Enterprise Application Integration

A Wiley Tech Brief

William A. Ruh
Francis X. Maginnis
William J. Brown

Wiley Computer Publishing

John Wiley & Sons, Inc.

NEW YORK • CHICHESTER • WEINHEIM • BRISBANE • SINGAPORE • TORONTO

Publisher: Robert Ipsen
Editor: Theresa Hudson
Developmental Editor: Kathryn A. Malm
Managing Editor: Angela Smith
Text Design & Composition: Publishers' Design and Production Services, Inc.

Published by John Wiley & Sons, Inc.

ISBN 0-7879-5850-6

This title is also available in print as ISBN 0-471-37641-8 (pbk. : alk. paper)

Wiley Tech Brief Series

Contents

Introduction

Information technology has become essential to the successful operation of any enterprise. Every aspect of a business relies on some form of automation. In the past automation was custom developed. Recently the use of packaged applications has reduced the amount of development. Most of these packaged applications exist as stovepipes, applications that are self-contained. The requirements for next-generation systems mandate the integration of these stovepipes with new forms of business logic.

The term *Enterprise Application Integration* (EAI) has recently entered the jargon of the computer industry. It was created by industry analysts to help information technology (IT) organizations understand the emergence of a type of software that eases the pains of integration. This pain has increased as organizations realize they increasingly need to interconnect disparate systems to meet the needs of the business. In fact, most organizations have come to realize that they need to connect their entire enterprise.

The implications of applying EAI to the enterprise and its inherent complexity have left IT practitioners and managers struggling to effectively apply it across their systems. Organizations such as the Gartner Group have reported that 35 to –60 percent of an organization's information technology resources are spent on integration. This book is intended to give the reader a framework for effectively architecting, designing, and developing systems based on EAI technology.

How This Book Will Help

After reading this book, you will have an understanding of EAI in its broadest sense. You will have enough practical knowledge to organize an EAI-based project. This includes identifying when to apply EAI technology, selecting appropriate technology, architecting a solution, and then applying it successfully to the problem.

EAI products' use by early adopters has resulted in lessons that are presented in this book. These lessons include how to identify your integration problem, how to structure and architect an EAI solution, how to select the right technology, and how to use methods and processes focused on integration. We have each been involved in the application of EAI with a number of early adopters, using a wide range of products and technology. As a result of these experiences, we do not have any bias toward any product or technology. Each of the major technologies that underpin EAI is explored in the book, and guidance onwhen to apply the technology if offered.

We believe that architecture and design are the most critical elements in successful application of EAI. Therefore, a significant amount of this book is dedicated to these topics. Patterns of integration are described to help the reader understand what problems are related to EAI. In constructing an EAI architecture a set of common building blocks will always exist. These building blocks are discussed prior to the technology discussions. After technology, the topics of architecture and methodology are described in detail.

Our intention is for the book to be read and then used as a reference. The chapters are organized around specific topics that can be referenced to help with specific aspects of a project. Two appendices are included. Appendix A has a list of companies that have products related to EAI. The decision to include this type of appendix is difficult due to the transitory nature of the information. In addition, it is difficult to include a complete list. Appendix B contains a list of EAI-related Internet resources.

Organization

The book is organized into four different sections. The first section focuses on the business case for EAI and contains Chapters 1 and 2. Chapter 1, "The Business Drivers of EAI," introduces the business applications that require EAI. Chapter 2, "Types of Integration," takes you through the nature of integration. After reading this chapter, you will understand how to organize your thinking with regard to a specific integration problem.

The second section focuses on EAI architectures. Chapters 3 through 6 cover this material. Chapter 3, "Basic Building Blocks," introduces the concept of the EAI architecture and sets a framework for an organization to develop its own architecture. Chapters 4 through 6 deal with the three major middleware technologies on which an EAI architecture is constructed.

Chapter 4, "Messaging Architecture and Solutions," examines the various forms messaging mechanisms, message oriented middleware, and messaging products. Chapter 5, "Object Architecture and Solutions," focuses on distributed object technology, component architectures, and some of the more popular and effective implementation technologies such as OMA, IBM Component Broker, Microsoft DNA 2000, and Java/EJB. Chapter 6, "Transaction Architecture and Solutions," covers transactions and the mechanisms required to support full transactional integrity within and across applications.

The third section examines the application of EAI to the enterprise. It tackles the construction of an EAI architecture as well as the process for applying it to building systems. These topics are covered in Chapters 7 and 8, respectively.

Chapter 7, "Enterprise Architecture," focuses on how to produce an enterprise application integration architecture, including making legacy, stovepipe, COTS, and modern technology applications interoperable. Chapter 8, "Effective EAI Development: The SAIM Methodology," discusses the needs and requirements for and principles of an EAI methodology with insight into an existing, proven approach.

The fourth section summarizes the book. It contains Chapter 9, "Building Future EAI Business Solutions," which ties together the vast knowledge that is contained in the first three sections and offers practical advice to implementing EAI from scratch.

Appendix A, "EAI Related Vendors," provides a list of resources for the reader to expand his or her research and as a starting point for identifying EAI products to solve user problems. Appendix B, "EAI Related Internet Resources," is a short list of online sources that cover EAI concepts and implementations.

The book is intended to bring the reader through a logical progression of topics that start at a very high level and slowly go deeper into the topic of EAI. It is an introductory text to the topic. Each chapter, especially those in sections 2 and 3, could be a topic for a book. In the case of Chapters 4 through 6 there exist several books on each topic.

Why This Book Is Different

This book has three unique features that make it different from anything else on the market. These features are as follows:

- A broader focus on the underlying technology on which to architect using EAI
- An emphasis on EAI architecture based on experience, guidance, and frameworks
- The description of a process for applying EAI in an organization

Most practitioners of EAI focus on a single underlining middleware. We give a broader perspective that encompasses the three major middleware categories on which to build an EAI architecture. No other book captures this wide range of choices.

EAI is often viewed as a specific product, not as an architecture. We have strong views on the topic of architecture. As a result, this book emphasizes the need for architecture, the basic building blocks, and how to construct as well as how to apply an architecture. Most other books focus exclusively on the technology.

Finally, this book provides guidance on the application of the technology to the development of systems. Process and technology go hand-in-hand for building successful systems; however, process is often overlooked. We have included information that is vital to applying EAI technology.

Final Thoughts

You are about to embark on a learning experience that we hope helps to organize your thinking on this topic while being immediately useful to your daily activities. Although integration has been around for a long time it continues to change and mutate in its application to information systems. We hope you will see how to apply EAI technology successfully to your IT problems

CHAPTER 1

The Business Drivers of EAI

Key Facts

- EAI—combining the functionality of an enterprise's existing applications, commercial packaged applications, and new code by means of common *middleware*—is emerging as a key IT technology.
- EAI is crucial to enabling critical new solutions that provide competitive value, including improved customer and supply-chain relationships, streamlined internal processes, and reduced time to market.
- EAI allows the enterprise to leverage its investment in legacy and packaged applications.
- EAI is crucial to seizing the opportunities presented by the Web.
- To effectively employ EAI, enterprises must overcome the barriers of chaotic architectures, managing complexity, training their staff, and securing their applications.

IT organizations have been struggling for some time to find better ways to make their applications work together. Even though these organizations had constructed many ad hoc solutions to application integration, the term *Enterprise Application Integration*, or *EAI*, was never heard in information technology circles. Today, more and more firms identify their products as "EAI products," consultants are establishing "EAI practices," and systems integrators are setting up teams devoted to EAI.

What is behind this phenomenon? Why has the concept of EAI suddenly burst upon the IT community with such force? What are the implications for your enterprise? What do you need to do to exploit EAI effectively in your enterprise? This book answers these questions. We discuss the technologies that are important to EAI, and we show how they make it easier for you to integrate existing applications and commercial packaged solutions into new composite applications. We give you guidelines about how you can apply EAI more effectively. But before plunging into a discussion of the technology, let's begin by considering two basic questions: What is EAI? And how does it apply to your business?

EAI

Enterprise Application Integration (EAI) is the creation of new strategic business solutions by combining the functionality of an enterprise's existing applications, commercial packaged applications, and new code using a common *middleware*.

EAI

Enterprise Application Integration is the creation of business solutions by combining applications using common middleware.

Middleware refers to technology that provides application-independent services that mediate between applications. The term can also refer to the software products that implement these services. Before the advent of EAI middleware, enterprises often attempted to integrate applications by *ad hoc* means, but they were quickly overwhelmed by the complexity and scale of the effort. Middleware-enabled EAI reduces the complexity of integrating applications by providing the following:

- Mechanisms by which applications can package functionality so that their capabilities are accessible as services to other applications. For example, a banking application could support services to create accounts or to transfer funds.
- Mechanisms by which applications can share information with other applications. For example, numerous applications across an enterprise can share information related to customers. Because the information to be shared is rarely organized and formatted the same way in independently developed applications, EAI middleware

often provides capabilities to translate and convert data as it flows from one application to another.

- Mechanisms by which applications can coordinate business processes. For example, EAI middleware may support a capability to manage defined business workflows or for one application to declare an event to other applications that has a well-defined meaning in the context of a business process.

middleware

Middleware is application-independent software that provides services that mediate between applications.

EAI is attractive for developing new applications because few changes to existing legacy or packaged applications are required and because there is little or no need for extensive programming or custom interfaces. EAI leverages existing application programming interfaces (APIs) and databases. In extreme cases, where no APIs exist, EAI may access an application's functionality by mimicking normal users through the application's user interface, using screen-scraping technology. Screen scraping is the copying of data from specific locations on character-based screens that are displayed on an end-user computer.

The ultimate goal of EAI is to allow an organization to integrate diverse applications quickly and easily. By employing EAI effectively, an enterprise can leverage its existing assets to provide new products and services, to improve its relationships with customers, suppliers, and other stakeholders and to streamline its operations. EAI also allows the enterprise to greatly simplify interactions among enterprise applications by adopting a standard approach to integration, replacing hundreds or thousands of ad hoc integration designs. Moreover, once an EAI infrastructure has been put in place, new EAI-based applications can usually go online more quickly than traditionally developed applications because an enhanced technical infrastructure exists on which to base future development. By enabling all these new capabilities, EAI can help an enterprise create competitive advantage.

EAI Enables Critical New Solutions

The ultimate rationale for EAI is to enable critical new solutions for the enterprise. EAI does the following:

- Enables you to improve relationships with customers
- Supports strengthened supply chains

- Helps to streamline internal processes
- Helps you bring new applications online more quickly

Let's take a look at each of these solutions.

Improving Customer Relationships

EAI helps an enterprise achieve a 360-degree view of its relationships with its customers (see Figure 1.1). Each customer perceives the enterprise as a whole, rather than as a collection of departments or lines of business. Customers expect to be recognized by the enterprises with which they do business. When dealing with one division, they don't want to be required to provide information that they've already supplied to another division. When calling to resolve a service problem, they become irritated if they have to repeat what they told one department when they are transferred to another department. Frequent customers want to perceive that their loyalty is valued.

The enterprise wants to be able to take advantage of all that it knows about a customer. Knowing the products that a customer previously has bought can create opportunities for selling other products or additional services related to previous purchases. And it should not matter if the customer sometimes interacts with the enterprise via the Web, sometimes via a call center, and sometimes in person—all the information should be integrated.

Achieving improved customer relationships demands application integration. The 360-degree view of customer relationships requires that all information relevant to a customer be available in an integrated form, even though that information may be

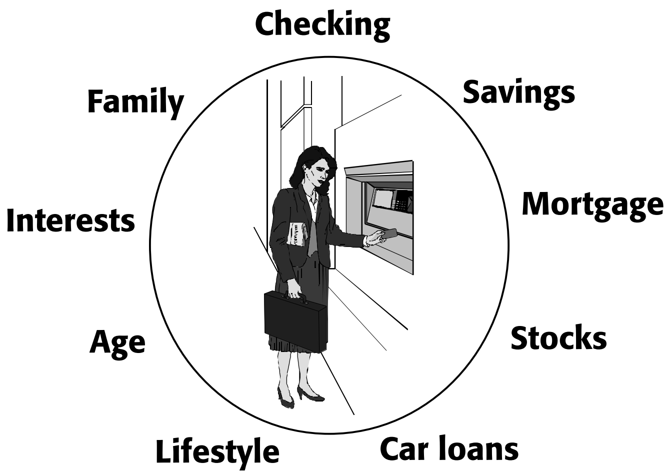


Figure 1.1 EAI enables a 360-degree view of the customer.

scattered in numerous "stovepipe" applications developed to support various lines of business.

stovepipe applications

"Stovepipe applications" are applications that were developed to support a specific business function, such as order entry, or an organizational unit, such as the Accounting department. In a diagram of the enterprise's systems, they look like isolated vertical structures, hence "stovepipes." Because they were not designed to interoperate with other applications, stovepipe systems are a significant challenge to integrate.

Improving Supply-Chain Relationships

In addition to improving relationships with customers, enterprises also want to improve relationships with supply-chain partners and other outside organizations. Opportunities for electronic information exchange are growing rapidly. By sharing such information as inventory levels, enterprises can coordinate their activities more effectively. Partners can also leverage new technologies to create new services. By establishing electronic links with its shipping partners, for example, a retailer can offer enhanced order status tracking.

Improving customer relationships and attaining higher levels of integration with supply-chain partners introduce new security concerns. If the Internet is used as a communications channel, rather than dial-up or dedicated lines, an enterprise must take steps to ensure the confidentiality of information flow. Controls must be put in place to ensure that partners can access only the information that they are supposed to see. An important aspect of access control is to ensure that a partner cannot see information relating to dealings with any other partners.

Integration requirements for interactions with partners are similar to those for interaction with customers. New applications often will be required to integrate several existing stovepipe applications. The ability to support information exchange utilizing technology such as the *Extensible Markup Language* (*XML*) is becoming a key factor in automated business-to-business integration. A variety of front-end channels may be needed to provide differing levels of functionality to differing relationships. Businesses may want to mix and match older technology such as EDI for automated exchanges with newer Web-centric technologies. These technologies can include XML for automated exchanges and Web interfaces for smaller partners. Such mixing and matching can be part of an IT evolution strategy.

XML

XML stands for Extensible Markup Language. It is the emerging standard for communicating information between applications, both in business-to-business and in intra-enterprise contexts. XML facilitates the definition of the content of the information flowing between applications. It promises to greatly reduce the effort required to define and maintain communicating applications.

Improving Internal Processes

Improving an enterprise's internal processes can be as important a driver for EAI as improving relationships with customers or supply-chain partners (see Figure 1.2). EAI techniques can be used to simplify information flow between departments and divisions of the enterprise. In many organizations, providing integrated information for decision support is a key reason for undertaking EAI projects. EAI technology can help to populate data warehouses that can be used to analyze market trends, evaluate the effectiveness of a business initiative, and assess the performance of organizations within the enterprise. EAI facilitates the construction of a data warehouse by mediating the flow of information from stovepipe applications to the common warehouse and by supporting the conversion of data from various applications' formats to a common format.

Just as customer self-service is an important class of applications in customer relations, employee self-service is important in improving business processes. Web-enabled interfaces may provide employees with better access to the information they need to do their jobs. Employee self-service Web sites for benefits administration and other HR functions are becoming increasingly popular.

EAI can be used to help eliminate manual steps in business processes and to avoid redundant entry of data. Such applications of EAI often employ a workflow-automation tool to bridge between the applications that are being integrated.

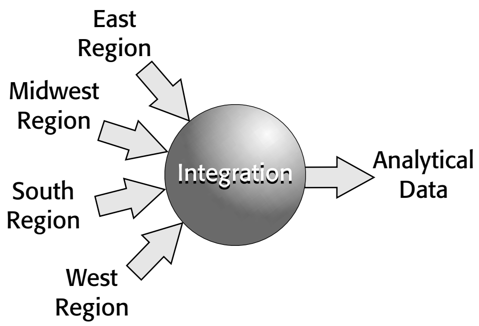


Figure 1.2 Business processes can be improved using EAI technology.

Reducing Time to Market

Let's face it. IT organizations' track record in developing mission-critical applications on time and on budget hasn't been very great. Close to 85 percent of software development projects are never successfully completed, 58 percent of large systems projects do not come in on budget, 63 percent do not come in on schedule, and 58 percent of companies report a success rate below 50 percent [Standish Group 1995].*

Given this sad state of affairs, one of the greatest benefits of EAI technology is reducing the time to market of new applications. EAI contributes to faster rollout in several ways. First, EAI leverages the capabilities of existing applications. Often the existing code does its job well and already has been debugged. What is needed is to make existing functionality accessible to new front-end channels like the Web or to new composite applications. Once an EAI infrastructure is in place, time to market can be reduced in the most straightforward way: There's less code to write. It must be recognized, though, that initially putting the EAI architecture in place can itself be a significant effort. The enterprise must be willing to undertake the cost of establishing EAI to reap the benefits in the long term.

A second advantage is that EAI, through the use of middleware, allows developers to avoid many of the trickiest and most error-prone aspects of IT, such as integrating functionality that is hosted on diverse hardware and operating system platforms. Again, EAI reduces the amount of code to be written for an application, but perhaps more importantly it allows developers to concentrate on the business aspects of the application, not the "plumbing." There are maintenance benefits as well. Only 30 percent of the code in mission-critical applications is business logic (that is, the code that actually performs business computations); the rest is infrastructure—much of it in areas subsumed by commercial EAI software. A recent study reports that maintenance for in-house-developed infrastructure software can cost 15 to 25 percent of the original development cost each year [Standish Group 1997]. Replacing home-brewed infrastructure with EAI middleware can thus lead to significant maintenance savings, even after allowing for the software vendor's maintenance charges.

The Value of Legacy and Packaged Applications

An enterprise's legacy applications can be its crown jewels. Over time, these applications have become the repositories of the enterprise's corporate knowledge. Business rules encoded in legacy applications (for example, under what circumstances to give a discount) are often documented nowhere else. As a result, it can be extremely difficult to replicate or reengineer the legacy applications. Being able to utilize them "in place" to support new requirements is a tremendous advantage.

*The Standish Group International, Inc. (www.standishgroup.com) is a market research and advisory firm specializing in mission-critical software and electronic commerce.

Legacy applications that run on mainframe computers have several additional advantages. They can take advantage of the ultra-high reliability of mainframe systems, which in many cases will suffer just a few seconds of downtime per year. Mainframe systems also can support extremely high transaction rates and numbers of users. Keeping legacy applications on mainframes can help spread the high fixed costs of that computing across a large number of applications.

Another reason why leveraging existing applications is valuable is that most enterprises have just invested huge sums to deal with the Year 2000 (Y2K) problem—the inability of many legacy applications to correctly interpret two-digit representations of the year beyond the year 2000. Having made this investment in their legacy code, enterprises have a strong incentive to keep that code around and treat it as a valuable asset. Moreover, many enterprises are finding that, as a side effect of their Y2K correction, they now have a much better understanding, and better documentation, of their legacy applications' design than they did before.

Packaged applications, such as Enterprise Resource Planning (ERP) packages from SAP, Peoplesoft, Oracle, and other vendors, also figure prominently in the EAI landscape. Utilization of packaged applications has grown significantly. These packages are attractive in that they provide proven solutions to common business needs. Development and maintenance become the problem of the vendor, not the enterprise's IT organization. This allows the in-house IT staff to focus on problems that are specific to the business and to create capabilities that can become differentiators. Packaged products from a single vendor, however, will never meet all of an enterprise's needs; hence they must be integrated both with enterprise-specific custom applications and with other packaged solutions. For example, it is not uncommon for an enterprise to employ a packaged product from one vendor for financial functions such as general ledger, accounts payable, and accounts receivable, and another product for human resources functions like payroll and benefits. Package vendors have begun to recognize this need, and they are beginning to incorporate EAI capabilities into their new releases. Indeed, through partnerships many packaged application vendors have ensured that their products are EAI ready.

The Web

The Web is a key factor in the emergence of EAI as an important technology. We have already mentioned several examples of Web-based applications that depend on EAI. Developing Web applications isn't the only impetus for application integration, and not all EAI projects are Web-based. But the emergence of the Web, and its underlying Internet technologies, are fundamentally important to EAI and to the changes in the IT environment of which EAI is a part. "Revolutions" seem to happen about once a year in IT, but there are good reasons for believing that the emergence of the Web is different—a real revolution (see Figure 1.3). How many other IT technologies have so invaded the public consciousness? Businesses from the National Football League to local florist shops are urging customers to check out their Web pages. Virtually every advertisement on commercial television—which is certainly not aimed at only computer aficionados—

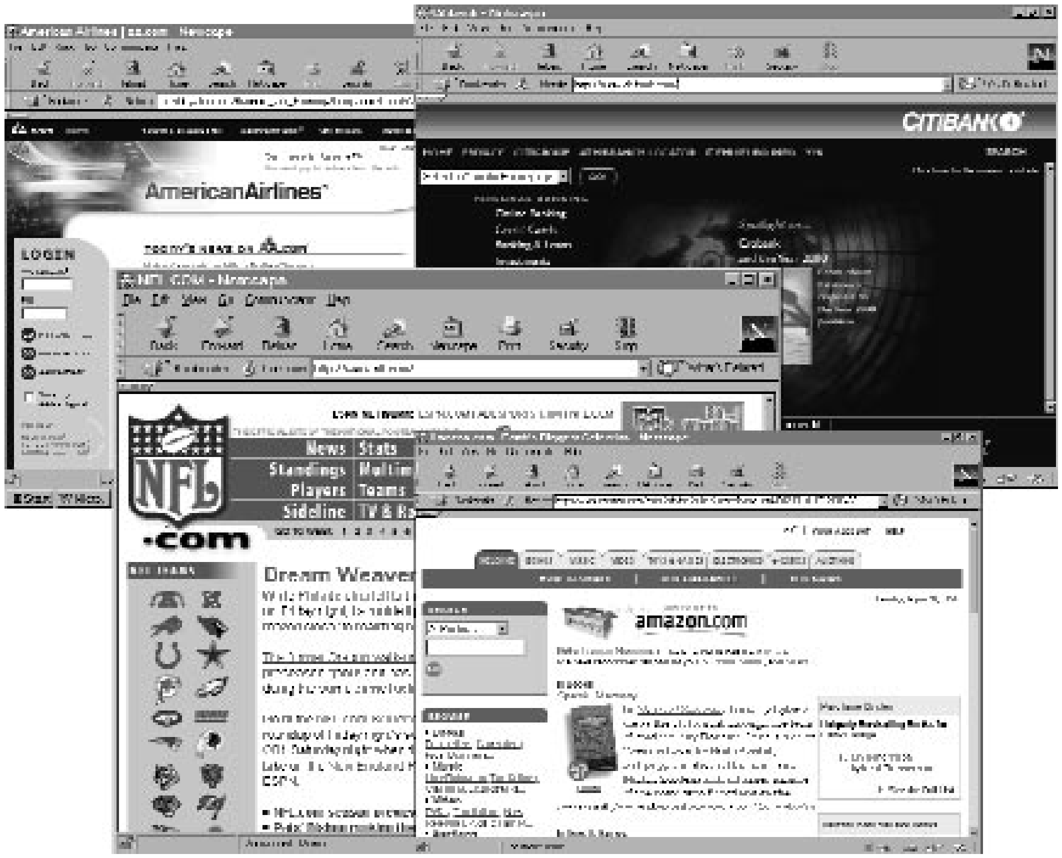


Figure 1.3 The Web is pervasive.

has at the bottom "www.OurCompany.com." Indeed, "dot-com" has entered the language: as a noun referring to the new class of fast-growing Web-based businesses.

dot-com

dot-com refers to enterprises that do a significant portion of their business over the Internet. "Pure play" dot-coms do business only over the Internet: They have no physical retail locations.

Comparing the impact of the Web with the inventions of printing in the sixteenth century or telegraphy in the nineteenth century is not far fetched. In fact, all three of these technologies—printing, telegraphy, and the Web—have the common characteristic of causing a fundamental change in the ease with which people could communicate with one another. Each technology enabled more people to produce information, made it easier for them to disseminate that information, and made the information available to a much larger audience, all at significantly less cost than with earlier technologies. Barriers that had seemed insurmountable have now been scaled. Anyone can easily express a point of view or sell a product to a global audience on the Web.

