Service project provides public Web interfaces that are accessible to Web service clients. It can be present independent of the type of user interface (such as a Web client or a Windows client), if any, used in your application. It is one of the ways of remoting application servers. This project is similar to the Business Facade layer except that other Web service clients access its methods, which means it is not limited to being accessed by other parts of the application.

WebUI: This project implements a Web application that renders Web pages to present data, capture user input, perform data validation, provide task guidance to users, send user input to the Business Facade project, and receive results from the Business Facade project.

WinUI: This project implements a Windows form application that renders a user interface to present data, capture user input, perform data validation, provide task guidance to users, send user input to the Business Facade project, and receive results from the Business Facade project.

Implementing the Solution

Select Enterprise Template Projects from the Project Types list box. Next, select Visual C# Simple Distributed Application from the Templates list. Finally, enter *IssueTracker* in the Name box (as shown previously in Figure 1-7), and click the OK button. Visual Studio .NET will prompt you for a Uniform Resource Locator (URL) that points to an Internet Information Server (IIS) location, as illustrated in Figure 1-8. In most cases, the default localhost setting is fine.

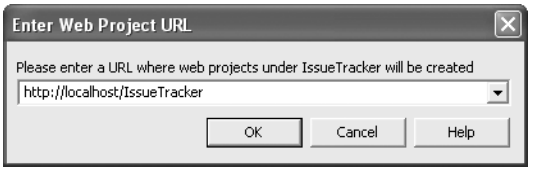

Figure 1-8. Visual Studio .NET can store Web projects on any accessible IIS Web server.

Visual Studio .NET will quickly generate the new solution in different locations. The bulk of the source code will appear within the following location:

```
\My Documents\Visual Studio Projects\IssueTracker
```

The generated Web service project code will appear within the following location:

```
\inetpub\wwwroot\IssueTracker\IssueTracker_WebService
```

The generated Web application project code will appear within the following location:

```
\inetpub\wwwroot\IssueTracker\IssueTracker_WebUI
```

The IssueTracker solution contains seven projects that implement the business facade, business rules, data access, system framework access, Web user interface, and desktop user interface. As the development of the IssueTracker application continues, new projects will appear within the same solution. Visual Studio .NET displays each project in the Solution Explorer, as illustrated in Figure 1-9.

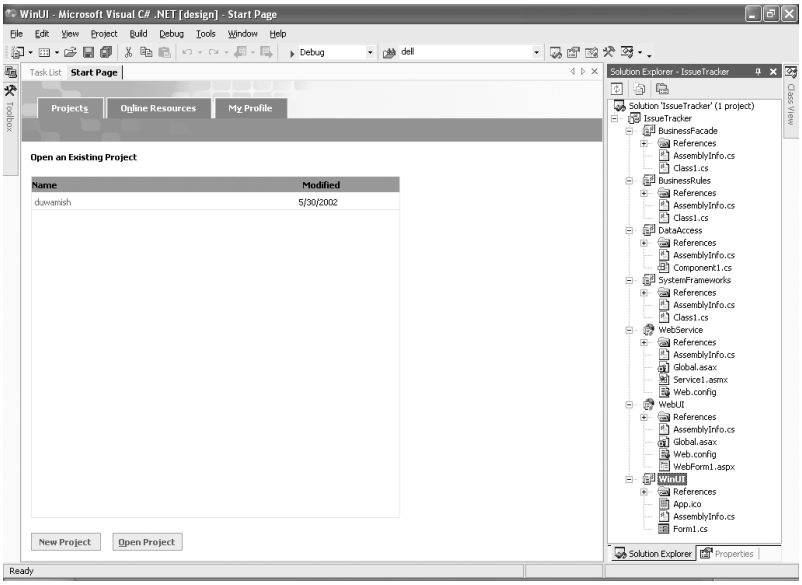

Figure 1-9. The empty enterprise solution in Visual Studio .NET

Once the solution has been created, development of the IssueTracker enterprise application can begin. Over the next few chapters, you will fill in the individual projects under this solution that handle database access, directory services accessibility, messaging, business process automation, e-mail support, and the user interface. Each project will be complete with source code and a test utility.

Summary

As long as there are large businesses, there will always be a need for enterprise applications. This chapter started by defining an enterprise application. Next it described some of the technologies involved in building an enterprise application, such as database access, directory services, messaging services, mail services, process automation, reporting, and user interfacing. The chapter also explained how enterprise applications are designed and how you can use Microsoft's .NET Framework to support rapid development. Finally, the chapter introduced the IssueTracker enterprise application that you will incrementally develop with each chapter of this book.

In the next chapter, you will be exposed to the data services of a .NET enterprise application. You will learn how a structured data access framework can accelerate the development time of data-centric applications. You will also see how you can use certain .NET characteristics—such as properties, collections, and reflection—to implement business objects and an intelligent business object manager.