From the perspective of EAI, several aspects of the Web revolution are especially significant. First, the Web provides universal connectivity. Anyone with a Web browser and an Internet connection can view your Web site—and can move to your competitor's with a single mouse click. This fact leads to the second point: The Web creates a whole new arena for business competition. A third point is equally important: The user is in charge. Users who have experienced the sense of control that is achieved by the best Web sites—be they customers, supply-chain partners, or employees—are reluctant to settle for anything less in any Web site they use.

For any enterprise, the Web revolution cuts two ways. On one hand, a business can offer its customers revolutionary new products and services through the Web. On the other hand, virtually every business is compelled to join the race to offer those new products and services because its competitors may do so—and those competitors are just a mouse click away!

In some cases, the Web simply offers a new and better way to perform an existing service, such as checking an account balance. Other Web applications offer novel products or services that are possible only through Web technology. Business partners and consumers can meet and exchange goods and services through electronic markets. Business partners can greatly improve their coordination through Web-based exchange of information, streamlining the flow of goods and paperwork and squeezing costs out of the supply chain. And, in fact, they *must* do so; if they don't, their competitors will. Enterprises can use internal Webs, called intranets, to share information and improve business processes within their own organization or risk being at a competitive disadvantage to those who do.

The Web has led to new customer expectations. Customers expect to be able to find a business on the Web. They expect to be able to get information about products and services. They expect to be able to enter orders or check accounts. They want to be able to check the delivery status of an order, whether the order was given via the Web or some other channel. Soon, customers will expect that if they run into a problem on a Web site, they can contact a service representative either on the phone or via Internet messaging. They expect that the service representative to whom they are connected will then be able to see exactly what the customer was doing on the Web.

Customer self-service is perhaps the best example of the impact of the Web on customer relationships. Customers like the feeling of control that they get from the best self-service sites. Retailers like Amazon.com have become major players in the marketplace by offering customers a Web experience that is unlike anything they could enjoy through more traditional channels. Businesses also like customer self-service because it can significantly reduce the cost of a transaction. It has been estimated that a typical

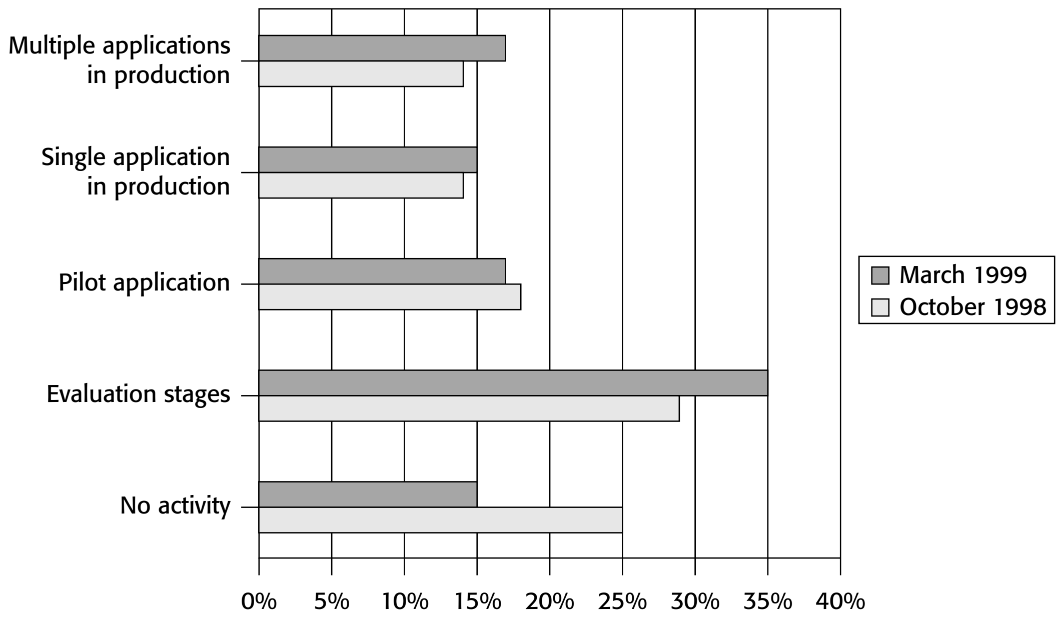


Figure 1.4 The Web revolution is just beginning.
Source: SoundView Technology Group.

banking transaction costs $1.07 at a branch, 54 cents by telephone, 27 cents at an ATM, and 1 cent on the Web [Jupiter Communications 1997].*

Web-based application integration has moved from the experimental, "bleeding-edge" phases to the mainstream, but it is still in its early stages. A March 1999 study by SoundView Technology Group found that only 33 percent of the enterprises surveyed had any Web-based electronic commerce applications in production, and half of those had only one application (see Figure 1.4). Still, the direction is clear. In addition to the 33 percent that had applications in production, an additional 52 percent were in either pilot application or evaluation stages. It seems clear that the Web revolution is just beginning.

Effective Web-based applications often depend on the integration of existing applications. To a large extent, the information and services that enterprises want to exploit in their public Web sites and their corporate intranets are locked up in existing IT applications. The account balances that a bank wants its customers to be able to check via the Web may already be available—but in terminal-oriented systems that were designed to provide the information only to branch employees. If an employer wants to be able to offer its employees Web-based self-service benefits administration, it must find a way to

*Jupiter Communications (www.jup.com) is a media research firm focusing on how the Internet and other technologies are changing traditional consumer industries.

access the capabilities in its existing Human Resources applications. To build the Web applications that it needs to remain competitive, an enterprise must master EAI.

Web applications and EAI are not synonymous, of course. Many Web applications do not involve application integration, and many EAI projects have nothing to do with Web applications. The emergence of the Web, however, has provided a crucial incentive for enterprises to utilize EAI. While not all Web applications involve application integration, the most important ones—the ones that are most fundamental to enterprises' e-business strategies—usually do depend on integration. And when an enterprise must move at "Internet speed" to be competitive, EAI is essential to minimize time to market.

Barriers to Effective EAI

So far we have discussed the business reasons why enterprises want to employ EAI. We also need to consider the following barriers to effective EAI:

- The complexity of existing enterprise architectures, which in many cases have grown up without any controlling blueprint, and the sheer difficulty of accomplishing large-scale application integration because of software complexity, competitive architectures, and performance issues.
- The lack of skilled staff who are familiar with EAI technologies. As is often the case with rapidly growing new technologies, the demand for skilled personnel exceeds the supply.
- The increased need for security as applications become more highly integrated.

These barriers represent risks that must be overcome. They also motivate the use of middleware technology, which can help overcome some of the barriers by providing standard services for defining and maintaining interfaces, simplifying architectures, and securing applications. Staff can be trained more readily because they need only learn a common middleware technology, rather than a host of application-specific APIs. In the following sections, we will consider each of the barriers to effective EAI in turn.

Chaotic Architectures

In many enterprises, IT architectures have evolved with little forethought or coordination. Departments set up their own midrange servers, usually based on Unix or Windows NT. Because each department picks its own technology, the enterprise architecture as a whole has no coherence. No one is responsible for the overall architecture. Often, the new departmental applications require information from the corporate mainframe, but each application interface is treated as a local and unique problem. Information about customers and suppliers is often replicated by each line of business, leading to a never-ending struggle to maintain consistency. Definitions of data elements vary from application to application, so that even if the information can be brought together, analysts find themselves comparing apples not to oranges, but to pianos.

The result is that, when looked at from an enterprise perspective, IT architectures are a mishmash. One company that we know of refers to its IT architecture as "the fur ball" because its connections between applications are so tangled. Even limited integration efforts are time consuming and costly. Indeed, Gartner Group estimates that 35 percent of large enterprises' IT budgets are devoted to integration.

Given this situation, it is not surprising that application integration can be extremely difficult. IT staff must master the diverse applications and technologies of existing in-house applications, plus the new packaged applications that are continually being introduced. If they are using EAI tools, they must master those as well, and they often must struggle with a poor match between the tool's capabilities and their needs. They must figure out how to harmonize the data in various applications, increasing its consistency.

Introduction of packaged applications, like the ERP packages from vendors such as SAP and Baan, have solved some problems while introducing others. Vendors often provide APIs based on their own custom technology, rather than on industry standards. The APIs can be extremely complex. And they may change in the next release of the product. Fortunately, this situation is beginning to change, at least for the major ERP vendors, which are beginning to support standards-based integration approaches. Many packaged application vendors, though, continue to provide little integration support or to support only proprietary interfaces.

API

API refers to "Application Programming Interface." An API is a mechanism provided by an application to access its functionality or data.

Just as an existing chaotic architecture is an impediment to EAI, getting control of the architecture is an essential element in the introduction of EAI. EAI is not just about technology: Enterprises must adopt new approaches to such topics as interface control, information standards, and systems management in order to employ EAI effectively.

Skill Shortages

Shortage of skilled personnel is a major barrier to successful EAI. EAI technologies often require different skills than those that have traditionally been available in corporate IT groups. Middleware skills are in particularly short supply, including knowledge of Message Oriented Middleware (MOM) technology, the Common Object Request Broker Architecture (CORBA) and its associated products, Microsoft Distributed Component Object Model (DCOM), Enterprise JavaBeans (EJB), and message broker technologies. (We will discuss each of these technologies in the course of this book.) Perhaps even more important is the lack of skilled architects who know how to combine these tools with legacy and packaged software to provide complete solutions.

Security Issues

New applications that rely on application integration and on Web and Internet technologies can raise nettlesome security problems. Security for these systems requires a more comprehensive and integrated approach than security for more traditional applications.

The security problem is real. In the 1998 FBI/Computer Security Institute Computer Crime and Security Survey, 64 percent of respondents said their enterprises had been successfully attacked. Data modification occurred in 14 percent of the attacks. Quantifiable losses reached $136 million.

EAI applications can be particularly vulnerable, especially if they involve Internet applications. The Internet has millions of users, most of whom will be unknown to your enterprise and some of whom will be malicious. The nature of the Internet makes controls difficult. A hacker doesn't even need to be particularly skilled to attack your enterprise: Powerful tools and expert hacker guidance are easily accessible on the Internet itself (see Figure 1.5).

Netscape - [PHRACK MAGAZINE HOME PAGE]
Welcome to the official Phrack Magazine Web Page.
Netscape - [The Hacker Starting Point]
THE HACKER STARTING POINT
Hacking Links
Welcome to the UNDERGRO
Hack & Crack Utils
Unix Password Crackers for DOS
Netscape - [2600 Magazine]
2600
The Hacker Quarterly
Document: Done

Figure 1.5 Guidance for hackers can be found on the Internet itself.

Even internal applications are vulnerable. Attacks do not come just from hackers on the Internet: 44 percent of the attacks involved unauthorized access by employees. And the nature of EAI applications is to make information more accessible, to move critical data and functionality from the data center "glass house" onto the corporate network where, without proper controls, it is accessible to anybody.

Protecting applications is difficult. Many security attacks exploit obscure bugs that can make programs vulnerable in unexpected ways. Sending a message a thousand characters long to a program that expects messages to no longer than a hundred characters can cause the program to perform in an unexpected way and turn it into a tool for the unscrupulous hacker. Once again, EAI applications may be more vulnerable because in an EAI situation, legacy applications are being accessed in ways and by users that their designers did not expect. For example, a legacy application may require users to log in via a user ID/password screen. In an EAI situation, this application might be accessed directly by another program that bypasses the terminal-based front end, including the log-in screens.

These security problems cannot be overcome on an ad hoc, case-by-case basis. EAI applications require a comprehensive, consistent, and coherent security architecture. This architecture must provide for the reliable authentication of individual users, consistent application of security policies for access to information and functionality, accountability through accurate auditing of security-relevant activities, and encrypting of confidential information to protect it from interception, especially information that flows over the open Internet. Moreover, the security system must be easy enough to use and administer that the sheer burden of complying with security requirements does not lead to the introduction of new vulnerabilities.

Final Thoughts

Enterprise application integration is everywhere on the IT landscape today. EAI's emergence as a key topic for IT is driven by business imperatives. These include the emergence of the Web, the need to develop and deepen relationships with customers and partners, streamlining internal business processes, and reducing the time to market for new applications.

In responding to these imperatives, enterprises are strongly motivated to make effective use of their existing custom-written legacy applications and commercial packaged applications. These applications can be the enterprise's crown jewels. They work, they are often highly efficient and robust, and they represent an enormous historical investment. And they have just received significant additional investment because of the Y2K problem.

Despite the strong pressures, line-of-business and IT managers must realize that implementing effective EAI will not be easy. In many cases, the existing enterprise IT architecture is chaotic. Applications are deployed on a wide variety of platforms and utilize many different technologies. Data is often redundantly stored in various systems, inconsistent, and difficult to integrate. Staff members with the right skills are in short supply. And there are numerous and difficult security challenges.

A bewildering number of strategies and technologies can be used to integrate existing applications. In the next chapter, we will discuss the various types of integration and how they can be applied.

CHAPTER 2

Types of Integration

Key Facts

- Integration can occur at three points in an application—the presentation, functional, or data layer.
- A goal of integration is to reduce coupling.
- Tightly coupled integration can become a serious maintenance burden.
- Presentation integration is easy but very limiting.
- Data integration provides a broader solution than presentation integration but may require rewriting software to properly manipulate the data.
- Functional integration is the most important model, but it is the most complex.
- There are three types of functional integration: data consistency, multistep process, and plug-and-play components.
- A good EAI architecture will implement solutions for data and functional integration and possibly presentation integration.

The concept of integration as it applies to information systems has changed dramatically over the last three decades. Originally integration focused on bringing together a diverse set of hardware in support of a software application developed from scratch. As hardware advances allowed increasingly complex software applications to be constructed, the nature of integration changed. Integration grew to mean the combining of hardware and software to form a system. Now the concept of integration is dominated by software. Organizations are increasingly focused on integrating their existing and new software to form new applications. The emergence of networking has created a situation where this software resides across the network on different hardware platforms, making this integration more complex than ever before. EAI represents the evolution of system design and technology aimed at reducing the complexity of solving the integration problems of today.

With the change in the nature of integration has come an increase in both the range and complexity of integration options. These options include sharing data between applications while ensuring quality and consistency, providing integrated front-end access to applications, bridging applications through workflow, and building new applications that pull together information from existing applications in an innovative manner.

The ultimate goal of EAI is to allow an organization to rapidly integrate diverse applications quickly and easily. In this chapter we examine the breadth of approaches to performing integration. Three models of integration will be presented and explored. A good EAI strategy and architecture will allow an organization to apply all of these to satisfy the business requirements described in Chapter 1, "The Business Drivers of EAI."

Integration Models

An *integration model* is a prescribed approach and configuration used to integrate software. It provides a range of different options for the approach and configuration as well as set requirements and limitations, and it emphasizes one or two of the following attributes:

- Simplicity of performing the integration
- Reusability of integration for different configurations
- Breadth of possible approaches to integration
- Expertise required in performing integration

Different models of integration have emerged as the complexity of integration has shifted to a software-dominated view. Originally integration was accomplished through the customization of software to produce the desired outcome. Integration through customization of software can be used to solve any problem. It, however, requires significant effort to handcraft each integration. A large part of this effort is related to the distributed processing aspects of the integration that must be custom developed as well. These custom integrations also were difficult to reuse, requiring a duplication of effort for each integration.

integration model

An integration model defines how applications will be integrated by defining the nature of and mechanisms for integration.

Several models have appeared along with supporting tools that each aimed to reduce the time and cost or increase the reusability and flexibility of integration. Each of these focused on integrating through a different part of the application. The three possible points of integration in an application are the presentation, the software functionality of the databases, or files.

The integration models that represent different state-of-the-art approaches to integrating software are these:

- Presentation integration
- Data integration
- Functional integration

The *presentation integration model* is one of the simplest forms of integration. In this model the integration of multiple software components is accomplished through an application user interface. Typically the integration results in a new, unified presentation to the user. The new presentation appears to be a single application although it may be accessing several legacy applications. Integration logic, the instructions on where to direct the user interactions, communicates the interaction of the user to the appropriate software using their existing presentations as a point of integration. It then integrates any results generated from the various integrated software components. For example, a screen-scraping tool could be used to take a set of mainframe applications and integrate them into a new Microsoft Windows application. This single presentation would replace a set of terminal-based interfaces and might incorporate additional features, functions, and workflow for the user. This creates an improved flow between legacy applications for the user.

presentation integration model

A presentation integration model allows the integration of new software through the existing presentations of the legacy software. This is typically used to create a new user interface but may be used to integrate with other applications.

The *data integration model* is based on integration via access to the data of a software component. The software of an application is bypassed, and the data that is created, managed, and stored by it is directly accessed. For example, a database gateway could be used to access a customer order system that used an IBM DB2 database and a customer billing system that used an Oracle database. The gateway would pull information from each database into a data mining application that assesses customers' buying habits. With the use of the gateway, the order processing and billing applications are bypassed.

data integration model

A data integration model allows the integration of software through access to the data that is created, managed, and stored by the software typically for the purposes of reusing or synchronizing data across applications.

The *functional integration model* is based on integration of software at the code level. This might be at the level of a procedure or an object. It could also be done through an application programming interface (API) for the software if one exists. For example, the functionality of the ordering and billing applications would be accessed to update a customer's address from a third application. Functional integration rather than data integration would be used if the address information provided to the billing or order software required manipulation that already existed in the legacy software prior to the data being retrieved or stored. Rather than re-create the logic in the new application it is more efficient and less error prone to reuse the existing logic. Access is customized to each application, including the semantics and behavior of the application. Another approach to functional integration would use a connector that hides the internals of the application. A connector would support the request for customer information or could respond to a request to change an address. A connector is software whose purpose is to provide access into the software and its functionality while hiding the complexity of creating the actual connections into the software. It makes the software appear to have been designed with the original intention of providing easy access and integration.

functional integration model

A functional integration model allows the integration of software for the purpose of invoking existing functionality from other new or existing applications. The integration is done through interfaces to the software.

Before we describe each of the three models in further detail, we need to explore the important concept of coupling.

Integration and Coupling

Whenever two software components are integrated together several factors heavily influence the quality and utility of the integration. The factors are these:

- The integration model that is applied
- The tools that are used
- The designer's choices

One of the most important outcomes of any integration is to reduce the level of coupling between the software components that are integrated. *Coupling* measures the level of interdependency between the two components and the impact that changes in one component will have on the other. The goal is to have loosely coupled components so that there is little or no dependency.

coupling

Coupling defines the degree of integration. Loose coupling is where the integration is dependent on a few discrete interfaces. Strong coupling is usually where implementation dependencies occur.

A related concept is *white and black box integration*. Presentation and data integration models utilize a *white box* approach to integration. This means that the inside of the application or database is visible to the integrator, as opposed to a *black box* integration approach in which the internals of the application or database are hidden from the integrator. Functional integration can be implemented in either a white or black box approach. A black box approach requires the existence of APIs, connectors, or other forms of interfaces from which the internal functionality can be accessed. A white box approach is required when the software to be integrated does not have any APIs, connectors, or interfaces from which to access the internal functionality. In this instance the creation of a connector would allow future integration to be performed in a black box approach if the connector can be reused. White box integration creates a higher degree of coupling than black box integration. Black box integration tends to lead to more reusable, plug-and-play systems.

white box integration

The white box integration approach exposes the internals of the application or database to the integrator in order to produce the required integration. This requires detailed understanding of the internals of the software and usually leads to tighter degrees of coupling.

black box integration

The black box integration approach hides the internals of the application or database from the integrator, and the integration is done through an API, connector, or some other form of interface. The details of the internals of the software are hidden and usually lead to loosely coupled systems.

Presentation Integration Model

When terminals were replaced by the personal computer, user expectations also changed. PCs employed graphical user interfaces (GUIs) with improved access to existing applications. The requirements for improved access included the ability to integrate with multiple applications as well as add business logic related to the management of the interface, such as validation, error checking, and calculation. By presentation we are referring to the user interface that provides access to an application.

The presentation integration model is based on the concept of accessing the legacy application through its existing presentation logic. It allows a new user interface to be developed by remapping the old presentations. Each user interaction, however, must ultimately map into the old presentations in order to integrate. It is also possible to use these integrations with other applications as well, but doing so is limited to what actions are possible through the existing user interfaces.

Figure 2.1 shows the presentation integration model in which a common presentation is built from the existing presentations of two different applications. Screen scraping is a popular technology for integrating systems under this model. Screen scraping allows the programmer access to the legacy presentation and the ability to use new technology to create a graphical user interface.

When to Use It

You would use the presentation integration model when you want to do the following:

- Put a PC-based user interface on an existing terminal-based application in order to provide an easier-to-use application for an end user
- Present an interface that the user perceives to be a single application but is, in fact, the composite of several applications
- Integrate with an application whose only useful and implementable integration point is through its presentation

This form of integration is useful only when the integration can be accomplished using the user interface or presentation level of the legacy applications. Integration of this type is typically oriented to textual user interfaces such as IBM 3270 or VT 100 interfaces. Examples where the presentation integration model is best applied are these:

- Providing a Microsoft Windows interface to a mainframe application
- Providing a unified HTML interface to an SAP R/3 and mainframe application
- Providing a unified Java-based interface to multiple mainframe applications

In the first example, the integration is light. It encompasses only the integration from the mainframe to the Windows application. The second and third examples require additional integration logic as they combine functions and data across two applications. This form of integration has been successful because it can be applied quickly to solve

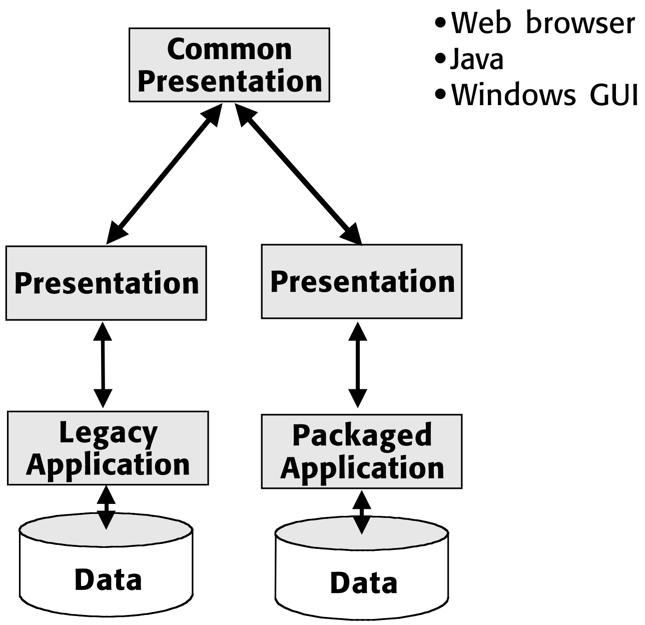


Figure 2.1 The presentation integration model integrates through the user interface of applications.

a specific type of integration problem. It is a simple form of integration requiring limited expertise in the integration tools, and therefore it has a lower cost to implement. Reusability across applications is limited, however, and there are a limited number of features and functions. For instance, presentation integration can be used to improve a user's experience by reducing the complexity of accessing multiple applications.

Pros and Cons

Presentation integration is very easy to accomplish and can be done relatively quickly. Presentation logic is usually less complex than either data or functional logic because it can be viewed, it is easily discovered, and it is usually well documented or self-describing. When the tools used to perform this integration work well they do most of the work necessary to create the integration. The developer focuses on the construction of the new presentation.

On the other hand, presentation integration occurs only at the user interface level. Therefore, only the data and interactions defined in the legacy presentations can be accessed. In addition, presentation integration can have performance bottlenecks because it adds an extra layer of software to the existing application. The underlying data and logic of the existing application cannot be accessed.

The presentation integration model is the most limiting of the three models. The integration takes place in the presentation and not in the interconnection between the applications and data.

The Data Integration Model

The data integration model goes directly into the databases or data structures of an application, bypassing the presentation and business logic to create the integration. The data integration model is depicted in Figure 2.2. The integration may be as simple as accessing a database management system (DBMS) that is used by an application or may require more sophisticated integration with custom databases or files managed by the application.

A variety of tools and data access middleware has been used to access and integrate information from databases. Examples of these tools and middleware include the following:

Batch file transfer. Tools that allow files to be moved between systems and applications in either an ad hoc or a prescribed manner. This class of tool was originally oriented around the transfer of batch files created on mainframes to other applications. They were the earliest tools used for data integration. It can be argued that these tools do not provide real data integration; rather, they merely move data around.

Open Database Connectivity (ODBC). A standard application programming interface that abstracts access to heterogeneous relational databases. This interface was

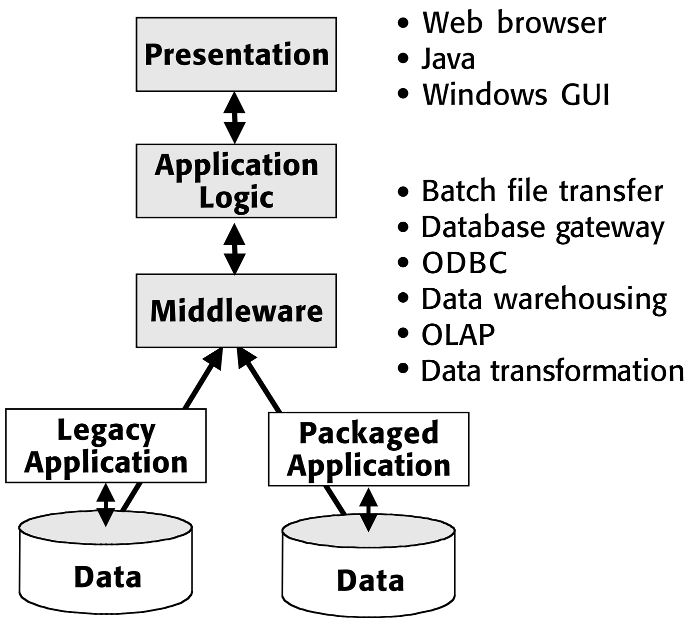


Figure 2.2 The data integration model integrates directly to the data created and managed by the application.

the first widely accepted standard for accessing relational databases. It provides an interface that can be used to integrate with a database that supports it. In addition, the interface can be used on other forms of data storage as long as they support the semantics of the prescribed operations.

Database access middleware. A form of middleware that focuses on providing connectivity to distributed databases. Middleware implies that these tools allow for connectivity between software components in combination with runtime capabilities to manage interactions between the components. Data access middleware focuses on the exchange of queries, management of results, connectivity to databases, pooling of connections, and other housekeeping tasks related to data management.

Data transformation. A tool that usually complements middleware. It provides the ability to convert data from the source database format into the target database format. Because data definitions, structures, and schemas are often different for applications the exchange of data often requires conversion. Examples of conversion include converting EBCDIC to ASCII data format or changing dollars to yen. A transformation tool will handle these types of problems.

Each of these technologies can be used to solve point problems or can be combined to solve more complex data integration problems. For example, batch file transfer and database access middleware might be used to connect a variety of data sources, some of which might support ODBC. The data might be processed in a data transformation tool before being placed in an application. These technologies have evolved over many years as data integration requirements have expanded.

data access middleware

Data access middleware is a type of software that facilitates access to databases through the use or creation of connectors. In addition, it provides the runtime environment to manage the requests sent to these databases and returns the results.

When to Use It

You would use the data integration model when you want to do the following:

- Combine data from multiple sources for analysis and decision making. For example, when you want to access data from multiple data sources that contain customer information and place this data into a statistical software package.
- Provide multiple applications with read access to a common source of information. For example, when you want to create a data warehouse that has a complete set of customer information that can be accessed by a variety of data mining and statistical applications.

- Allow data to be extracted from one source and reformatted and updated in another. For example, when you want to update customer address information in all data sources so that they remain synchronized and consistent.

In each of these instances integration occurs through the movement of data between applications. Either data is moved in bulk form or an application requests a specific data record. Examples of when to use the data integration model are these:

- Combining customer data from Sybase, IBM DB2, and SAP R/3 databases into a call center application. In this example, database access middleware would be used to access the data and integrate it into the call center application. The call center application would send a request to the data access middleware. The middleware would send the request to each of the databases in the correct format and wait for the responses. The responses would be passed back to the call center application.
- Creating an executive information system (EIS) that uses data from mainframe and Oracle databases. In this case, there are several possible solutions. One might be to perform a nightly batch file transfer from the databases into the EIS. Another would be to use data access middleware to request the data specific to an analysis to be performed. In the case of the batch file transfer it would be scheduled on a regular basis, such as every night. The batch file transfer tool would move a copy of the data file to the EIS system at the prescribed time. The EIS would never be involved in the movement of the data. In the case of the data access middleware, the EIS system will make a request for the data it needs and await a response. In the batch mode, data will be current only up through the last transfer. In the data access middleware case, it will be the current data.
- Allowing other applications to access information in Peoplesoft and custom Oracle databases. This could be accomplished through the use of ODBC interfaces provided by the databases that will be accessed. New applications would use the ODBC as a consistent interface to access data. Using ODBC interfaces will give an application the ability to send a request to the custom Oracle application and Peoplesoft databases. The applications that use these ODBC interfaces, however, will be required to manage the interactions. Database access middleware may be used to ease the burden from the applications.

The data integration model is applicable in each of the previous examples. Selection and configuration of the tool, though, will determine the characteristics of the integration. These scenarios differ in the complexity of the integration of data, the freshness of the data (Is a nightly transfer acceptable or should data be fetched in real time?), and ease of access across the system. Each of these characteristics must be considered in deciding on the best approach and tools to apply data integration.

Pros and Cons

The data integration model provides greater flexibility than the presentation integration model. It provides access to a wider range of data than is available when integrating through a user interface. It also allows access to either a complete set of data or subsets depending on the need of the new application.

This approach also simplifies access to data sources. When the databases are easily accessible with interfaces or when the middleware provides multisource data integration to new applications, the data integration model promotes rapid integration.

The data integration model also allows data to be reused across other applications. Once an integration is complete it can be reused. For example, in our call center example, this integration could be reused in a new self-service Web site. By integrating at the data level, however, applications must write any functionality required to manipulate the data. For example, if a balance for a customer is calculated based on the value of several fields this logic must exist in every application that uses the data.

The need to rewrite the business logic may appear to be a minor issue, but it can be a monumental problem. For example, consider a bank where the data integration approach was used for accessing checking information. The logic to recalculate the balance of the checking account may already exist in the business logic of the application that creates and uses the database, but it would not be available to other applications that integrated using the data integration model. In such a situation you would have to write logic to recalculate the balance of the checking account. This would occur because the checking account balance data might not reflect deposits that had yet to clear. This duplicate logic could have a very high cost in both creation and maintenance.

Last, each integration is tied to a data model. If a data model changes, the integration may break, making data integration sensitive to change. Because systems tend to be evolutionary this change can require significant effort to maintain the integration.

Functional Integration Model

A significant portion of IT budgets is spent on the creation and maintenance of business logic. Business logic is code written to perform required business functions in an application. It includes the processes and workflow as well as the data manipulation and interpretation rules. Therefore, it is not surprising that enterprises have found that both presentation and data integration quickly run out of steam as a solution to the integration problems faced in an enterprise. The business logic contains the rules that are required to properly interpret or construct the data and that are not always available through the presentation.

business logic

Business logic is the implementation of business processing in a programming language.

The functional integration model integrates at the business logic level, as opposed to the presentation or data levels. Figure 2.3 depicts the functional integration model.

Functional integration requires that the point of integration be in the code of the application. This point of integration could be as simple as access through a published application programming interface (API) or as difficult as requiring additional code to create a point of access.

Remote procedure calls were once thought to be the answer to solving these problems. The use of distributed processing middleware, though, has become the preferred mechanism. Remote procedure calls provided only the definitions for access and basic communications capabilities. They typically required significant effort to use in the development of software. Middleware provides a more robust approach that combines interface definition and communications as well as runtime support for managing the requests between software components.

distributed processing middleware

Distributed processing middleware is a type of software that facilitates the communication of requests between software components through the use of defined interfaces or messages. In addition, it provides the runtime environment to manage the requests between software components.

The three categories of distributed processing middleware are as follows:

Message Oriented Middleware (MOM). Provides integration by passing messages between applications. Similar to the concept of the Post Office, a message is placed into the MOM. The MOM is then responsible for delivering the message to the target system. MOMs can be implemented in a variety of configurations, including message queuing and message passing. Examples of products include IBM's MQSeries and Talarian's Smart Sockets.

Distributed object technology. Applies object-oriented concepts to middleware. Interfaces make software look like objects. The software then can be accessed by other applications across a network through the object interfaces. Technologies such as OMG's CORBA, Microsoft COM+, and Sun's Java 2 Enterprise Edition (J2EE) environment are all examples of distributed object technology.

Transaction processing monitors (TPMs). Provide mission critical support for distributed architectures by allowing a transaction to be managed using concepts such as two-phase commit. They preserve the integrity of distributed information resources such as databases, files, and message queues. Products such as BEA's Tuxedo are transaction processing monitors.

Each of these types of distributed processing middleware can be combined (and have been by a variety of product vendors). Individually or in combination they can be used

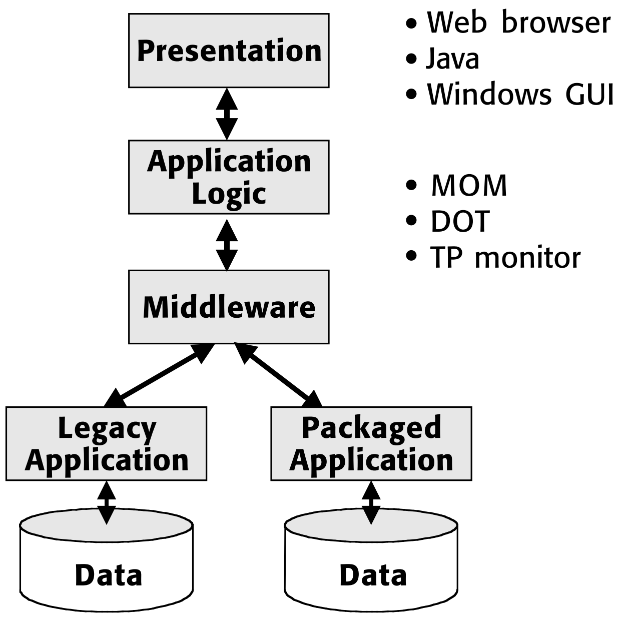


Figure 2.3 The functional integration model integrates directly with the code of the application.

to integrate applications at the functional level. Each of these types of middleware will be addressed in more detail in Chapter 4, "Messaging Architecture and Solutions," Chapter 5, "Object Architecture and Solutions," and Chapter 6, "Transaction Architecture and Solutions."

The Breadth of Functional Integration

Functional integration is more flexible than data and presentation integration. It can be broadly applied using three different approaches. Each approach has different characteristics and is used to solve a different type of functional integration problem. These approaches are as follows:

Data consistency. The coordination of information updates from one or more sources across integrated applications.

Multistep process. A coordinated set of actions executed across integrated applications.

Plug-and-play components. The creation of reusable interfaces across applications that simplify construction of new applications.

We will now look at each of these in more detail.

Data Consistency Integration

In many enterprises there exists a wide variety of applications that have been developed and enhanced over a long period of time. These applications might represent various relationships that a customer would have with the enterprise, manage different types of employee information and benefit systems, or contain different information and business processes related to a product. Data about customers, employees, and products is often duplicated in each of these systems. When an update is made to one system there is a need to propagate updates across systems. Figure 2.4 is an example of this situation. For example, when customers change their addresses they want the updates to occur in every system that contains their information.

data consistency integration

Data consistency integration is integration through the code of an application where the purpose is to access or update data. The integration facilitates the communication of data and actions.

Although data consistency may seem like a data integration issue, it is a functional integration problem. If a customer makes a deposit into a checking account, changing the

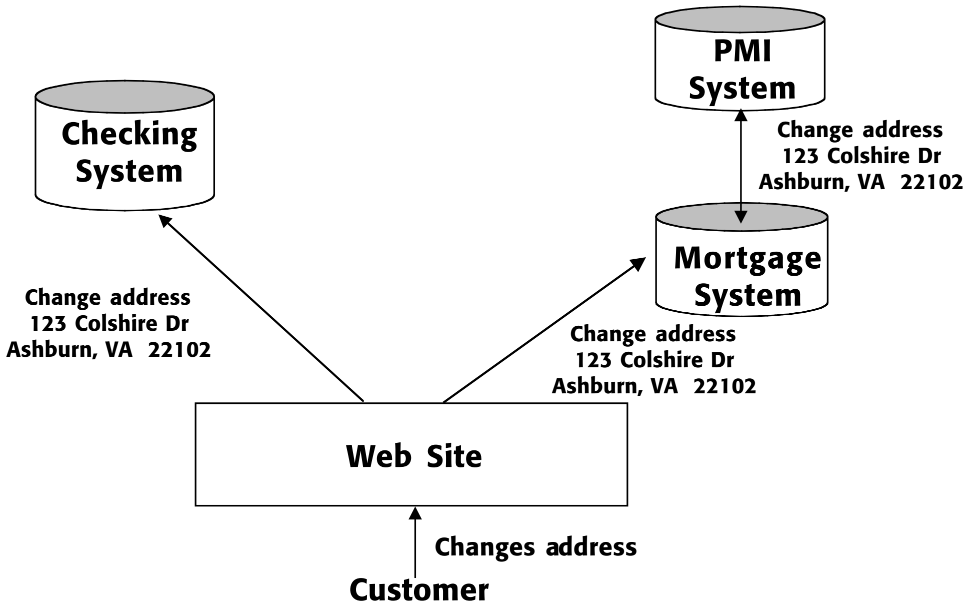


Figure 2.4 Data integration should ensure data consistency across applications.

database directly may not have the desired effect. Why? In this example the effective checking account balance may be a combination of available balance plus deposits waiting to clear. Banks often place a three- to five-day waiting period on deposited checks. A deposit might be made through the teller or ATM systems. In this instance a deposit requires updating several records as well as performing calculations that would require the replication of business logic in the new application. Duplication of this is costly to develop and maintain and creates additional chances of error. The logic would be required in the checking, teller, and ATM systems in the case of data integration but only in the checking system in the case of functional integration. Therefore, the application sending the deposit information to the checking system should not modify the database but should instead use the functionality of the application.

Another example where applying the data consistency approach is utilized is in the propagation of information. For example, when a change is made to a master catalog the update may need to be promulgated across multiple applications and databases. A distributor might have a broad range of products it sells represented in the master catalog. In some instances a distributor might create custom catalogs for classes of customers, distribution channels, or special clients. In addition, catalog data in new Internet-based e-commerce systems might be updated from the master catalog. These updates would be done through the functionality and not the database to ensure that the data is properly entered into the database.

Typically the data consistency type is implemented by sending a request to each system that describes the action that is intended along with the data. This request may be passed on to other applications as necessary. For example, in the master catalog example, a request would be transmitted to each application that maintained a catalog database to update or delete information based on changes. In the banking example, a deposit request could be generated to the checking system.

The Multistep Process Integration

The concept of multistep process integration, also referred to as *straight-through processing*, is beginning to take hold in many enterprises. It means that an action is properly processed across all relevant applications in the correct order of precedence without human intervention or retyping of information. For example, an order can be processed from creation and shipment to billing and payment. Figure 2.5 is an example of multistep process integration.

For example, an order is placed through a Web site to purchase a product. The order processing system creates the order and then notifies the logistics and shipping system to ship the product. When the order is completed, the order processing system is notified to change status and the billing system sends out a bill for payment. Once payment is received the order processing system is notified to close out the order.

Multistep process integration is often associated with workflow. In fact, workflow software is one type of software that can be used to implement a multistep process integration.

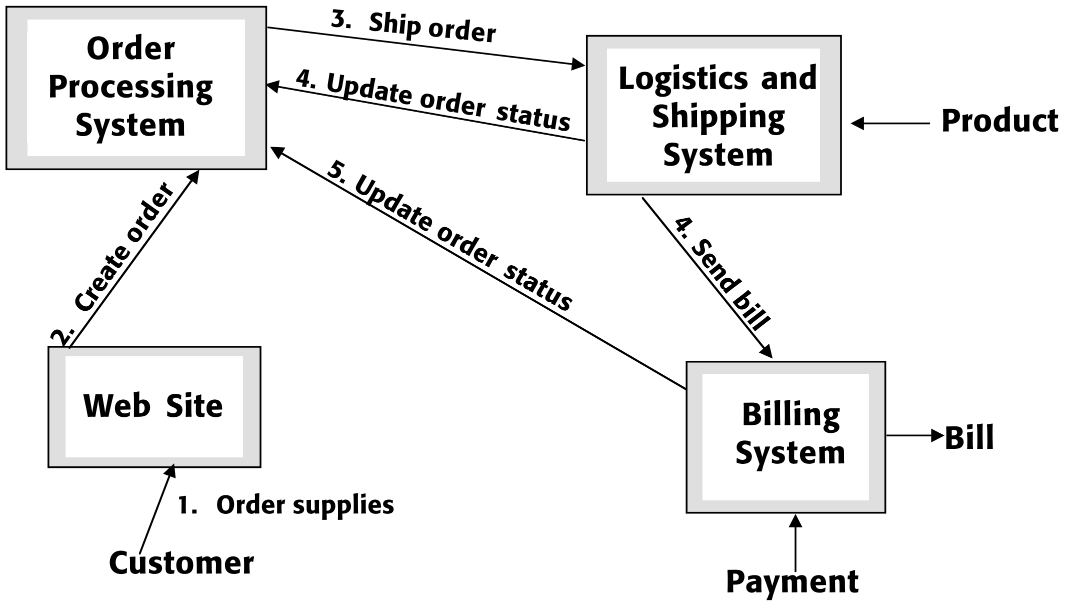


Figure 2.5 Multistep process integration is a form of business automation.

multistep process integration

Multistep process integration, also known as straight-through processing, is the integration of applications where there is not only communications of requests but also the coordination and management of these requests across applications. The integration facilitates communication of the request and manages the flow and sequencing.

Like the data consistency type, the multistep process type of integration is also implemented using requests. The requests, however, can have very complicated paths between applications, unlike the data consistency type, in which the request has a simple communications model and is stateless (meaning that once it is received it is forgotten). This requires that state be maintained in order to ensure the appropriate handling of requests. Both of these attributes require more sophisticated capabilities.

In order to perform multistep process integration, there needs to be a capability to define a series of steps that represent the flow of integration between applications. In a basic capability the steps would be followed in the same sequence each time. This capability does not require a workflow engine unless the steps become more complex and include decision making, fan in and out, or complicated timing.

Multistep process integration can be difficult to understand for a designer or developer because the overall business logic for the solution is not contained in any single application. The software that performs the integration controls it. This is usually a middleware solution based on some form of distributed processing like MOM or DOT. This style of application development requires a fundamental change in the mindset of many designers and developers, who are trained to have all the logic exist in an application and view an integration as merely a communication mechanism.

The Plug-and-Play Component Integration

The topic of plug-and-play components has been discussed for many years. This concept, often referred to as *component integration*, is that software can be created as components with easy-to-understand interfaces that can be easily connected to form a new application. The concept of tinker toys is often associated with this concept as a way to visualize component integration. Component integration is the third approach to functional integration.

The concept of components and plug-and-play integration has been promised since the early 1980s. Recently technology has matured to a point where this is feasible. This type of functional integration is the most complicated of the different approaches but has the greatest value to the enterprise. The ability to pull out a component and replace it or add a new component without writing code would substantially reduce the time to produce new applications. Figure 2.6 shows an example of component integration.

component integration

Component integration is the integration of applications where a well-defined interface exists that allows a component to be easily connected with other components without modification. The integration facilitates the communication of requests and handles all of the interface definition and management.

The implications of plug-and-play components are these:

- The interfaces are formed using a consistent set of interface rules. This is required so that every interface can be understood by any other component. The rules are also referred to as the syntax.
- The definitions of the actions that can be performed are consistently applied. This is required so that when a new component is plugged into the application other components can communicate their requests. For example, all applications would use create_customer rather than new_customer or some other form. These definitions are also referred to as semantics.

Component integration requires specific definitions of interfaces that can be accessed from other components. This means the interface represents all of the actions that can

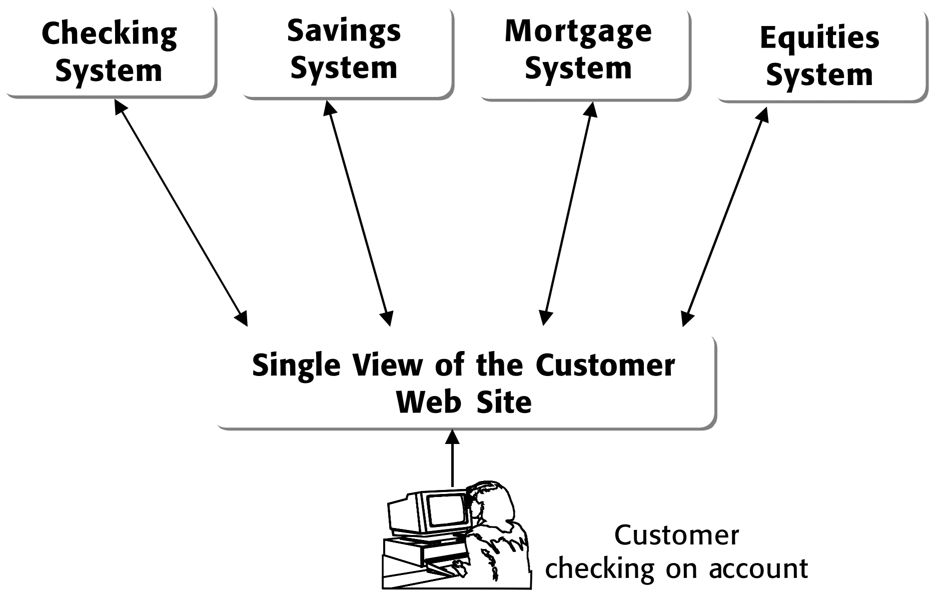

Checking System
Savings System
Mortgage System
Equities System
Single View of the Customer Web Site
Customer checking on account

Figure 2.6 Component integration is the most practical way to build and integrate applications.

be taken when integrated with other components. It also means that any other component should be able to make use of this component by issuing a request. The requests can be sent simultaneously to several components and require that state be maintained. The interfaces for each component must identify the specific functions that the component supports. In order for a component to be replaced the new component must be able to respond to the same requests that exist in the current system.

Implementing plug-and-play components is the most difficult of the three approaches to implement in an organization because of the coordination that is required by designers and developers in the enterprise. Technically it requires the development of sophisticated connectors that are more than just APIs. The connectors must be able to handle the mapping of requests in the applications, maintain state information about the integration, and provide operational support such as application management, security, and error handling. Organizationally, the connectors must ensure the consistent use of the tools and distributed processing middleware across the enterprise. Furthermore, in applying these tools and middleware the organization must enforce standards for their configuration and application. Very few organizations are able to do this today.

Comparing and Contrasting the Three Approaches

Enterprises need to be careful of falling into a trap of believing they need to use only one of these three approaches to solve their integration problems. They will most likely need to apply all three approaches at one time or another. The most sophisticated enterprises will know how to select the appropriate approach for any problem. Table 2.1 compares the three types.

Data consistency provides loosely coupled integration. This means that the applications have very little dependency on, or even knowledge of, one another. The requests tend to be one-way, from sender to receiver, and sent out by the sender, who will continue to process without requiring a response from the receiver. Because it provides a simple type of integration, the data consistency approach does not require (or support) complex business rules in the integration. Data consistency integration is easy to understand and implement but will solve only a limited number of problems, specifically synchronizing data between applications.

Multistep process integration tightly couples the integration of the applications. If any application is taken out of the sequence, the integration will typically no longer be able to function. Because of this complexity, the integration can be either one-way or request/reply, depending on the action being performed. In request/reply the sender expects to hear back from the receiver at an appropriate time. These requests can be either asynchronous, in which the sender continues processing after the request, or synchronous, where the sender awaits a response. The integration represents significant business logic in their actions. The result is that multistep process integration solves a wider variety of business problems. Specifically, it can be used to integrate full business processes that cross a wide number of applications and must occur in a specific sequence. In contrast, data integration is used to handle only one piece of a business process. It might be brittle to change, though, depending on the quality of the workflow tool applied.

Component integration can be loosely or tightly coupled integration, depending on the complexity of the interfaces. The more generic the interface definitions, the looser the coupling. The nature of plug-and-play tends to be more like procedure calls or invocations. Component integration can have the greatest amount of business logic embodied in the integration. The abilities to have significant business logic in integration and to replace components when required will allow an enterprise to create and evolve their information systems very rapidly.

Table 2.1 Comparison of Functional Integration Approaches

DATA CONSISTENCY	MULTISTEP PROCESS	COMPONENT INTEGRATION
Loosely coupled integration	Tightly coupled integration	Tightly coupled integration
Asynchronous dominates	Asynchronous or synchronous based	Synchronous dominates
One-way	One-way and request/reply	Request/reply
Simple abstraction of business rules	Complexity in abstraction of business rules	Complex abstraction of business rules

When to Use It

The functional integration model is unique in that it can solve a wider set of problems, including solving the same problems as presentation and data integration. This is accomplished in a fashion different from those employed in presentation and data integration. It is accomplished by accessing the code that the old presentation accessed or the code used to access the data. For example, use it to do the following:

- Put a PC-based user interface on an existing terminal-based application in order to provide an easier-to-use application for an end user by replacing the existing terminal interface and directly accessing the code from the new interface.
- Present an interface that the user perceives to be a single application but is, in fact, the composite of several applications by accessing the functionality of each application and integrating through the new user interface.
- Combine data from multiple sources for analysis and decision making. For example, when you want to access data from multiple data sources that contain customer information and place this data into a statistical software package. This is done by accessing the functionality that creates and manipulates the data source rather than going directly to the database.
- Provide multiple applications with read access to a common source of information. For example, when you want to create a data warehouse that has a complete set of customer information that can be accessed by a variety of data mining and statistical software. This would be accomplished by integrating each application into the data warehouse through the code that accesses and manipulates the data for the applications.
- Allow data to be extracted from one source and reformatted and updated in another. For example, when you want to update customer address information in all data sources so that they remain synchronized and consistent. This is accomplished through the creation of interfaces that notify other applications when an update occurs at one of the applications.

Deciding when to use functional versus data or presentation integration to solve these problems depends on a number of factors. The first is the feasibility of accessing the functionality in the applications. In some cases, accessing the functionality may be so difficult that it can be integrated only by using the presentation or through direct data access. The second factor is related to performance. The approach selected must meet the performance requirements of the system. Performance must be assessed on a case-by-case basis because it is so dependent on the particular situation. Finally, the issue of how much future reuse is expected must be addressed. Functional integration gives greater reusability but may be more difficult to apply, depending on the design of the code that performs the function to be accessed. Additional areas where functional integration is uniquely suited are situations that do the following:

- Require an action to be taken by the new application, such as making a deposit or placing an order that is managed by another application or set of applications. This requires the integration at the point in the code of the existing application that can process the action.

- Imply a workflow in the integration, such as processing an order all the way through distribution and billing. This requires creating an integration through a set of applications similar to the previous example but with sequencing and coordination in the integration software.
- Ensure transactional integrity between applications. For example, you want to ensure the deposit is complete before giving the customer a receipt. It is important to note that this is one of the more difficult types of integration, and it is not always possible.
- Anticipate the substantial reuse of a business function. For example, you want to enable a balance to be checked by a teller, a call center, an ATM machine, or an Internet banking application. This type of integration transforms existing applications into reusable components.

When applying the functional integration model, you can write additional business logic in the middleware layer that is separate from each of applications. This can be done in a variety of ways such as with code, with scripting languages provided by some of the tools supporting functional integration, or with graphical tools that configure the distributed processing middleware. In addition, you can also create new presentation logic for the user. New presentation logic would be created separately from the middleware layer and is dependent on the requests. An example would be a new e-commerce system.

Examples of functional integration are as follows:

- Updating a customer address from a call center into a Java application, mainframe application, and Oracle database. In this case, an interface or message would be defined and sent by the client application to which each application or database would respond.
- Processing an order from a Web site through delivery and payment. In this case, the distributed processing middleware is responsible for moving the order through the system in the correct sequence.
- Creating a single view of the customer across all enterprise applications and databases. In this case, the distributed processing middleware is used to create interfaces to each application. The interfaces represent the various actions and data related to a customer that is available from any application. New software is built that provides access to the applications that contain customer information and provides an integrated user interface.
- Integrating across multiple human resource systems that each support a different business unit to ensure the consistency of information that is duplicated or allow the transfer of employees across business units. This is done using connectors and distributed processing middleware.
- Creating an integrated supply chain by integrating the system of suppliers into your procurement system to streamline general procurement. This is done by passing messages using distributed processing middleware.

Pros and Cons

The functional integration model provides the most robust integration capabilities of all the models. It is the most flexible in the problems it can solve. It can be used to solve presentation or data integration problems. More importantly, it provides a higher degree of reuse of the components it creates than the other two integration models if properly applied.

This model, however, includes the increased complexities that come along with trying to integrate at the business logic level. The learning curve for the software that aids in functional integration is higher than for the presentation of data integration tools. In addition, it may be difficult to access the business logic of some applications because the source code may not exist or there may be no APIs.

Final Thoughts

Organizations must put together an EAI architecture that allows them to address a broad spectrum of integration problems. A good architecture will deal with at least data and functional integration. Using a presentation integration model is not mandatory but should be considered. This chapter has provided a broad overview of integration. In the next section we address the basic building blocks of EAI architecture with detailed discussions of the supporting middleware.

CHAPTER 3

Basic Building Blocks

Key Facts

➤ There are four basic building blocks in building an EAI architecture: communications model, method of integration, middleware, and services.

➤ An enterprise builds an EAI architecture by determining its requirements and then selecting products that meet them across the building blocks.

➤ The two types of communications models are synchronous and asynchronous. Both are required in a good EAI architecture.

➤ Interfaces and messages are the two methods of integration.

➤ Messages are most useful in data consistency and multistep process problems.

➤ Interfaces are most useful in component integration problems.

➤ Not all middleware is useful as the core of an EAI architecture.

➤ MOM and DOT are the two most popular core middleware building blocks.

➤ The need for TPM is based on the types of transactions in the enterprise.

➤ Services add significant value to the middleware and communications capabilities, making them easier to use or more beneficial.

Implementing EAI in an enterprise requires both methodology and technology. *Methodology* is the combination of definitions, processes, and guidance that when put together gives an architect, designer, or developer a structure to organize and execute the development of a solution more efficiently, effectively, and predictably. *Technology* is used to implement solutions. Methodology and technology share a symbiotic relationship where each must support the other in the development of a solution. Methodologies lead to a solution that can be implemented with existing technology. A variety of technologies can be used for integrating applications.

An EAI architecture, which is a combination of technologies brought together in a structured manner, is based on four basic technology building blocks. The fundamental building blocks are these:

- Communications model
- Method of integration
- Middleware
- Services

All four of these building blocks must exist and be interconnected into a single architecture. Methodology is necessary to help construct a solution and identify which portions of the architecture are most appropriate to the problem at hand. This chapter will address each of these building blocks in more detail.

An enterprise will need to physically construct an EAI architecture by selecting the right products that implement one or more of the building blocks and then bringing them together into a unified architecture. The selection of products should be based on how the particular product implements the features and functions of a building block. The enterprise also needs to address how a set of products will fit together into a unified architecture when selecting products. What differentiates one EAI architecture from another is the breadth of features and functions in each building block. In this chapter we will address each of these building blocks.

The Communications Model

The manner in which systems can interact is critical to their flexibility. The two basic choices for a communications model are synchronous and asynchronous communications. In general, synchronous communication requires the sender of a request to wait until a reply is received before continuing to process. It typically needs to wait because it needs the results of the request in order to continue processing.

Asynchronous communication allows the sender to continue processing after the request or content is sent. A reply may or may not be expected by the receiver as a result of a request. The term *request* is used throughout this chapter to refer to the communications from a sender to a receiver. In fact, this communication may not be a request but information. For consistency we will refer to them as requests.

receiver

A receiver is software that receives a request from a sender. The receiver may send a reply back to the sender as a result of the request.

request

A request is a formatted set of actions and data sent from a sending piece of software to a receiving set of software.

reply

A reply is a formatted set of data and possibly associated actions that are sent as a result of a request.

sender

A sender is software that sends a request to another software component.

Synchronous and asynchronous communications can be implemented and applied in a variety of forms. In fact, in the hands of a determined developer synchronous communication can be forced to act asynchronously and vice versa. Obviously, this is not the preferred way to use each of these tools; this topic has been the subject of much debate over the past few years. In any organization developers should have both in their toolbox and apply them correctly to the problem at hand.

Synchronous Communication

Synchronous communication requires a sender and a receiver to coordinate their internal processing in conjunction with the communications. This coordination implies that synchronous communication has a high degree of coupling. The rules for this coordination are dependent on the type of synchronous communication used.

synchronous communication

Synchronous communication occurs when the communication between a sender and receiver is accomplished in a coordinated manner. This requires the sender and receiver to operate dependent on the processing of request.

Synchronous communication is preferable in situations where the sender needs the result of the processing from the receiver or requires notification of receipt. Interactive systems require synchronous communication. Interactive systems include any application where a user expects to see information based on actions he or she takes or where two applications must work together to accomplish a business process. Examples include Web sites connected to databases, Java applications interacting with mainframe applications, or a Windows application interacting with an ERP package like SAP R/3.

Synchronous communication requires a reliable network infrastructure. If the network is unreliable then the sender may be in a wait mode while the request may have been lost in transit to the receiver. The sender might continue to send requests until a reply is sent, but without reliability the performance becomes significantly degraded and most likely unacceptable.

There are three popular types of synchronous communication:

- Request/reply
- One-way
- Polling

Synchronous communication is required for sending applications that need a response from a receiving application in order to continue processing. Each of these types handles this requirement differently. Let's now take a look at these types.

Request/Reply

Request/reply is the basic pattern for synchronous communication. An example is shown in Figure 3.1.

In request/reply communication, one application sends a request to a second application and blocks until the second application sends a reply. By blocking we mean it waits for a response from the receiver. The reply can be anything from an acknowledgment of receipt to complete processing with a response. After receiving the reply the sender continues processing. What it does when it continues to process usually depends on the

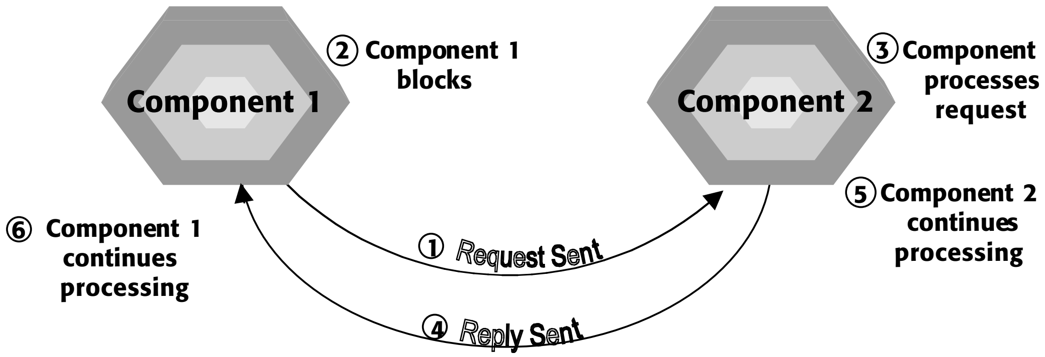

Component 1
Component 2
② Component 1 blocks
③ Component 2 processes request
① Request Sent
④ Reply Sent
⑤ Component 2 continues processing
⑥ Component 1 continues processing

Figure 3.1 The synchronous request/reply pattern is commonly used for inter-application communication.

response it gets from the receiver. The developer of the application programs this logic into the application.

request/reply communication

Request/reply communication is a form of synchronous communication where a sender makes a request of a receiver and waits for a reply before continuing to process.

This approach is used in situations in which the reply contains information that is necessary for the sender to continue its processing. Its one downside is that the sender must stop and wait for the response from the receiver. If the receiver's processing of the request takes a considerable amount of time, then the performance impact can be substantial and potentially unacceptable. If the receiver has a problem and is unable to comply with the request, then the sender might not be able to continue processing at all.

One-Way

The one-way pattern is the simplest form of the request/reply pattern because it sends a request and blocks only until it receives acknowledgment of receipt. An example is shown in Figure 3.2.

one-way communication

One-way communication is a form of synchronous communication where a sender makes a request from a receiver and waits for a reply that acknowledges receipt of the request.

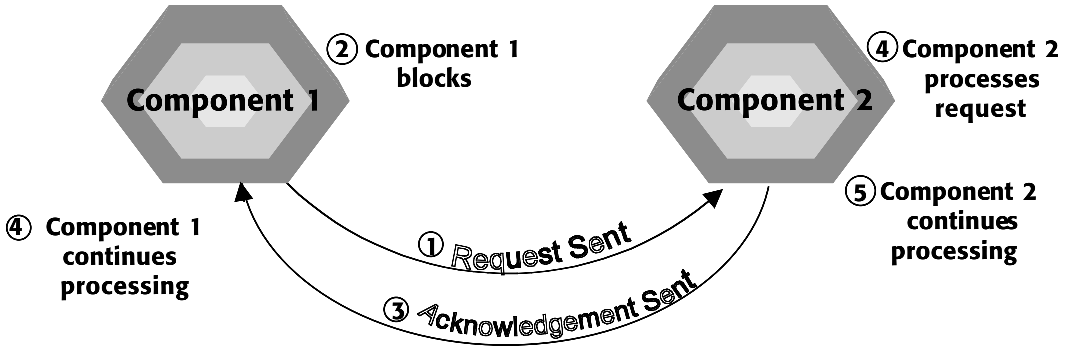

Component 1
② Component 1 blocks
Component 2
④ Component 2 processes request
⑤ Component 2 continues processing
④ Component 1 continues processing
① Request Sent
③ Acknowledgement Sent

Figure 3.2 The synchronous one-way pattern is an alternative form of synchronous communication.

This approach is used in situations where the sender must synchronize its processing with the receiver. The sender cannot continue processing until the receiver has received the request. The first drawback to this approach is there is no information other than the acknowledgment sent to the sender by the receiver. The second drawback is the performance implications of having the sender block. Receipt of acknowledgment should be much faster, however, than waiting for the processing to complete and then receiving a response.

Synchronous Polling

Synchronous polling allows a sender to continue processing in a limited manner while waiting for a reply from the receiver. An example is shown in Figure 3.3. In this communication pattern, the sender sends the request but instead of blocking for a reply it continues processing. It must stop and check for a reply periodically. Once the reply is received it may continue processing without polling. Of course, this case is useful only when the sender has something useful it can do while waiting for a response.

synchronous polling communication

Synchronous polling communication is a form of synchronous communication where a sender communicates a request to a receiver but instead of blocking continues processing. At intervals defined by the developer, the sender checks to see if a reply has been sent. When it detects a reply it processes it and stops any further polling for a reply.

This approach is used in situations in which the sender needs a reply but is able to continue processing while waiting for the response. It will be up to the developer of the sending application to determine the feasibility of applying this pattern. If an applica-

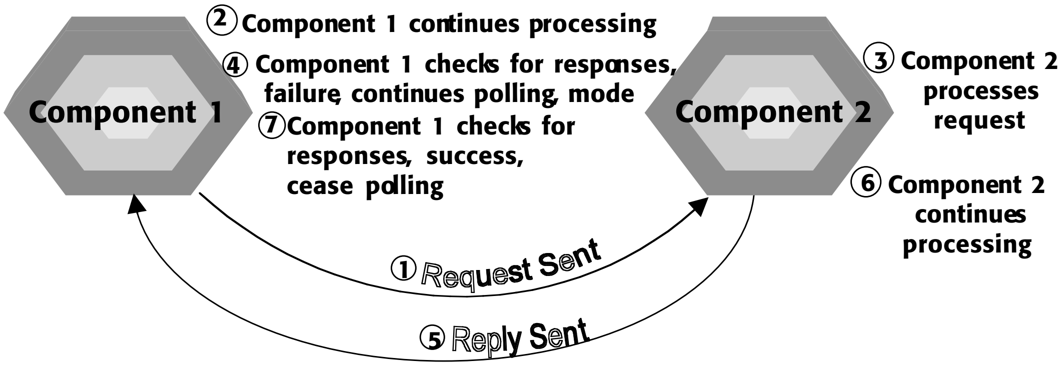

Component 1
Component 2
② Component 1 continues processing
④ Component 1 checks for responses, failure, continues polling mode
⑦ Component 1 checks for responses, success, cease polling
③ Component 2 processes request
⑥ Component 2 continues processing
① Request Sent
⑤ Reply Sent

Figure 3.3 The synchronous polling pattern is another form of synchronous communication.

tion fits this profile it will improve on the performance problems that exist in request/reply and one-way synchronous communications. It does have the drawback of being more complex to implement due to the requirement for polling.

Asynchronous Communication

Asynchronous communication does not require the sender and receiver to coordinate their processing with any communications. It provides a lower degree of coupling than synchronous communication. The sender communicates the request and then continues processing. It does not concern itself with when the message is received, how it is processed, or the results from the receiver as a part of the communication. It is used when the communication of information is required without the need to coordinate activities or responses.

asynchronous communication

Asynchronous communication occurs when the communication between a sender and receiver is accomplished in a manner that allows each of them to operate independently of the other. The receiver of the request is under no obligation to handle the communications or respond to the sender. The sender continues to operate once the request is sent without regard to how the receiver handles the communication.

There are three popular types of asynchronous communication:

- Message passing
- Publish/subscribe
- Broadcast

Asynchronous communication is useful when the purpose of communication is the transfer of information. Furthermore, it can operate in an unreliable environment where networks and servers may not be always available. These include applications where an update is sent from one application to all others that have copies of the same data or when a change occurs in a database. Another example is when an event is triggered that requires the notification of another application.

Message Passing

In message passing, component one creates and sends a message and continues processing, as shown in Figure 3.4.

This is the simplest form of asynchronous communication. In order to be effective, this communication mechanism must be implemented on a reliable network or with a

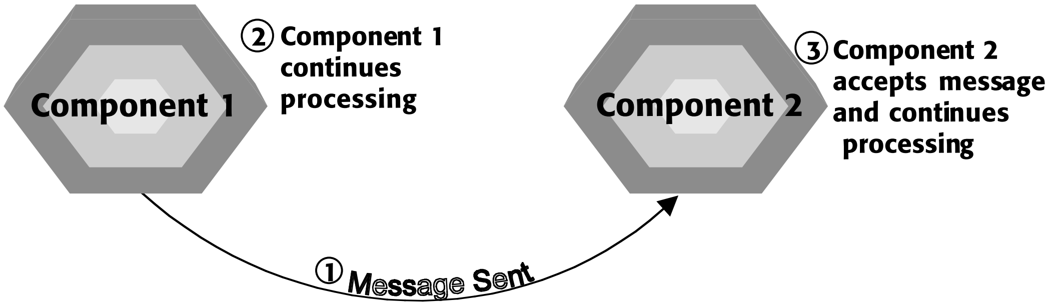

Component 1
② Component 1 continues processing
Component 2
③ Component 2 accepts message and continues processing
① Message Sent

Figure 3.4 The asynchronous message-passing pattern allows the sender and receiver to continue processing

service to provide guaranteed delivery. The guaranteed delivery service must be capable of continuing to try to send the request across the network and to the receiver until it is able to complete the communication.

message passing

Message passing is a form of asynchronous communication where a request is sent from a sender to a receiver. When the sender has made the request it essentially forgets it has been sent and continues processing. The request is delivered to the receiver and is processed.

This approach is used in situations where information needs to be transmitted but a reply is not required. When coupled with a guaranteed delivery service it can be used effectively on unreliable networks. Its primary drawback is that without the guaranteed delivery services a request may be lost due to a failure before it gets to a receiver.

Publish/Subscribe

Publish/subscribe allows you to determine the addressing of a request based on the interest of the receiver. The receiver subscribes to requests through a declaration of interest that describes attributes of a request it wants to receive. This type is shown in Figure 3.5.

publish/subscribe

Publish/subscribe is a form of asynchronous communication where a request is sent by the sender and the receiver is determined by a declaration of interest by the receiver in the request.

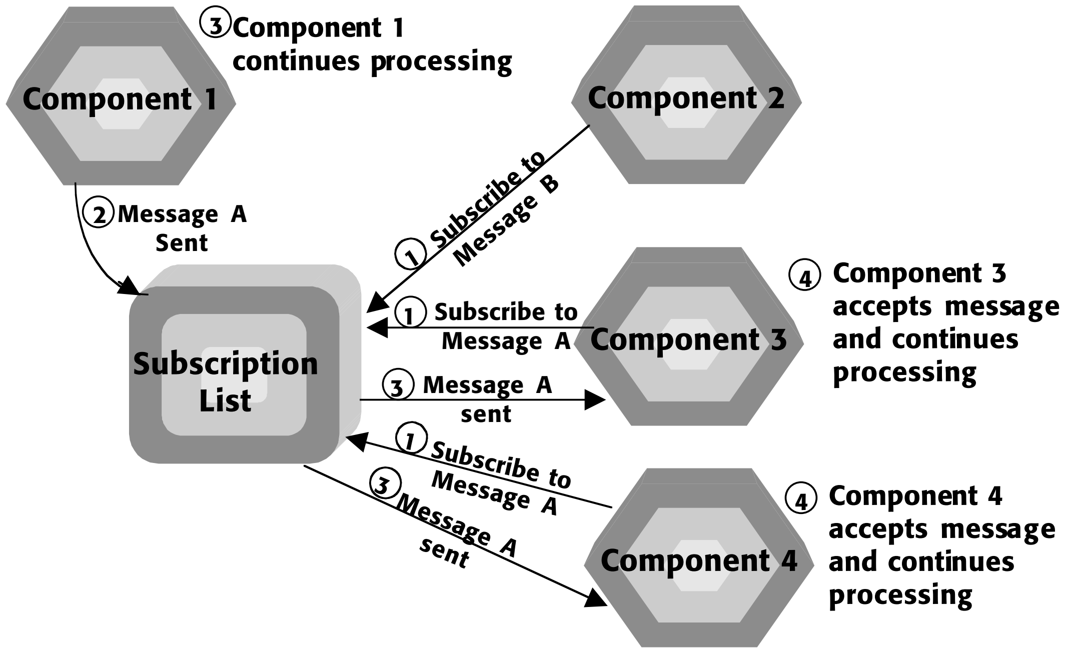


Figure 3.5 The asynchronous publish/subscribe pattern is very common in controlled information distribution.

Publish/subscribe allows each application in the system to decide about which events it wants to be notified. It does this by defining information, data structures, or types of requests it is interested in receiving. The developer of the receiving application defines this interest. This approach is used in situations where a reply is not required and the receiver is determined by the content of the request. In all of the other patterns we have discussed, address or location on a network determines the receiver.

This pattern is useful in a multistep process integration system where the request is actually the notification that an event has occurred—for example, notifying the order processing system when a product order has been shipped or a bill paid. The drawback is the design complexity of determining which receivers get a request produced by the sender and which do not.

Broadcast

In asynchronous broadcast communication, shown in Figure 3.6, a message is sent to every application in the system. The receiver decides if it is interested in the message. If it has interest then the request is processed according to logic programmed into the receiver by its developer; otherwise it is ignored.

The broadcast pattern needs to be used judiciously because it can become a significant performance bottleneck since each possible receiver must look at each broadcast message.

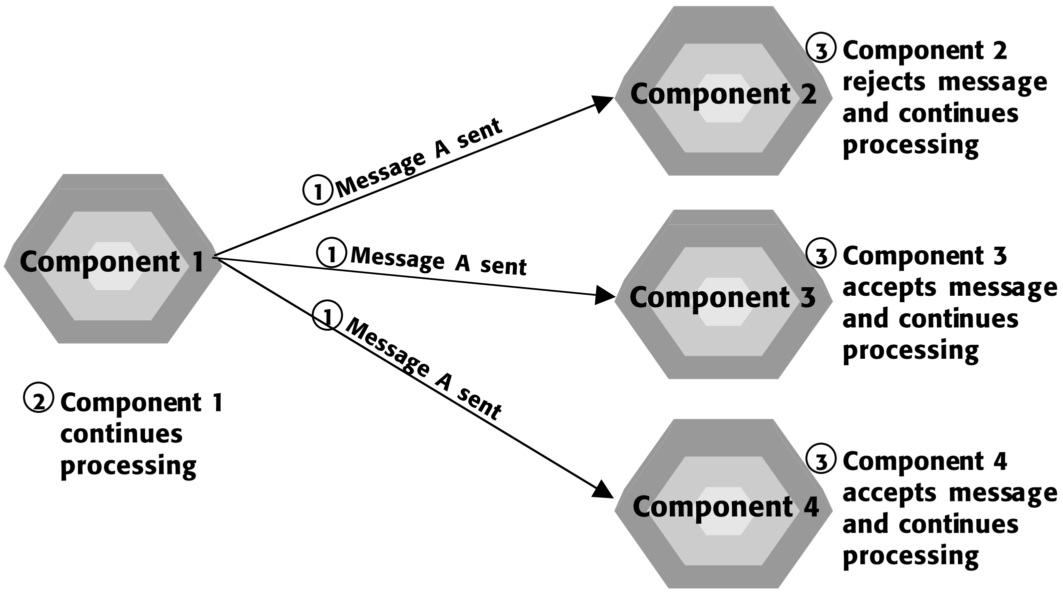

Component 1
① Message A sent
① Message A sent
① Message A sent
Component 2
Component 3
Component 4
② Component 1 continues processing
③ Component 2 rejects message and continues processing
③ Component 3 accepts message and continues processing
③ Component 4 accepts message and continues processing

Figure 3.6 The asynchronous broadcast pattern is a simple solution to information distribution.

broadcast

Broadcast is a form of asynchronous communication in which a request is sent to all participants, the receivers, of a network. Each participant determines whether the request is of interest by examining the content.

Methods of Integration

A method of integration is the approach used to construct a request from a sender to a receiver. There are two primary methods of integration:

- Messaging
- Interface definitions

Related to the method of integration is the concept of connectors or adapters. A connector, sometimes referred to as an adapter, is an interface into the application that defines the requests the receiver will accept while hiding the underlying complexity of accomplishing the integration. Whether you are using messaging or interfaces, connectors will be required to create the "plug" into the application through which requests are transmitted.

The choice of communications model and method of integration is not an either/or proposition. In fact, addressing the range of integration models described in Chapter 2, "Types of Integration," and achieving a robust EAI architecture require the existence of both communications models and potentially both methods of integration. The more flexible the EAI architecture, the greater the need to provide guidance on the application of each model or method.

Let's look at messaging and interface definitions in more detail.

Messaging

In a messaging integration approach, the sender constructs a message that contains the information on the actions desired as well as the data to perform the actions. The message contains both the control information and the data. This coupling of actions and data provides the receiver with the context of the data. For example, a message could contain the amount of $174.39 and an account number such as 017359. This data has very little meaning, but if an action is provided such as *deposit in checking* this information could be used by the application. Messaging offers an easy to understand approach for the construction, use, and processing of the data. The coupling of the data and control information is critical to their utility because it helps to lower the degree of coupling between applications. All the information required is in the message. The message, however, needs to be coded and decoded exactly the same way by all senders and receivers. Any deviation from the format will create confusion in an application because the application will misinterpret the information and actions.

The designers of the integration between applications must predefine the messages. Good EAI architecture should give the designer tools for accomplishing these designs. Any application that will send a message must be able to do the following:

Create the message in the appropriate format. For example, a message might contain information on sender, receiver, time stamp, account number, account action, and value. The sending application creates the message based on the definition of the message format.

Place the message into the communications system. For example, use of asynchronous communications service would transmit this message to the appropriate receiving application while the sender continues processing.

Once the message is in the communications system the receiver must be able to do the following:

Receive the message from the communications system. In this case the communications system alerts the receiver and passes the message into the application.

Parse the message into its control information and data. The account number, account action, and value are split out of the message and placed into appropriate variables.

Determine what to do with the message. The receiving application takes appropriate action. The designer of the receiving application must be able to read the message and then take the action he or she thinks is most applicable. The designer uses the description of the message format as the guide for the design.

Messages provide a lot of flexibility because the control information can be easily changed and extended. For example, if a bank decided to allow customers to change their names online after a legal change has occurred, such as for a marriage, then a control code for update name could be added to the update message. Another approach would be to create a new message called update name. It is important to note, though, that the message is independent from the application.

One problem with messaging is that it is not always visible to the designer which applications can respond to which messages. Message handling is hidden from view unless it is well documented or services provided to determine the linkage. Furthermore, if message management is poor reuse is not possible because there is limited visibility into the methods of integration.

Interface Definition

In an interface approach, the sender communicates through an interface, which defines the actions that can be invoked by an application. Any data to be processed is sent through the interface.

The differences between messages and interfaces as the method of integrating are subtle but important. Interface-based integration requires the specification and implementation of a well-defined interface that describes the actions that an application can perform. The interface is associated with an application. Messages are not associated with any application. In addition, with interfaces the actions that can be processed by any application are easy to read and explicitly stated. Messages, as described previously, inherently hide the applications that use them. The nature of interfaces usually requires less processing to decode than a message, and errors are discovered earlier in the development process. The bottom line is that messages provide a lower degree of coupling than interfaces at the cost of greater potential of errors and inability to reuse solutions.

The process that a component uses for integration with an interface parallels traditional programming structures. Interfaces make the application look like a procedure or an object using a language such as C, C++, or Java, to name a few examples. The process for using interfaces is as follows:

1. **Create an invocation or call to the component.** For example, call account_deposit with parameters account number and value.
2. **Execute the call.** For example, the asynchronous communications system makes the invocation. By using the asynchronous communication features the sender is able to continue processing.

The receiver must be able to do the following:

Receive a remote invocation or call. The communications system alerts the receiver through the interface and passes the parameters.

Execute the action based on the particular interface that is used. The receiving application takes the appropriate action.

This is exactly the model used in programming to make procedure calls or method invocations. In fact, if this is properly implemented the application may not realize that the call is from a different application and would view it as a request internal to the application. The difference is that the interface is externally visible and usable by any application on the network.

Over the long term, interface-based integration should be easier to reuse and maintain because the interface is explicitly defined and visible to the developer while messages can be hidden in the applications. It does not require looking at the code to see if an application will respond to a request. The interfaces are self-describing in terms of the actions that can be taken. With messages either documentation or code must be read unless a directory service is provided that maps message use by application. Interfaces, however, may be more difficult to change and extend depending on implementation. For example, depending on the design of a message, control information can be modified without having to change application code. Changes to an interface may require a compilation of any and all applications to make them effective. Finally, interfaces require some discipline in the definition of the interface to allow plug-and-play.

Connectors

Few applications have prebuilt message processing capability or interfaces. In this case the application will require an access point that allows either a message or invocation on an interface to be passed into the application. A *connector* or *adapter* is the access point.

connector

A connector is logic that is programmed into an application whose sole purpose is to provide access to the presentation, data, or functionality of the application in a structured manner. The connector hides the complexity of translating and communicating a message or an invocation on an interface for use by the application.

A connector is more than just an interface. It provides a significant amount of additional capability, such as the following:

- Error handling and validation checking to ensure the request is valid, detect any failures, or provide status of nonresponsive system components such as applications or networks. This information may be provided to the EAI software, the applications participating in the integration, or the network management software.
- Marshalling and unmarshalling of the data from the message or object. In some cases, marshalling and unmarshalling may include automatic conversions between the data types used on different machines.

- Conversion and transformation of data into formats that are acceptable to the receiving application. This would include conversion from EBCDIC to Unicode, changes to the formats of dates, or changing the value from dollars to yen.
- Managing state information to provide guaranteed delivery or providing for graceful recovery. Managing state information includes managing the pathways for integration and storing the information until a successful delivery has occurred.

marshalling

***Marshalling* is the process of converting sequences of parameters and complex data structures into "flat" strings of bytes that can be transmitted over a communications link. *Unmarshalling* is the process of correctly restoring the original structures at the receiving end.**

Because many applications do not come with connectors or were not considered for use by external applications, a convenient entry point into the application may not exist. In these instances, you may need to use data files, databases, user interfaces, or memory as the entry point for injection of the request. It is at this point that the correct integration model must be selected—presentation, data, or functional—to build the right connector. It is also important to ensure the request is valid and in the correct format before it is sent into the application. This is done by the designer of the application, comparing the message or interface format to the application's internal structures. Connectors and adapters are important to allow all applications to play in the enterprise.

Middleware Choices

Middleware is an overused buzzword and is usually touted as the silver bullet for rapid application development and system integration. For the purposes of this book we will view middleware as a technology intended to allow disparate technology-based systems to interconnect. EAI architectures are based on middleware.

There are five basic types of middleware in the market today:

middleware

Middleware is a type of software that facilitates the communication of requests between software components through the use of defined interfaces or messages. In addition, it provides the runtime environment to manage the requests between software components.

- Remote procedure calls
- Database access middleware
- Message oriented middleware
- Distributed object technology
- Transaction processing monitors

Each of these types of middleware was invented to solve a different problem related to the movement from monolithic applications that operated on a single computer to distributed applications that needed to connect.

The communications model and method of integration are closely tied to the middleware that is employed. A specific middleware product will embed one or more communications models and methods of integration. Not all middleware, however, is applicable to EAI. It is important to understand the different types to select the right EAI tools and to apply them to the right kind of problems.

It is worth noting that EAI technology is also considered middleware but at a higher level of middleware than these five basic types. EAI extends middleware to reduce the integration work that needs to be accomplished. These five middleware types represent a basic level of infrastructure capabilities.

Remote Procedure Calls

Remote procedure calls (RPC) is a procedure call with a longer "wire." It allows procedure calls to be sent across a network without any knowledge of the network by either the sender or receiver.

remote procedure calls (RPC)

RPC is a type of middleware that is based on the notion of developing distributed applications that integrate at the procedure level. It created the ability to make procedure calls across a network.

RPC has been around since the 1970s. Its popularity peaked in the late 1980s when the Open Software Foundation attempted to create a standard around the Distributed Computing Environment. The importance of RPC dwindled as the use of object technology rose. Because the RPC model was closely associated with a procedural model of programming it was supplanted by other types of middleware, such as distributed object technology and message oriented middleware.

RPCs have had significant influence on middleware in areas such as interface definition languages. The work on interface definitions was used by most of the distributed object technology middleware including CORBA and COM/DCOM.

RPC technology is still used in many enterprises; however, it will not form the underpinning of any modern EAI architecture. It is declining due to the procedural emphasis of this technology, the rise of object technology which uses a better model, and the difficulty of applying RPC beyond the software development paradigm and into existing software integration. It is still used but only in development activities that use procedural languages such as C that do not require legacy integration.

Database Access Middleware

Database access middleware provides the ability to access remote data files and databases. It was developed during the client/server era of computing in the late 1980s and early 1990s. Enterprises were being driven to provide increased access to information.

database access middleware

Database access middleware is a type of middleware that is based on the notion of accessing distributed data whether in files or databases. It integrates at the data level and allows data queries or data movement to occur across a network.

Each database vendor developed its own middleware solution to the problem of distributed data access. These proprietary solutions allowed new applications to talk with the vendor's proprietary database but did not provide a heterogeneous solution. Even though the Structured Query Language (SQL) was a standard language to query databases, the access to the database was not standard. Early data access middleware provided a proprietary mechanism to access the data. Recently standard mechanisms such as ODBC and the related JDBC, the Java-based equivalent, technology have provided standard mechanisms for access.

Open Database Connectivity or ODBC has become a popular standard for accessing databases. ODBC is a standard interface definition for accessing a database. In the same way that SQL is a standard and yet vendors build significant proprietary extensions, ODBC still competes with a variety of proprietary solutions, such as the interfaces provided by vendors such as Oracle. ODBC has been extended to support Java access as the JDBC interface.

ODBC

ODBC stands for Open Database Connectivity. It is a standard interface originally intended for relational database management systems. It has been applied to other sources of data.

Database access middleware provides the underpinnings for solving the data integration problem described in Chapter 2. It allows you to directly access data files and databases through the middleware, bypassing the presentation and functionality of an application. Database access middleware is effective access to different sources of data. It allows for the rapid development of connections to data. It can be easily duplicated using distributed object technology, message oriented middleware, or transaction processing monitors. Solutions have been built that are completely ODBC- or JDBC-based or are database gateways that implement the proprietary interfaces provided by the vendors. Because the focus of EAI extends beyond data access, data access middleware is not appropriate as the core of EAI architecture. It might be an adjunct, however, if the core of the EAI architecture does not support data access.

Message Oriented Middleware

Message oriented middleware (MOM) provides the ability to integrate diverse applications through the use of messages. MOM technology is based on the concept of a message as the method of integration. As described previously, a message is a combination of control information and data. The simplicity of this technology is one of the key reasons for its popularity.

Message Oriented Middleware

Message oriented middleware is a type of middleware that uses messages as the method of integration; it provides the ability to create, manipulate, store, and communicate these messages.

MOM is often closely associated with EAI due to the popularity of IBM's MQSeries for solving mainframe integration. The early success of MQSeries and related products helped to define the EAI market. In fact, MQSeries represents only one type of MOM, a queuing pattern. MOM technology became popular in the 1990s as organizations realized that the mainframe computer was not going away and they needed technology to make old applications work with new ones.

Many EAI tools, such as IBM's MQIntegrator, Saga's SagaVista, and Talarian's Smart Sockets, are based on MOM. It is well suited for data consistency and multistep process problems but is not effective in component integration because messages can be easily defined for updates. Furthermore they can be generated and received by all applications that manage databases with that information. Messages can also be defined and generated for events that occur in applications and sent to receiving applications for coordination and further action. Component integration implies reuse as well a plug-and-play. Messages as methods of integration are not suited to this use because the messages are not as easily visible as interfaces to the developer of new applications. MOM technology as applied to EAI is described in detail in Chapter 4, "Messaging Architecture and Solutions."

Distributed Object Technology

Distributed object technology (DOT) is the latest entry into the middleware market. It is similar to RPC middleware, but it is based on an object-oriented paradigm. DOT provides the ability to create object-oriented interfaces to new and existing applications that are accessible from any other application. Because most new applications are based on object-oriented languages such as Java it is a required element in any EAI solution.

Distributed Object Technology

Distributed object technology, or DOT, is a type of middleware that extends the concepts of object-oriented technology to distributed processing. Interfaces are developed for applications that make software look like objects.

By applying DOT an application makes an invocation on any object without being aware of its location. This allows the software components to be moved, replaced, or replicated without affecting any other components. Like RPCs, interface definition languages and synchronous communications are critical elements of a DOT solution. The interface definition language allows for the creation of a distributed object while synchronous communications allows for the transmission of invocations to the interfaces. Recently, DOT has been extended to include asynchronous communications. This was done to allow greater flexibility in using DOT to solve different integration problems.

DOT is required in order to achieve good component integration. DOT is well suited to the creation of component-based systems. The interface definitions are explicitly declared so the designer is able to clearly understand how to use them. The down side of DOT is its complexity, compared to MOM technology, because it requires a higher degree of coupling between applications.

Although there are many problems that DOT will solve, a good EAI architecture will combine MOM and DOT technology to be able to solve the broadest set of problems. DOT technology as applied to EAI is described in detail in Chapter 5, "Object Architecture and Solutions."

Transaction Processing Monitors

Transaction processing monitors (TPMs) are a critical technology to any large, transaction-oriented enterprise. TPMs ensure that a transaction maintains ACID properties (atomicity, consistency, isolation, durability) throughout its life. TPMs were first widely employed on mainframe computers in the form of IBM's IMS and CICS products. In the 1980s there was a push to create distributed TPMs that were based upon standards such as the distributed transaction processing standard defined by X/Open Ltd. This standard includes the XA interface which is the most widely supported portion of the standard. Products such as BEA's Tuxedo form the TPM middleware.

Transaction Processing Monitors

Transaction processing monitors are a type of middleware that preserves the integrity of a transaction. They support features such as rollback, failover, auto restart, error logging, and replication to eliminate single points of failure.

TPMs allow a transaction to be formed by the sender and then ensure that it gets to the right place, at the right time, and completed in the right order. This may appear to be a simple proposition, but TPMs are the most complex of all the types of middleware.

ACID Properties

ACID properties are properties of software that are desirable in the processing of transactions. They are atomicity, consistency, isolation, and durability.

Few EAI tools use TPMs as their base technology. Recognizing the complexity of some integration problems, TPM functionality has been integrated into both MOM and DOT technology. TPM technology as applied to EAI is described in detail in Chapter 6, "Transaction Architecture and Solutions."

EAI and Middleware

In reading the next three chapters you will learn about these three types of middleware in detail. A robust EAI architecture requires that MOM, DOT, and TPM technologies are integrated into a consistent framework. No EAI product in the market will address all of the needs of an enterprise in building a robust architecture. An enterprise will need to integrate across middleware to get a full EAI architecture.

Service Building Blocks

The communications model, method of integration, and middleware are the core of any EAI solution. Services are the final critical element. *Services* are extensions to a basic

services

A service is a functional extension to basic communication or middleware capability.

communication or middleware capability that enhance it in some manner. Services are not a part of the core, but they can significantly aid the architect and developer in implementing a solution. Services are intended to reduce the burden of applying the core technology. They can also enhance the core to meet certain desired characteristics such as security or reliability. The range of possible services varies widely.

The following are the services we consider important to a successful EAI-enabled enterprise:

Directory. Tracks all the components and key information about the system. It is used to automate the action of locating any element. Furthermore, it can be used to catalog and manage interfaces or messages as well as metadata.

Lifecycle. Aids the developer by automating the creation of any objects or messages as well as ensuring that they are properly managed and disposed of on completion.

Security. Often the forgotten service. Everyone wants it but ends up ignoring it because of the complexities involved in doing it right. A good security service will provide all of the capabilities required to secure any integration, specifically authentication, authorization, and secure communications.

Conversion and transformation. Representation of data is a problem in any enterprise. Data exists in many formats with different definitions. It is necessary to be able to convert and transform data into the correct format to properly complete any integration.

Persistence. Ensuring that state information and data are safely stored is critical to ensuring that information is not lost. The persistence service provides the capability to save state information or data.

Events. The ability to identify and track events, which is a very useful service in an EAI solution. The ability to identify when a particular problem or unique event occurs is what this service will provide.

Notification. Once an event is detected the notification service will alert the interested component that the event has occurred.

Workflow. Managing a set of requests or messages across a series of components in a prescribed order as a single action. This capability is also known as process management to distinguish it from the original generation of workflow systems that concentrated on automating paper flows among people.

Certain services are almost always required—directory and lifecycle—because of the nature of distributed systems. These services aid in locating and performing requests. In the case of the other services, the selection of a particular service is based on the use of the EAI architecture. For example, security is dependent on how the EAI architecture will be used. If the requests are sent across a public network there might be concern over some one intercepting the data.

Any good EAI solution will support most, if not all, of these services. Services are an important building block to be considered in EAI tool selection. While you may not need all of the services you should be able to extend your EAI solution to support all these services to be able to handle all of the different models described in Chapter 2.

Final Thoughts

There is no one right answer to constructing an EAI strategy and architecture. A long-term EAI strategy will ensure that there is a robust architecture that addresses all the elements of the building blocks and fits them together into a consistent architecture. You have learned about the basic building blocks. We will now turn to the three critical middleware technologies, MOM, DOT, and TPM, in the next three chapters to help understand the details of the technology. This sequence is followed by a discussion of how to architect a solution. Keep these building blocks in mind as you read each of these chapters.

CHAPTER 4

Messaging Architecture and Solutions

Key Facts

- Messaging is a critical component in EAI.
- Messaging products vary in type and the benefits they bring to businesses.
- Sophisticated messaging will include request/reply in a single transaction.
- Message queue management and routing are key to handling a high throughput of different message formats.

Messaging is a part of the proliferation of middleware software that enables the client/server interconnection of client-based applications and packages, server-based applications, and packages and legacy systems. Simple messaging is the movement of data from one software location to another. Many refinements have been added to this simple concept to make it a valuable business support tool for use in information systems. Message Oriented Middleware (MOM) technology provides an asynchronous communication mechanism that uses messaging as the communication mechanism.

This chapter covers the primary concepts of message queuing. Basic message queuing introduces the essential concept. Request/reply messaging describes how messages are part of interapplication transactions. Multiple queuing and routing and queue management deal with managing messaging services. Message translation covers the need to convert data when passing it between applications. Publish/subscribe messaging discusses a practical use of messaging.

Message Queuing

Message queuing is the ability to queue data for passing between applications and the implementation the message passing.

There are various forms of message queuing:

- Point-to-point message queuing, where the message transmission is between two prespecified servers or application nodes
- Broadcasting, where all other clients or servers connected to are simultaneously sent the same message
- Quality of Service, where CORBA is used to implement messaging and the Trader Service can be used to find a server with the required Quality of Service features to be messaged rather than a specific server to be identified in advance

The basic principles of message queuing are shown in Figure 4.1.

message queuing

Message queuing is a way to pass data or make an invocation of a remote function from a specific client to a specific server, or from a server to a server.

In message queuing, a client produces a message containing some data and feeds it to a message queue. A client locates a queue that suits the type of message it needs to send, assembles the message into the correct format, and puts the message into the queue. The queue feeds a server, which consumes or *handles* the message. The message handling may be predefined at design time or defined within the message itself. More specifically, the client needs to locate a queue that suits the type of message it needs to send, assemble the message into the correct format, and put the message into the queue.

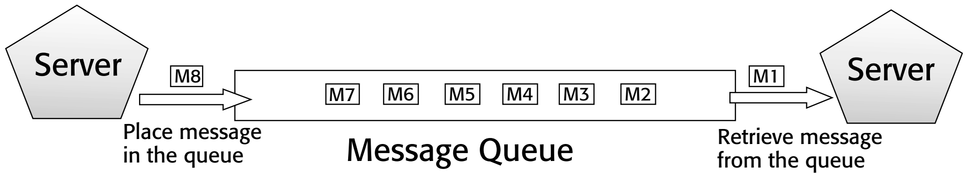


Figure 4.1 All variations of message queuing follow the same basic principles.

In a simple messaging implementation requests and replies to the requests are separate transactions. A request message transaction will be fulfilled when the message from the source has been received by the target. A reply message transaction will be fulfilled when the message from the target has been received by the source. To fulfill a request/reply transaction requires additional code so that the request is responded to appropriately. Sophisticated messaging implementations use request and reply in the same transaction in a seamless manner.

Request/Reply

Request/reply messaging is an example of point-to-point messaging. Request/reply messaging requires the queue to handle the response for each specific reply, as shown in Figure 4.2. The queue can handle the request either synchronously or asynchronously. If the request is handled synchronously, the requesting software must idle until a reply is received. If the request is handled asynchronously, processing that is not dependent on the reply can continue.

request/reply messaging

The request message from a source and the reply message from a target will occur within a single transaction.

It would seem that asynchronous processing would be preferable. There are times, though, when the synchronous mode of communication is preferred and may even be an application requirement. Synchronous processing is required where an immediate response (reply message) is critical to the transaction. Consider a user request to open a data file. Asynchronous processing is useful when message transactions can occur without being constrained by the sequence of other message transactions, such as running a process in the background and being able to continue with other work.

This method of messaging usually processes messages in a queue in the order in which they are placed. This is called *First In First Out* (FIFO). The order can and does vary; messages and queues can be assigned priorities that can lead to a complex set of rules for governing message queuing.

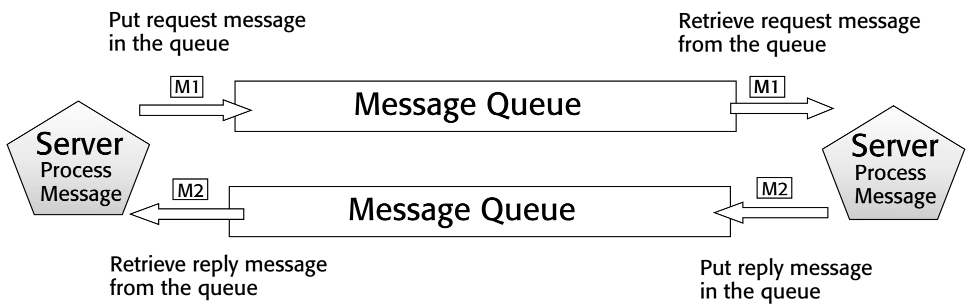

Put request message in the queue
Retrieve request message from the queue
M1
Message Queue
M1
Server Process Message
Server Process Message
M2
Message Queue
M2
Retrieve reply message from the queue
Put reply message in the queue

Figure 4.2 Request/reply message queuing requires the queue to handle the response for each reply.

FIFO

FIFO stands for First In First Out processing. The first message pushed into a queue will be processed first and pushed out of the queue.

Publish/Subscribe Messaging

Publish/subscribe messaging serves a different purpose from point-to-point messaging and is a form of broadcast messaging. In the publish/subscribe mechanism, a client subscribes to a particular information set based on specified criteria. When the server finds a match to the subscription criteria it publishes the information to the subscribing client (see Figure 4.3). Table 4.1 shows examples of criteria that could be subscribed to with the subsequent publication data that is returned when the subscription criteria are met.

Other publish options for the publish/subscribe approach are to broadcast messages to all clients or to send the message as clients become available. A common subscriber list for the same data benefits from broadcasting the publication of information, saving processing overhead. This is useful to support fixed organizational structures that depend

Table 4.1 Publication and Subscription Criteria

SUBSCRIPTION CRITERIA	PUBLICATION CRITERIA
Announcements by Microsoft Rise in Microsoft share price of > 5 cents	Bill Gates announces retirement 30 cents on the dollar increase
Drug discovery announcement by named competitors	Competitor announcement Change in competitor share price

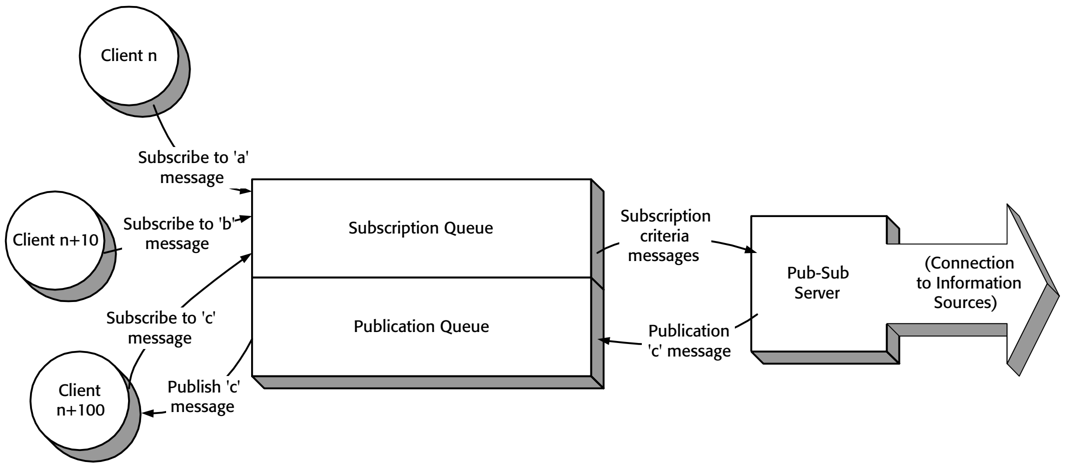


Figure 4.3 Publisher/subscriber messaging is a mechanism for providing required information.

on getting the same information at the same time. The single messaging approach is the default practice when dealing with individual subscriptions.

Message Translation

A message must be constructed before it is placed into the message queue. Once it's in the queue, it then must be translated back into a usable form by a translation, transformation, or conversion service, as shown in Figure 4.4. *Message translation* can be performed at several points, but it is usually performed by a server so that the client does not have to concern itself with a particular message format for a particular type of queue.

message translation

Message translation is the ability to convert data into a form that can be transmitted in a message and the subsequent ability to convert the data back to its native format once the message has reached its target.

Depending on the sophistication of the message queuing service, message translation may happen at fixed points or en route. Translation that occurs at fixed points enables you to distribute the translation software to a high-performance, high-throughput computer and is beneficial when the message flows along a set route. En-route translation guarantees that the translation will occur prior to message delivery but relies on a mobile translation service. This can take advantage of spare computing power, but the route will vary, such as happens using the Web.

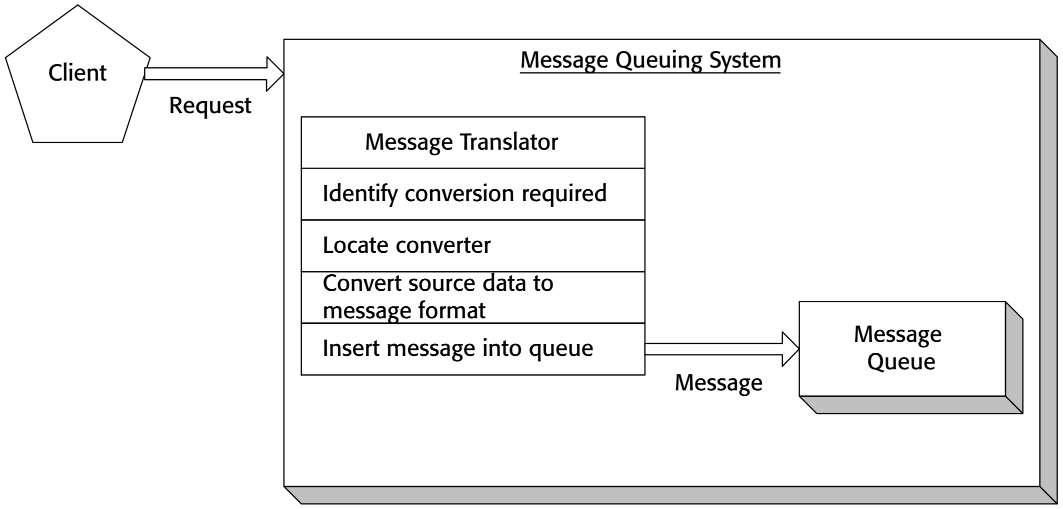

Client
Request
Message Queuing System
Message Translator
Identify conversion required
Locate converter
Convert source data to message format
Insert message into queue
Message
Message Queue

Figure 4.4 A message must be constructed and then translated to a form usable by the server.

The messaging service may also be able to dynamically discover available translators. This is a Quality of Service mechanism, like the OMG Trader Service, that enables the location of the appropriate translation service to be identified. This enables more flexible systems with runtime intelligence rather than design-time intelligence.

Queue Management

When there are multiple queues, management of the entire queuing system is critical (see Figure 4.5). The dependencies on a specific queue will often vary, and the queue management must be able to appropriately support the critical dependencies. The dependencies include the following:

Synchronicity. The ability to synchronize functions as required to achieve high performance.

Response time. The time taken for replying to a message.

Message content. The format and nature of data structures within a message.

Message size. The physical size of a message in bytes, Kb, Mb, or Gb.

Message priority. The importance of message delivery for a specific message (high, medium, low).

Queue volume. The number of messages that a queue can contain.

Queue timeouts. How long a queue can sustain the messages it contains.

Queue persistence. The ability of a queue to store messages in a persistent form.

Queue priority. The importance of one message queue compared to another.

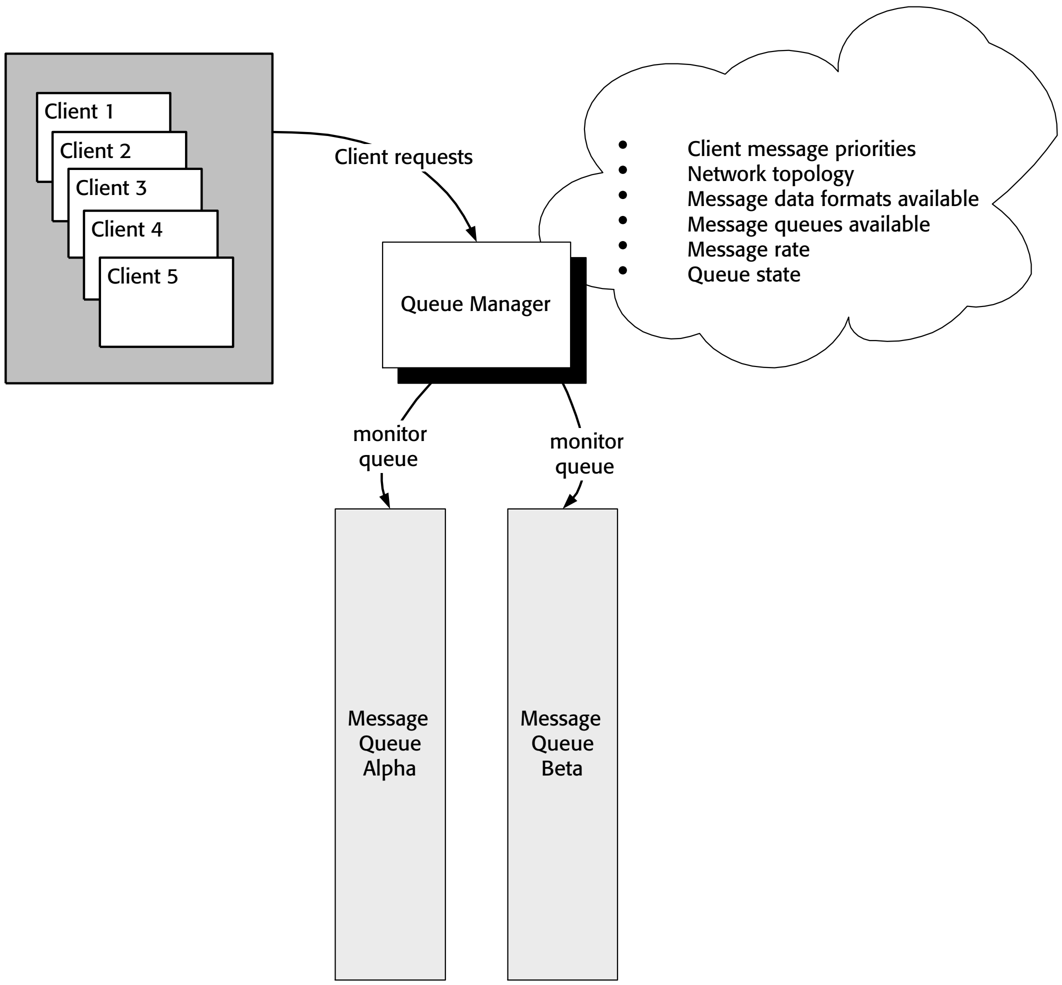


Figure 4.5 Managing multiple message queues is a critical task to implement the required messaging strategy.

Topology of queue system. The queuing network of queue interconnections and communication paths.

Queue Persistence

Another factor that must be taken into account and controlled by the queue manager is whether a message requires persistence and under what conditions the persistent version of the message can be destroyed. Persistence is the ability to temporarily store the message until delivery.

persistence

Persistene is the ability to store data and retrieve data. Transitory persistence is for the length of time required for one or more functions to complete, such as a transaction.

Normally when the message queue is part of a transaction, even in the case of a simple request/reply situation, the message will need to be persisted to ensure guaranteed delivery. For example, the message will need to be stored in a persistent mechanism that will guarantee that it is retrievable if the following conditions occur:

- The queue fails and the message never arrives
- The request message is corrupted
- The reply message is corrupted
- The recipient server fails before processing the message or sending the reply
- The requesting client fails before receiving the reply

This usually means storing the request message and the reply message until the requesting software has acknowledged that it has successfully received the reply, as shown in Figure 4.6.

In addition to performing persistence on a message-by-message basis, the entire queue may be persisted with individual messages indexed. The nature of persistence will vary from product to product. Relational databases and object databases are both common solutions to message persistence. Some products provide a set of generic database adapters so a range of database types is available for selection. The database or other persistent mechanism will underlie the ability to guarantee message and queue recovery.

Multiple Queuing and Routing

Sophisticated messaging systems often implement multiple queues and support variable routing across networked nodes to enable high throughput and performance. Reality dictates that many systems are complex; therefore interconnecting these systems would require multiple queues that can handle different types of messages (see Figure 4.7) or provide greater bandwidth for connectivity (see Figure 4.8).

While the need for multiple queues does not require a message queue locator, it would be inefficient for each client to have hard-wired locations of all message queues in a complex system. If the choice of the queue is one of bandwidth, then the client should not be expected to provide such queue management functions. This introduces the need for a *queue manager* or *router*. With the queue manager, the client needs to know only the location of the message manager software that contains the router(s) or the name of a queue, which in modern systems can be self-advertising. The idea is that the less hard-

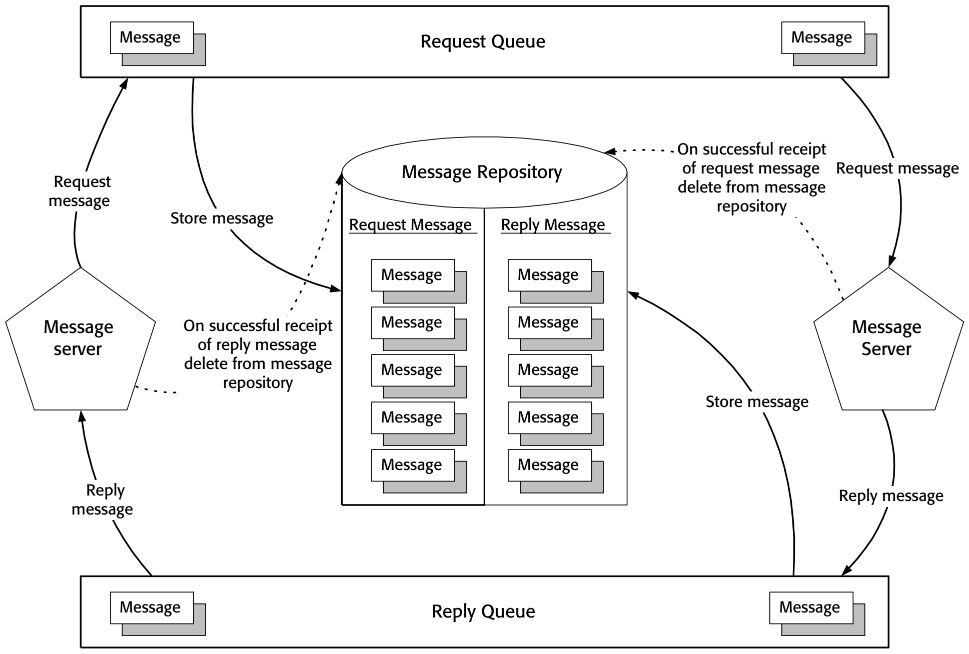


Figure 4.6 Persisting a message ensures that it cannot be lost.

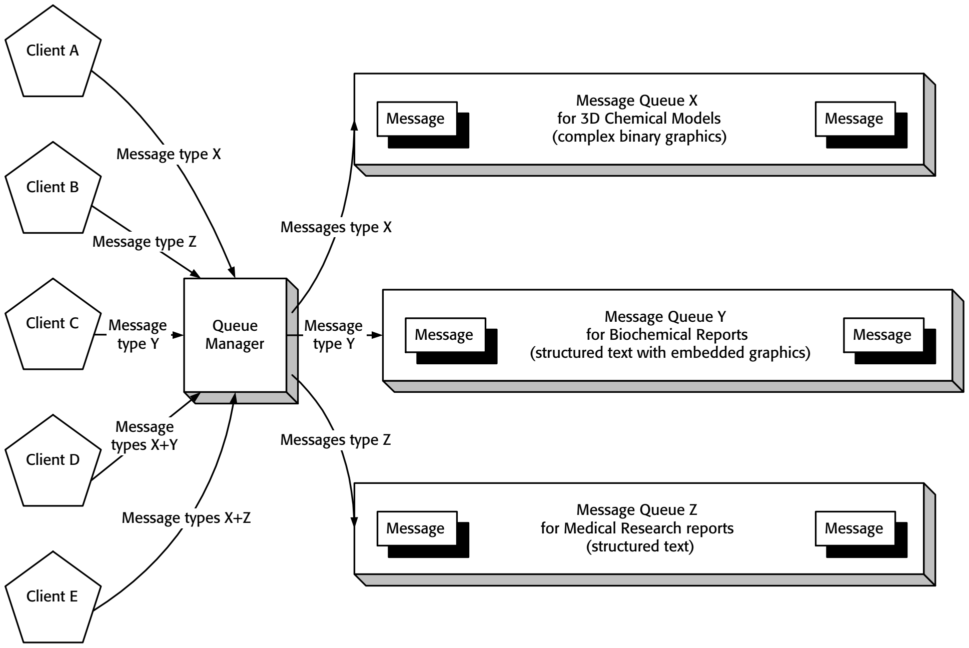

Client A
Client B
Client C
Client D
Client E
Message type X
Message type Z
Message type Y
Message types X+Y
Message types X+Z
Queue Manager
Messages type X
Message type Y
Messages type Z
Message Queue X for 3D Chemical Models (complex binary graphics)
Message Queue Y for Biochemical Reports (structured text with embedded graphics)
Message Queue Z for Medical Research reports (structured text)
Message

Figure 4.7 Complex systems may require multiple queuing for multiple message types.

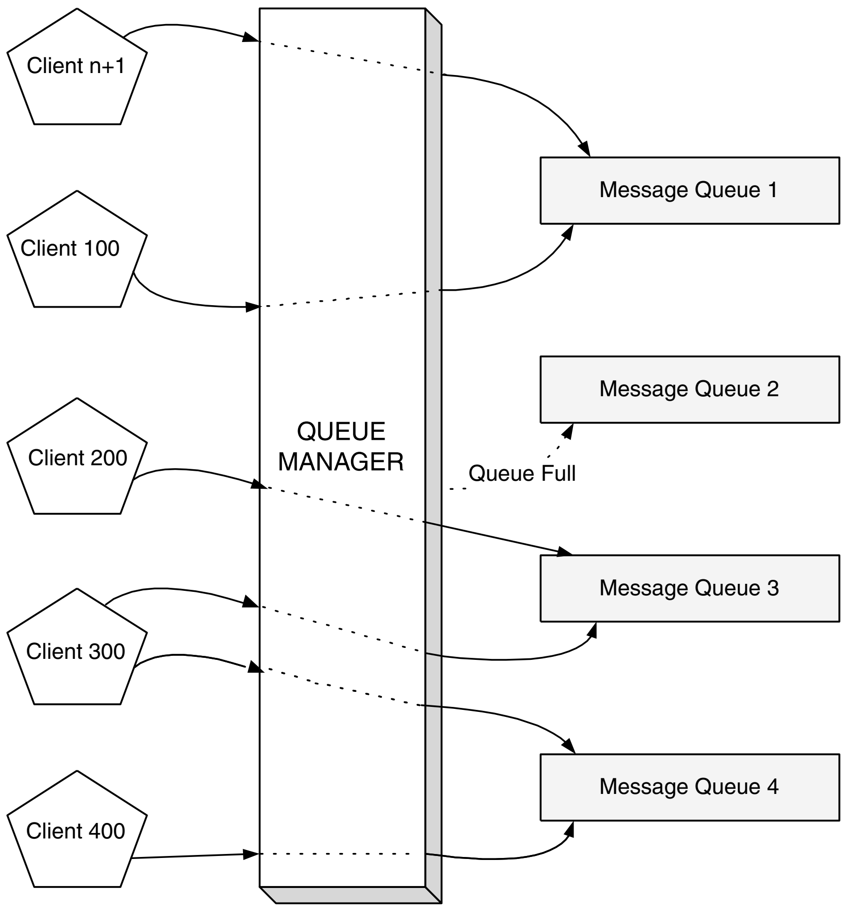


Figure 4.8 Complex systems may require multiple queuing for greater bandwidth.

wired information needed by a client for a server-side service, the more flexible the software. The router determines the most efficient route to get the message from client to server, or vice versa, not the sender.

queue manager/router

Queue manager/router is a service that can determine to which queue a message should be routed, based on message content, queue availability, or preset rules.

Multiple queues are highly useful when they are data type specific—for example, a queue that handles messages containing tables and a queue that handles messages containing MIME objects. If many data types are to be embedded in messages this is a way of localizing the supporting message queue. Otherwise, all messages have to be put into and extracted from a single form, such as BLOBs (binary large objects) because the size and type complexity of the message content may cause significantly variant processing rates and required message queue services.

Message Queuing Products

There are many messaging products in the marketplace, and they offer different levels of integration. Also many nonmessaging technologies, such as CORBA and DCOM/COM+, can be implemented in a manner to provide messaging. The messaging marketplace is still evolving and because many vendors' products in the messaging arena offer additional, nonmessaging services it is an unclear market.

Several levels of messaging occur at growing levels of abstraction and complexity, in much the same manner as the ISO-OSI network model that is, a layered model where each layer has a responsibility for offering a discrete service dependent on the layer below. The messaging layers are as follows:

Data Transport. The simple movement of data within a message from source to target with associated processing, such as data conversion, occurring.

Data Integration. Messaging with a conversion service, so that other applications can immediately and usefully consume the data. Request/reply is a single transaction.

Application Integration. Adds the ability to handle multiple messages and message formats to make integrating applications easier.

Enterprise Integration. Adds enterprise services that support the messaging, such as automated message routing and queue management.

The following article was written by Nancy Nicolaisen for *Distributed Computing* (Volume 2, Issue 6) in 1999 and is a useful review of messaging middleware products. It covers the range of messaging levels and gives the reader a useful baseline for evaluating the plethora of other messaging products available. The article is the copyright of Distributed Computing and is reprinted with permission. Table 4.2 provides a direct comparison of the messaging technologies.

Middleware: The Message Is the Medium

We all know about layered software architecture: the application layer, the session layer, the transport layer. Message oriented middleware (MOM) is the magic layer. It's better than a wand, less conspicuous than a fairy godmother, and vastly more flexible than a genie in a lamp. Middleware's magic exposes itself as a predictable set of miracles performed on demand, around the clock, unconstrained by the limits of platform, protocol, and even power failure. What makes MOM both exceptional and exquisite as a technology is that it almost imperceptibly overlays existing networks, application

Table 4.2 Comparison of Messaging Technologies

VENDOR PRODUCT	IBM MQSERIES	LEVEL 8 SYSTEMS FALCONMQ	PEERLOGIC, INC. PIPES PLATFORM	PRIMEUR GROUP SPAZIO FILE TRANSPORT	NETSYS VCOM	TALARIAN SMART SOCKETS
Message Model	Request and reply	Symmetrical Microsoft Message Queue technology across heterogeneous environments	Resource/Agents model oversees message routing transport	File Transport with Guaranteed Message Delivery	Synch/ asynch interprocess communication to integrate easily with existing applications.	Publish and scribe
Failover Strategy	Dynamic workload distribution with automatic failover	Tightly integrated with underlying NT and MS Transaction Service services; sliding window communication protocols & recoverable storage	"Self-healing" technology immunizes PIPES systems against single point network failures	Detects transport failures and triggers restart	Failover routing configurable on a per-node basis	"Hot failover" technology continuously and dynamically monitors transport status; reroutes past failures

(*Continues*)

Table 4.2 *(Continued)*

VENDOR PRODUCT	IBM MQSERIES	LEVEL 8 SYSTEMS FALCONMQ	PEERLOGIC, INC. PIPES PLATFORM	PRIMEUR GROUP SPAZIO FILE TRANSPORT	NETSYS VCOM VCOM	TALARIAN SMART SOCKETS
Areas of Market Strength	Real-time financial transaction processing; messaging model well suited to mobile applications where servers may disconnect from transport layer	Seamless integration with Microsoft Messaging technologies in heterogeneous environments	Inherently supports large-scale multitier applications with asynchronous, nonblocking messaging	Flexible, efficient reliable bulk file transfer across heterogeneous environments; integrates easily with other messaging products and technologies	Grew out of process control automation: complex time-manufacturing operatons	Highly efficient network bandwidth utilization; fast and reliable in applications where servers are are permanently connected
Contact Information		info@level8.Com www.Level8.Com LEVEL8 SYSTEMS 1250 Broadway 35th Floor New York, NY 10001 Tel: 212.244.1234 888.538.3588 Fax: 212.760.2327	info@peerlogic.com www.peerlogic.com PEERLOGIC, INC 555 DeHaro Street San Francisco, CA 94107-9586 Tel: 415.626.4545 Fax: 415.626.4710	primeurtechint @ primeur.com www.primeur.Com PRIMEUR GROUP Galleria Mazzini 1/8 16121 Genova, Italy Tel: +39.101.53451 Fax: +39.010.5354232 Tel: +46.31.720.6000	info@netsys.net www.netsys. net/vcom NETSYS TECHNOLOGY GROUP AB Nÿmndemansgatan 3 S- 431 85 Molndal Sweden Fax: 650.965.9077 Fax: +46.31.27.5010	info@talarian.com www.verimation.se www.talarian.com TALARIAN CORPORATION 333 Distel Circle Los Altos, CA 94022-1404 Tel: 800.883.8050

The article was written by Nancy Nicolaisen and is the copyright of Distributed Computing (Volume 2, Issue 6) and is reprinted with permission.

architectures, and platforms. It enables communication and collaboration between synchronously or asynchronously executing processes without imposing standards of behavior on any of the other layers.

The leaders in the MOM tools niche arose from a wide variety of enterprises, and their tools bear the distinctions of these origins. Here is a brief guide to standout products and their vendors.

SPAZIO File Transport

PRIMEUR GROUP

Galleria Mazzini 1/8

16121 Genova, Italy

Tel: +39.010.53451

Fax: +39.010.5354232

Primeur Group's SPAZIO File Transport is useful both as a stand-alone file transport mechanism for heterogeneous environments and as a file transport layer integrated with an underlying messaging and queuing architecture. In the second capacity, SPAZIO File Transport may be coupled either with a complementary element of the SPAZIO product line, SPAZIO M&Q, or with competing messaging and queuing solutions, such as IBM MQSeries and Microsoft Message Queue.

SPAZIO File Transport relies on store-and-forward file handling, although it can be configured to send files that bypass the queuing mechanism. Transport options are highly flexible and include support for splitting files into pointer/data configurations for optimized multicast file transport operations and simultaneous transmission over multiple channels. Sophisticated receipt and restart ensure against lost and duplicated transmissions. The product flexibly moves files among a variety of platforms, transparently spanning the protocol, data organization, and character set differences between sender and recipients.

SPAZIO's built-in event management and scheduling tool provides a fine-grained means of file dispatch control, allowing developers to defer file transfers, schedule them, or trigger transfers on notification of application-defined events. Integrated management facility includes tools for transmission management, support for integration with third-party SNMP management tools, Openview and Tivoli compatibility, and embedded logging and tracing capability.

FalconMQ

LEVEL8 SYSTEMS

1250 Broadway, 35th Floor

New York, NY 10001

Tel: 212.244.1234 / 888.538.3588

Fax: 212.760.2327

FalconMQ is distinguished by its clean integration with Microsoft Message Queue and the associated middleware technologies. Level8 Systems partnered with Microsoft to provide non-Windows implementations of MSMQ for enterprise application developers, extending MSMQ's reach to Solaris, HP UX, AIX, SCO-UNIX, Linux, VMS, OS, and MVS.

For developers familiar with the MSMQ programming model, this coverage constitutes a portability layer for code written to the MSMQ API. Messaging options allow developers to force messages to be delivered one at a time, in the same order in which they were sent, and to journal send and receive message queues. Both individual messages and message queues may be prioritized, and applications may preferentially access enqueued messages based on set priorities.

FalconMQ also provides a layer of protocol independence, offering the application a seamless view of the heterogeneous distributed environment. The product provides support for Windows NT Server clustering as a means of configuring automatic failover in high-availability applications.

Testing and deployment tools include fully interactive access to the MSMQ API, supporting real-world testing of message conversations, which would be difficult or impossible to observe using simulation techniques. Because FalconMQ relies on and extends many features and services that are integral to NT and the upcoming Windows 2000, it is substantially less expensive than other competitors in the middleware niche.

Smart Sockets

TALARIAN CORP.

333 Distel Circle

Los Altos, CA 94022-1404

Tel: 800.883.8050

Fax: 650.965.9077

Talarian's Smart Sockets is a stunning product, impressive as much for its highly polished development tool set as the sheer power it demonstrates in deployed applications. Smart Sockets delivers publish/subscribe, peer-to-peer, and request/reply messaging models, providing flexibility and virtually unlimited scalability. Designed to deliver maximum out-of-the-box functionality, Smart Sockets APIs cleanly integrate with C++, Java, and ActiveX objects. Instances of a Smart Sockets client may be replicated as concurrent processes on a single machine or as multiple processes on LAN, WAN, or Internet topologies. No local daemons or agents are required on machines that host Smart Sockets clients, streamlining deployment and administration of applications.

Another notable strength of Smart Sockets technology is the degree to which application behavior may be tuned after deployment. Most Smart Sockets system parameters are exposed and may be adjusted without interrupting or recompiling applications. For example, queue size, time-outs, buffer size, and message-logging destinations may all be modified on the fly. Talarian's solution provides a selection of guaranteed message delivery options, including GMD for one-to-many message dispatches.

Manipulation and hierarchical definition of the namespace allow dynamic registration and routing through the messaging domain.

Perhaps one of the product's greatest areas of achievement is the depth of its hot failover capability. It intelligently avoids operations that may block indefinitely, periodically checks for failure status, and if server processes fail, either restarts them or transparently replaces them with surviving processes.

PIPES Platform

PEERLOGIC, INC.

555 DeHaro Street

San Francisco, CA 94107-9586

Tel: 415.626.4545

Fax: 415.626.4710

PIPES Platform is broadly respected for its unique symmetrical distributed architecture, which protects distributed applications from single-point failures. Often described as being "self-healing," PIPES Platform builds and dynamically maintains a logical view of application resources, allowing it to compensate for the transport layer errors it detects.

Application components register with the PIPES dynamic name service to make themselves available to converse with other components, independent of physical location or host platform type. Because components use the dynamic name service to advertise their availability, applications searching for components do not need to specify or even know names of resources in order to locate them. PIPES Platform maintains information about component availability and location, automatically adapting to changes when components are added, modified, or removed.

PIPES consists of a communications engine, the PIPES kernel (which runs on each machine in the network), a suite of transport drivers that communicate with the local network transports, and the PIPES application programming interface. Each PIPES kernel is a peer and has equivalent functionality across all PIPES supported hosts. The PIPES kernels unify at runtime, creating the PIPES Platform logical network. Applications interact with the PIPES Platform system through the PIPES API (PAPI), which currently supports development in C, COBOL (CICS/MVS), and Visual Basic.

MQSeries

IBM

Dept. YES98

P. O. Box 2690

Atlanta, GA 30301

Tel: 888.411.1WEB code YEC98

Fax: 800.2IBM.FAX code YEF98

At nearly two-thirds overall market share in the financial services industry, MQSeries is something of a standard in high-reliability messaging middleware. Available on more than 35 platforms, MQSeries servers are sleek and modular, making them smoothly interoperable with other IBM server components and distributed processing environments. MQSeries servers were designed with scalability in mind, yielding applications known for ease of integration with legacy systems, flexible transaction processing architecture, and reliability.

MQSeries' flexible communications architecture serves as the infrastructure for mobile communications applications, transaction-oriented systems, and multitier network applications. Because MQSeries relies on message queuing, it is highly effective in mobile applications where servers frequently connect and disconnect from the network.

Fundamentally, MQSeries is a collection of recoverable objects including queues, queue managers, process definitions, and channels, all existing under the oversight and management of the MQSeries system. MQSeries' reliability and recoverability technology guarantee single delivery of messages, with commit rollbacks of processing if the message queue manager loses contact with any of the database managers during message processing.

MQSeries' breadth of coverage is matched by its ease of installation and management. It ships with Candle's Command Center Admin Pac for MQSeries, a suite of tools for testing, configuring, and monitoring MQSeries networks and applications.

VCOM

NETSYS TECHNOLOGY GROUP AB

Nÿmndemansgatan 3

S-431 85 Molndal Sweden

Tel: +46.31.720.6000

Fax: +46.31.27.5010

VCOM began life as a set of technologies developed by Volvo, designed to provide end-to-end automation and synchronization for one of the most complex consumer goods manufacturing processes of all time. The VCOM tool set served Volvo so well that it has been adopted by numerous multinational manufacturing companies inside and outside Scandinavia during its relatively short life span as a commercially available middleware tool.

Using a pair of store-and-forward messaging agents—the partner and the expediter—VCOM systems accept messages on a sending system, store them, wake up the receiving application, and transfer the data. Character set and codepage translation between sender and receiver are handled transparently. VCOM messages may optionally be receipted, so that sending applications can monitor the reception status of data transferred in messages.

Both synchronous and asynchronous messaging are supported. Messages en route over blocked or unavailable network channels are transparently failed over to alternate paths without intervention by application code or administrators. VCOM network monitoring tools allow configuration and management of partner/expediter components via a menu-driven console application. VCOM provides security and privacy by supporting exits and plug-in components for authentication and encryption. Built-in security is accomplished at the application level, where VCOM enforces consistency checking between the security IDs of parties to a message exchange.

Messaging Example: MegaMoney Bancorp

Let's consider how a messaging solution will work in practice. We will present a problem from the banking industry and show how it can be solved using messaging technology. We will return to this example in later chapters, so you can see how the same problem can be solved using distributed objects and transaction monitors.

Our example is a synthesis of cases we have encountered in our consulting engagements at Concept Five Technologies. Not all the details apply to any real banking client, but the overall picture is quite realistic. We have had to simplify the example to keep it to a reasonable length for this book.

The Bank

Our bank, MegaMoney Bancorp, is the result of numerous mergers and acquisitions over the last few years. It offers traditional banking products: checking and savings accounts, certificates of deposit, credit cards, and loans of various kinds. MegaMoney is also moving into nontraditional financial markets, including mutual funds, brokerage services, and insurance. MegaMoney is a major player in consumer financial products, and in our example we'll concentrate on the consumer side of its business.

Each of these products has its own information processing system. These systems run primarily on mainframe computers, but some run on midrange platforms such as Unix or AS/400. For simplicity, we'll talk about only a few of these back-end systems, but keep in mind that there could be many. It is not usual for a large bank to have 30 or 40 individual systems to support its products. MegaMoney has just invested millions of dollars in these legacy systems to make them Year 2000 compliant, so it has a strong incentive to keep them viable for as long as possible.

MegaMoney provides all the typical channels for interacting with its customers: branches, call centers, ATMs, and the Internet. Each of these channels has been integrated with the back-end systems in an *ad hoc* fashion. As a result, MegaMoney's IT architecture is extremely complex and hard to maintain. Each channel supports a different subset of the bank's products.

MegaMoney's Web site was thrown together quickly when it became clear that an Internet presence was a competitive requirement. The Web site provides nonintegrated information from a number of back-end systems. Customers can only query the Web site about account status; it does not support transactions.

EAI Goals

MegaMoney has two goals for EAI.

- It wants the Web site to offer integrated information and to allow transactions: open new accounts, change address and other contact information, and transfer funds between accounts.
- It wants to simplify its IT architecture by providing a single interface for each legacy system that can be accessed by any of the channel systems. Because existing services must be maintained, and each of the channels already has some level of interface to the back-end systems, the transition to the new architecture cannot occur overnight: MegaMoney will have to support both the old and the new interfaces for a period of time.

Figure 4.9 presents a notional view of the modernized IT architecture. Each front-end (or channel) application accesses a common customer application. The customer appli-

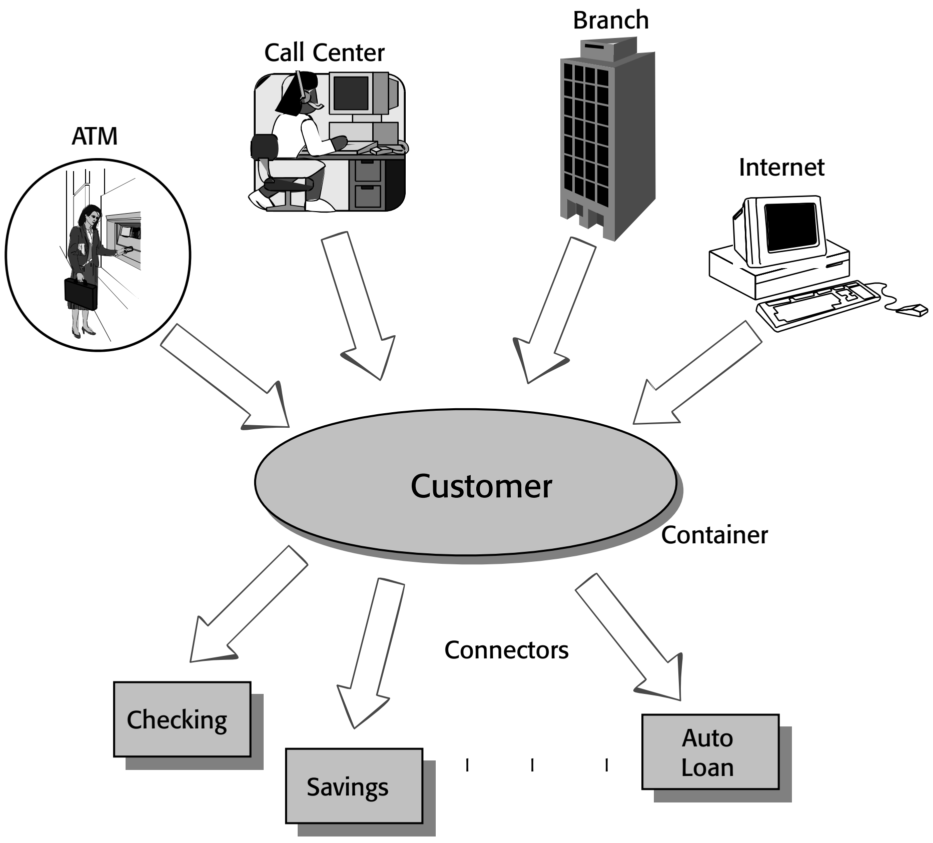


Figure 4.9 The connector-container model enables centralized access to back-end systems.

cation employs the container-connector model. A customer is represented by an entity called a container, which is linked via connectors to the various back-end systems.

The customer application maintains a database of customers, with a small amount of information about each. Each customer has a unique identifier and a list of associated accounts.

MegaMoney's goal is to support a wide variety of services via this architecture, but in the interests of space, we will concentrate on just two: changing the customer's address and transferring funds between accounts.

The change-of-address service will accept a request from a front-end system and, using the list of accounts associated with the customer, transmit the change of address to each back-end system. This may seem like a strange approach—why not just maintain the address information in the customer database and have each of the back-end systems access that address data? In fact, this may be a long-term goal. But each of the legacy systems already has a mechanism for maintaining the customer's address; switching to a centralized database would require modifying each of the back-end systems. MegaMoney's IT architects want to focus on developing new capabilities, so in the short term, they decide it makes more sense not to modify all the back-end systems, but to leverage the capabilities that are already there.

The transfer of funds between accounts is superficially like the change of address. The customer application receives a request from a front-end application and makes an appropriate request of each of the back-end systems to effect the transfer. There is one crucial difference. If the change of address request to one of the back-end systems fails, little real harm will be done, but if one of the back-end requests in the transfer fails and the other succeeds, the books won't balance: Either the bank or the customer will lose some money. Perhaps more importantly, the customer's confidence in the bank will be undermined to a much greater extent than would occur if there were simply an error processing a change-of-address request. So the criteria for the transfer are more stringent: It is imperative that both accounts be properly updated.

The Solution

Let's look at how a solution based on message queuing would work. A queue is associated with the customer application and with each back-end system. When the customer requests a service, say via the Web front end, the front-end application enqueues a message to the customer application. The customer application interprets the message and enqueues a corresponding request to each appropriate back-end system.

The change-of-address application is very straightforward; we simply enqueue a message to each system with the change request. The transfer application is handled the same way, but with one critical difference. For the change of address, it may be acceptable for the messaging system simply to make "best effort" to deliver the message to a back-end system. If, because of a glitch in the messaging system, a given back-end system receives two copies of the change-of-address request, no harm is done. But for the transfer of funds between accounts, it is crucial to guarantee that the transfer message will be processed by each of the involved back-end applications. Moreover, messages cannot be processed more than once. The messaging system must guarantee "exactly once" behavior.

The messaging approach is very flexible. It smoothes out momentary peaks in demand on the customer application; the application's input queue acts as a buffer. Back-end applications do not even have to be running when the original request is processed; messages can simply be enqueued for them until they can be processed. This can be particularly advantageous when the back-end applications depend on overnight batch applications.

The main drawback of the messaging-based solution to our problem arises if information must flow back to the originator of the transaction. In the case of the change of address, this is not a problem. The Web front end simply tells the customer that the change request is being processed, and this is sufficient. For the transfer, however, it seems reasonable to expect that the customer will be given the new balances for each of the accounts involved, and this requires that information flow back to the Web server. This requirement can be handled by establishing queues for the Web server application, but this complicates the problem. Moreover, we need to bear in mind that the Web server may be handling many customer requests concurrently, so some mechanism must be provided for it to sort out which response message applies to a given customer Web session. These complications can certainly be handled in a system based on message queues, but a more elegant approach would couple the initial request with the response in a single interaction. This kind of behavior is provided by other technologies that we will discuss in later chapters, and we will return to this example to look at alternative solutions based on these technologies.

Final Thoughts

Messaging is part of middleware technologies, which are becoming broader in scope and fuzzier in the functionality provided. Middleware is basically framework services that applications and components use as enablers to communication and interaction. Messaging is a key enabling technology. Message queuing is a way to pass data or make an invocation of a remote function from a client to a server, known as point-to-point message queuing. The alternative message queuing model is publisher/subscriber, where a client will subscribe to a particular information set based on specified criteria, and when the server finds a match to the subscription criteria it will publish the information to the subscribing client.

CHAPTER 5

Object Architecture and Solutions

Key Facts

- ➤ Object technology-based components support rapid development of enterprise applications using standard component architecture.
- ➤ Java/EJB supports Internet-enabled application and component development across an enterprise on the Java platform.
- ➤ CORBA provides a useful definition of services and facilities for enterprise application development with services availability still weak.
- ➤ Microsoft DNA 2000 is the next-generation of a proven Microsoft strategy for the enterprise.

Distributed object technologies (DOTs) are implementations that allow objects to be distributed among multiple clients and servers. Many popular distributed object-oriented technologies are offered by international vendors that are based on either proprietary or international standard architectures. Examples of these include the following:

- DCOM/COM+
- CORBA
- Enterprise JavaBeans

This chapter describes specific distributed object-oriented technologies, including OMG CORBA, Microsoft Distributed Network Architecture, and Java/Enterprise JavaBeans. The chapter gives a good, practical grounding on what is available and what benefits they offer.

Distributed Architecture Frameworks

A technology architecture will always affect the *application* architecture within which it is used. When a major technology product, such as an ORB, COM+, or EJB, is used for systems development, its architecture will shape the remainder of the architecture around it. The framework and services that the technology architecture provides constrain the application solutions built using it. That isn't to say that multiple technologies cannot be used together; as they conform to different technology architecture models the systems integrator must find a way to integrate these different architectures. This usually ends up requiring the definition and implementation of an architectural layer that interfaces to the disparate technologies to allow interoperability in the forms of gateways and abstraction layers, hiding the technology architecture differences. Figure 5.1 compares some of the leading architectural approaches offered by major vendors.

In modern technology approaches to architecture, components are the basic building blocks. They are allocated to layers to provide specific types of services and are often distributed. The concept of components is directly supported by popular distributed object technology architectures such as COM+, CORBA 3, and EJB.

The growth of systems that span geographically remote sites has magnified the need to solve the information-sharing problems that this causes. This need has outgrown the solutions offered by two- and three-tier client/server technology with standard protocols. The emergence of distributed computing technologies (see Figure 5.2) is a response to the distributed information-sharing requirements of business worldwide.

The more recent distributed technologies have been object-oriented in nature, and it is these technologies that bring the combined benefits of distributed application development support, standardized enterprise services, and object-oriented technologies (see Figure 5.3).

The most popular and important distributed architectures need to be discussed to appreciate the benefits that they can offer. We will present an overview of Object Management Architecture, along with a specific implementation, IBM's Component Broker.

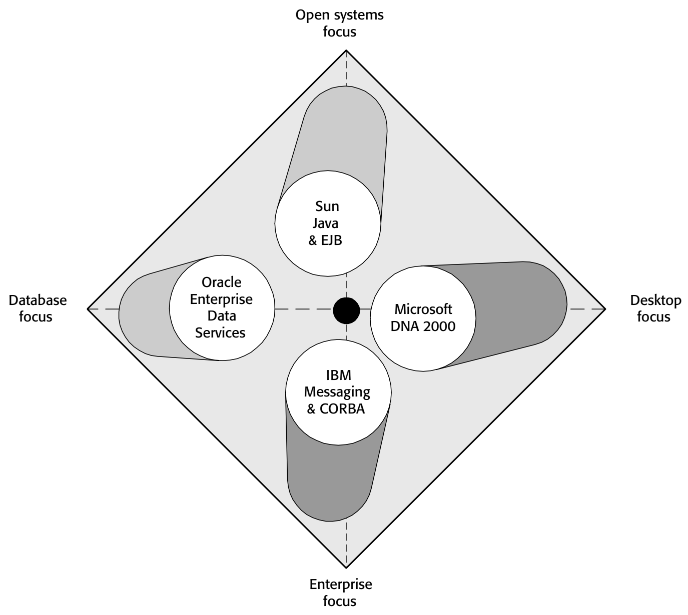


Figure 5.1 Vendor architecture approaches provide different EAI solution options.

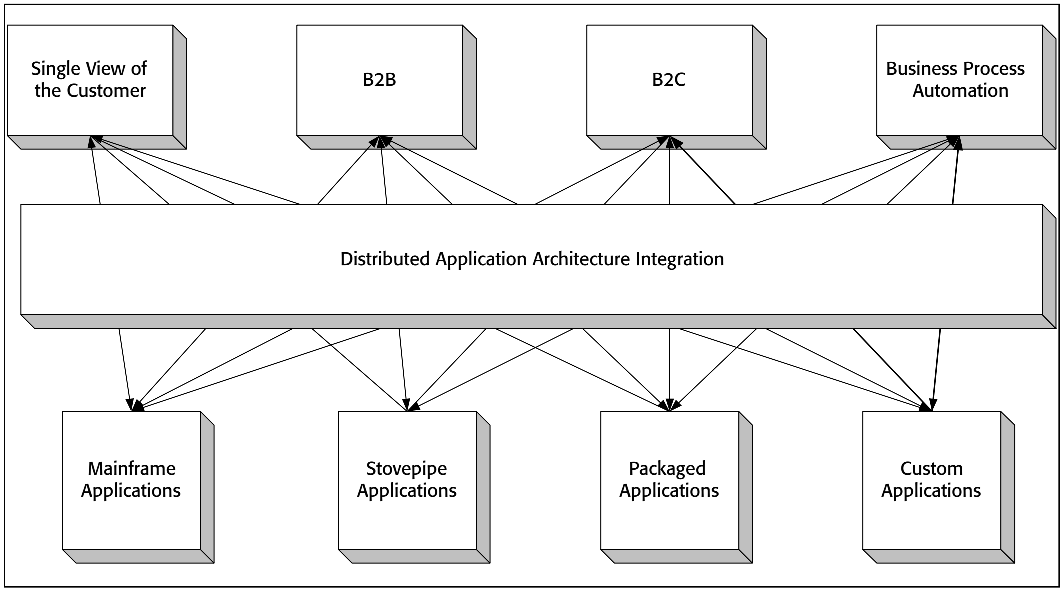

Single View of the Customer
B2B
B2C
Business Process Automation
Distributed Application Architecture Integration
Mainframe Applications
Stovepipe Applications
Packaged Applications
Custom Applications

Figure 5.2 Distributed computing technologies allow for information sharing.

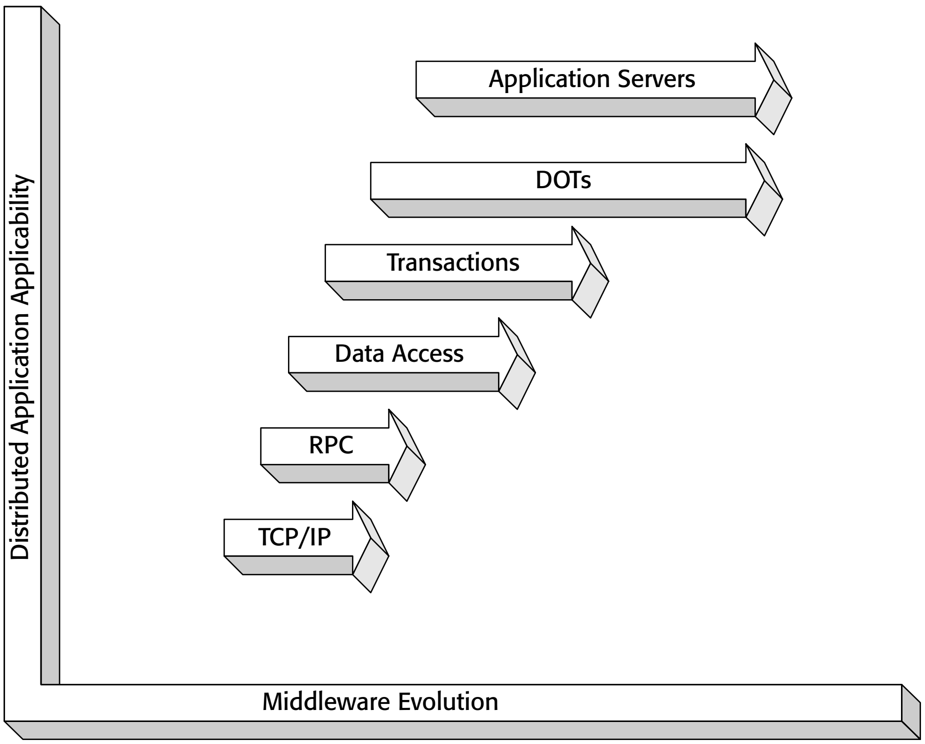


Figure 5.3 The distributed computing middleware evolution is the key to enabling EAI.

Both Microsoft Windows DNA 2000 and Sun Microsystems Enterprise JavaBeans are discussed as well to examine alternative technology approaches to distributed architectures.

Object Management Architecture

The *Object Management Architecture* (OMA) is the framework based on the concept of using an object request broker for object interoperability and distribution (see Figure 5.4). The OMA supports several different architectural approaches. The core subset of the approaches is defined by those directly supported by the Common Object Request Broker Architecture (CORBA). CORBA specifies the interfaces that are used to access CORBA-compliant software.

OMA

The Object Management Architecture is the specification for CORBA software in the form of services and facilities for business domains.

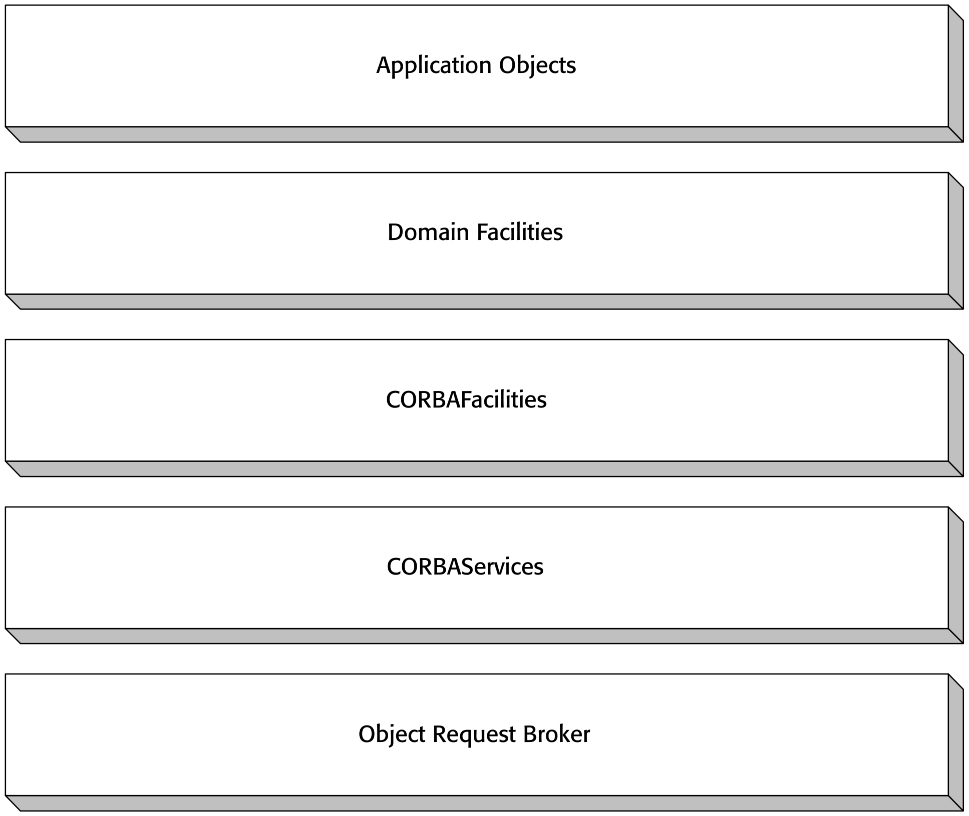


Figure 5.4 The Object Management Architecture specifies a CORBA solution to distributed software development and integration.

The following sections cover the specific aspects of the OMA in terms of the basic architectural concepts and the manner in which the architecture is implemented.

CORBA

The *Common Object Request Broker Architecture* (CORBA) is now in version 3 and is an open distributed computing infrastructure standardized by the Object Management Group (OMG). CORBA provides standard object-oriented interfaces but allows the implementation of the various types of services to be of any nature. Object Request Brokers are the central architectural concept that allows CORBA to be implemented.

The *Object Request Broker* (ORB) provides a mechanism for transparently communicating client requests to target object implementations. ORBs are intended to provide three basic functions:

CORBA

CORBA (Common Object Request Broker Architecture) is a distributed object technology that is platform independent and enables remote object creation and remote object method invocation. It is based on the use of an object request broker with published CORBA services.

OMG

The OMG (Object Management Group) is the focal organization for the definition and adherence to the OMA and CORBA. (www.omg.org.)

- Operating system transparency, so it should not matter what the mix of operating systems is because the services are provided in a consistent manner across all supported platforms
- Remote object lifecycle services, where data and functionality in the form of objects can be created, copied, moved between, and destroyed at distributed locations
- Remote object method calls, where distributed functionality can be invoked as if it were a locally resident function call

ORB

An object request broker (ORB) is a distributable component that can communicate with other ORBS to provide distributed object lifecycle services across multiple platforms.

The ORB simplifies distributed programming by decoupling the client from details of the method invocations. This makes client requests appear to be local procedure calls. In other words, thanks to some exceedingly clever machinery, it makes distributed programming resemble ordinary programming. When a client invokes an operation, the ORB is responsible for finding the object implementation, transparently activating if necessary, delivering the request to the object, and returning any response to the caller [OMG 1999]. Figure 5.5 shows the CORBA ORB architecture. These ORB mechanisms allow objects to be created and invoked remotely on another networked computer while providing operating system and network transparency.

In CORBA terms a *client* is a software component that can run on any computing node of a distributed computing network; either a physical end-user client computer or a physical server computer. A CORBA client is a logical definition of software that acts in

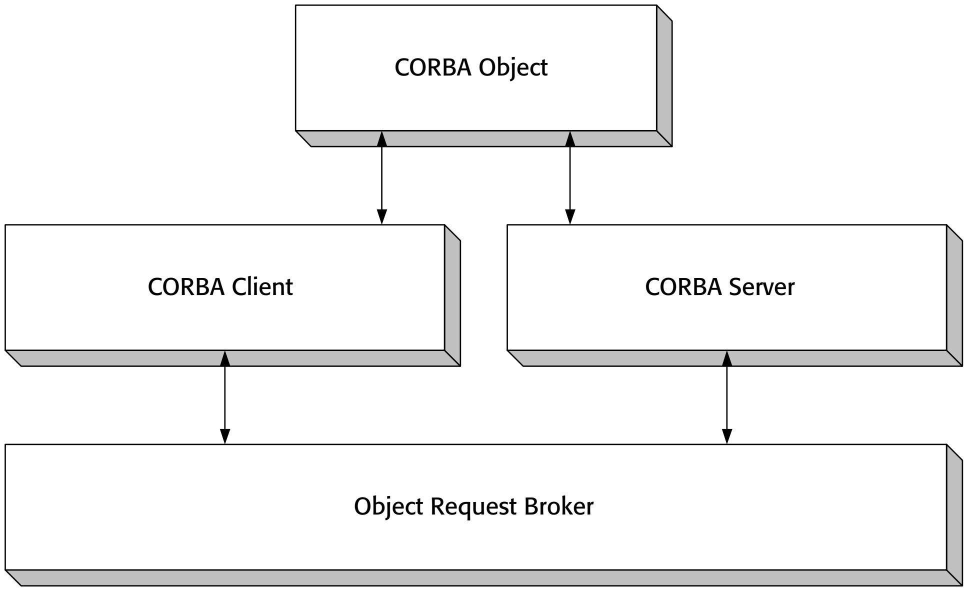


Figure 5.5 The CORBA ORB architecture enables distributed component development.

the manner of a physical client calling on *server* services to supply remote access to data and functions. Also in CORBA a *server* is also a logical, not physical, definition because it also can run on any computing node of a distributed computing network.

CORBA Services

CORBA Services provide a layer above the ORB that enables it to be implemented in a convenient and standard manner. These are domain-independent interfaces that can be used by software developers and systems integrators to build distributed object applications. Without these services a developer would need to reinvent them in a proprietary manner. This would be an expensive option that precludes the reuse, interoperability, and ease of use provided by the CORBA standards. Unfortunately, however useful, not many of these services are implemented by each ORB vendor. Additionally there are the expected proprietary features, which, it seems, standards bodies never manage to eradicate due to vendors who need to differentiate themselves to remain competitive.

CORBA Services

CORBA Services offer a set of distributed object functionality that is required to be available for an application developer to build the distributed application on top of, makes CORBA easier to use for distributed application development, and removes the need to program at the communications level of the ORB.

The OMG have standardized many CORBA services as referenced in the book *Inside Corba : Distributed Object Standards and Applications* (Addison-Wesley Object Technology Series) by Thomas J. Mowbray, William A. Ruh, Richard M. Soley.

Information Management Services define the standard services for the management of data. These include the following:

- **Property Service.** A dynamic set of attributes that can be attached to application objects.
- **Relationship Service.** A general-purpose service for establishing linkages and relationships among objects.
- **Query Service.** A general-purpose set of software interfaces for query objects.
- **Externalization Service.** The marshalling and streaming out of CORBA object attributes into a flat structure.
- **Persistence Object Service.** Gives the ability to make a CORBA object persistent.
- **Collection Service.** Provides interface definitions for common groupings of objects.

Task Management Services define the standard services to support distributed application infrastructure. These include the following:

- **Events Service.** Defines generic interfaces for passing event information among multiple sources and multiple event consumers.
- **Concurrency Service.** A general-purpose service for ensuring atomic access to distributed objects.
- **Transaction Service**. A general-purpose set of interfaces that can be used to encapsulate a variety of existing technologies and provide standard interfaces across all implementations of transaction monitors.

System Management Services define the standard services required for the development of distributed systems. These include the following:

- **Naming Service.** A generic directory service analogous to a telephone book white pages.
- **Lifecycle Service.** Includes the specification of a factory to create, copy, and move CORBA objects.
- **Licensing Service.** General-purpose interfaces for the flexible protection of intellectual property, particularly relevant to electronic commerce.
- **Trader Service.** A generic directory service analogous to a telephone book yellow pages where available services can be advertised and CORBA objects can look them up.

Infrastructure Services define the standard services for application and system extensions. These include the following:

- **Security Service.** Interfaces for authentication, authorization, audit, nonrepudiation services.
- **Time Service.** Secures time.

- **Messaging Service.** Interfaces to support fine-grain asynchronous processing all the way up to large-scale asynchronous messaging.

These services provide much of the required functionality for a developer to implement a distributed application on top of CORBA.

CORBAFacilities

CORBAFacilities are intended to support development of end-user applications by offering application-level services (facilities) that can readily be used by an application developer.

CORBAFacilities

CORBAFacilities are application-level services that are intended to be used as distributable components of a distributed application.

CORBAFacilities include the following:

- Distributed Document Component Facility
- Internet Facility
- Print Facility
- Workflow Management Facility
- Calendar Facility

These facilities are examples of application-level services that are common to multiple business domains.

CORBA Domains

CORBA Domains are business domain-specific standard groups within the OMG that define standard domain services; such as the Financial Domain Task Force that has led the standardization of the Currency Specification for use within the financial business domain.

CORBA Domains

CORBA domains are line-of-business oriented and issue specifications for use within their specific business domain.

The domain standard groups include the following:

- Business Objects
- Electronic Commerce
- Finance
- Life Sciences
- Manufacturing
- Medical
- Telecommunications

The CORBA domains cover many of the major business domains that use information technology in an increasing manner.

CORBA 3

CORBA 3 introduces new OMG specifications that make CORBA more extensive and easier to use. CORBA 3 provides two specifications for Internet integration:

- The firewall specification
- The interoperable name specification

The CORBA 3 *Firewall Specification* defines transport-level firewalls, application-level firewalls, and perhaps most interesting, a bidirectional GIOP connection useful for callbacks and event notifications. In other words, the communications protocol, GIOP, supports the basic requirements of request/reply messaging.

In CORBA, objects frequently need to call back or notify the client that invoked them; for this, the objects act as clients and the client-side module instantiates an object that is called back in a reverse-direction invocation. Because standard CORBA connections carry invocations only one way, a callback typically requires a second TCP connection to be established for this traffic heading in the other direction, which is prohibited in virtually every firewall in existence. Under the new specification, an IIOP connection is allowed to carry invocations in the reverse direction under certain restrictive conditions that don't compromise the security at either end of the connection. The Firewall Specification comprises two documents: www.omg.org/cgi-bin/doc?orbos/98-05-04 and an erratum, www.omg.org/cgi-bin/doc?orbos/98-07-04.

The second specification is the *Interoperable Name Service Specification*. The CORBA object reference is a critical innovation to distributed object identification and access. Previously, the only way to access an object and invoke one of its methods was to need to know its reference, which acts as a unique name. Even if you knew its location and that it was up and running, accessing a CORBA object was not possible. The interoperable name service allows a CORBA object to be identified by a URL (Uniform Resource Locator) reference, which is the standard used for Web site identification.

The URL for the Interoperable Naming Service Specification is www.omg.org/cgi-bin/doc?orbos/98-10-11.

CORBA 3 defines two specifications that affect the manner in which CORBA can be implemented:

- Asynchronous messaging and quality of service control
- Minimum, Fault-Tolerant and Real-Time CORBA

The *Asynchronous Messaging Specification* defines asynchronous and time-independent invocation modes for CORBA, allowing both static and dynamic. Asynchronous invocations results may be retrieved by either polling or callback, with the choice made by the form used by the client in the original invocation. Policies allow control of Quality of Service of invocations. Clients and objects may control ordering (by time, priority, or deadline); set priority, deadlines, and time-to-live; set a start time and end time for time-sensitive invocations; and control routing policy and network routing hop count. The URL for the Messaging Specification is www.omg.org/cgi-bin/doc?orbos/98-05-05.

Minimum, Fault-Tolerant, and Real-Time CORBA is primarily intended for embedded systems; that is, software that is written onto computer chips or ROMs and is permanent, with predictable external interactions and behavior. They have no need for the dynamic aspects of CORBA, such as the Dynamic Invocation Interface or the Interface Repository that supports it, which are therefore not included in Minimum CORBA. The URL for the Minimum CORBA specification is www.omg.org/cgi-bin/ doc?orbos/98-08-04.

Fault tolerance for CORBA is addressed by an RFP, also in process, for a standard based on entity redundancy and fault management control. The URL for all information on this RFP is www.omg.org/techprocess/meetings/schedule/Fault_Tolerance_RFP.html.

fault tolerance

Fault tolerance is the ability for errors to occur without crashing the system, ensuring a stable, known system state.

Real-time CORBA standardizes resource control, such as threads, protocols, connections, and so on, using priority models to achieve predictable behavior for both hard and statistical real-time environments. Dynamic scheduling, not a part of the current specification, is being added via a separate RFP. The URL for the Real-Time CORBA specification is www.omg.org/cgi-bin/doc?orbos/99-02-12; an erratum is www.omg.org/cgi-bin/doc?orbos/99-03-29.

CORBAcomponents Package

CORBAcomponents represents three major component-based innovations:

- A container environment that packages transactionality, security, and persistence and provides interface and event resolution

- Integration with Enterprise JavaBeans (EJB)
- A software distribution format that enables a CORBAcomponent software marketplace

The CORBAcomponents container environment is persistent, transactional, and secure (see Figure 5.6). For the programmer, these functions are prepackaged and provided at a higher level of abstraction than the CORBAServices provide. This leverages the skills of business programmers who are not necessarily skilled at building transactional or secure applications, who can now use their talents to produce business applications that acquire these necessary attributes automatically.

Containers keep track of event types emitted and consumed by components, and they provide event channels to carry events. The containers also keep track of interfaces provided and required by the components they contain, and they connect one to another where they fit. CORBAcomponents support multiple interfaces, and the architecture supports navigation among them.

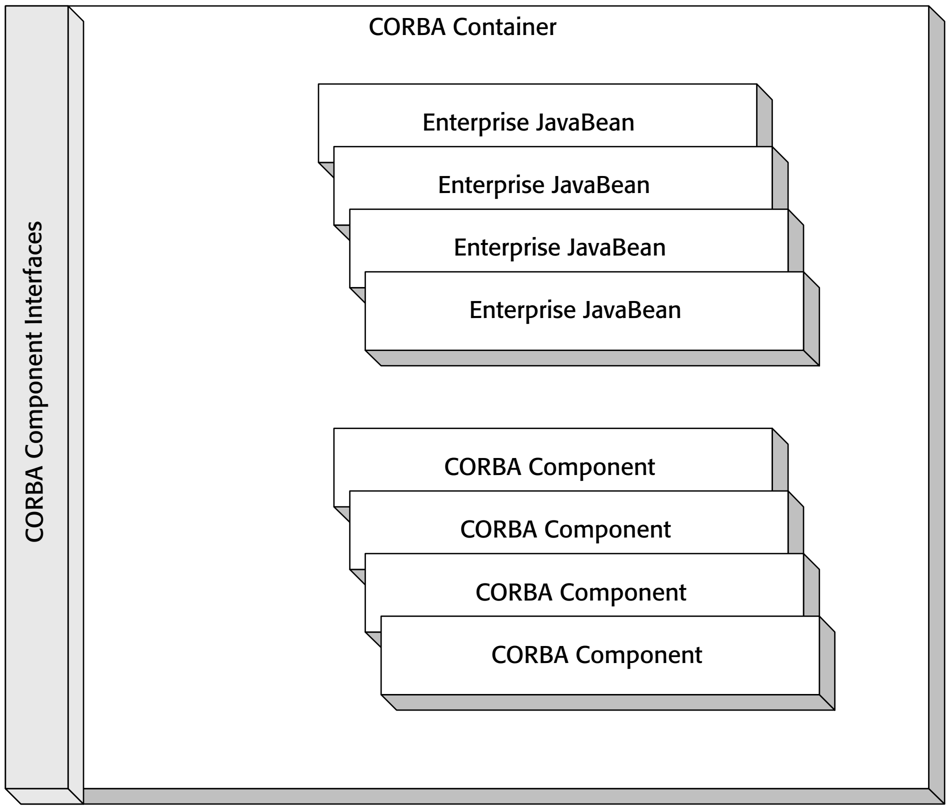


Figure 5.6 CORBA components enable EJB integration.

The CORBAcomponent specification allows EJBs to act as CORBAcomponents and to be installed in a CORBAcomponents container. Unlike EJBs, CORBAcomponents can be written in multiple programming languages and support multiple interfaces.

Drawbacks to CORBA

IBM (www.ibm.com) strive to lead in the creation, development and manufacture of the industry's most advanced information technologies, including computer systems, software, networking systems, storage devices and microelectronics.

Now that the OMG has resolved the lack of asynchronous support in CORBA with the Asynchronous Messaging Specification, the only serious drawback to using CORBA is that the ORB vendors still implement very few of the basic services that are essential to making an ORB truly useful. This significantly reduces the advantages of the services, facilities, and domain specifications because a serious enterprise user will need to implement the missing services for whatever ORB is selected. Another drawback is that many CORBA implementations are not as stable as enterprises require. But like all technologies a critical level of use will help drive improvements and ensure an increasing product maturity.

IBM Component Broker

IBM (www.ibm.com) strive to lead in the creation, development and manufacture of the industry's most advanced information technologies, including computer systems, software, networking systems, storage devices and microelectronics.

IBM Component Broker is an Object Request Broker, that is, a CORBA implementation. In other words, it is an ORB with a set of packaged CORBA services and facilities.

IBM Component Broker

Component Broker is an enterprise solution for distributed object computing that includes an operational environment and tool set and is available as a part of IBM Enterprise Edition of WebSphere Application Server.

It provides support for the following (see Figure 5.7):

Development. Designing, implementing, and debugging objects that represent a business model. Component Broker Toolkit provides leading-edge tools for developing object models and comprehensive frameworks for implementing and testing them using a number of object-programming languages.

Execution. Providing a secure, robust, and scalable distributed execution environment for objects that implement a business model. Component Broker Connector provides execution libraries for different systems platforms (e.g., MS Windows NT, IBM AIX, OS/390) and a variety of networking transport protocols (e.g., TCP/IP, IBM SNA, and LAN) to establish a mission-critical infrastructure that can be integrated tightly with existing technologies in enterprise scenarios.

Management. Providing tools to deliver, install, validate, configure, control, and troubleshoot large-scale distributed object systems. Component Broker Connector provides facilities for immediate use by operational end users and programming frameworks for customizing an installation to meet local needs and enterprise guidelines.

IBM describes Component Broker as follows:

- Component Broker applications are collections of client views of a set of business objects managed by the Component Broker runtime environment called CBConnector.
- Business object interfaces are defined in OMG Interface Definition Language (IDL). Using the CBToolkit Object Builder tools, both client views and implementation skeletons are automatically generated in Java, C++, or Smalltalk according to specification. Client views may also be generated with Microsoft ActiveX interfaces for access from Visual Basic programs or ActiveX container applications such as word processors or spreadsheets. Distribution of and communications among client views and Component Broker business objects are achieved using an IBM CORBA-compliant Object Request Broker.
- Objects in a business model created using these facilities typically fall into one of several categories defined within the Component Broker Programming Model. Briefly, they are defined according to their intent within the model:
- Application Objects represent the state that encapsulates process or flows of interaction (e.g., conversations and workflow) between clients and business objects in the server.
- Business Objects represent basic external objects in the enterprise model, which may be used directly or composed into higher-level objects.
- Composed Business Objects represent higher-level external objects that are dynamically composed from lower-level business objects in the same or different CBConnector runtime environments.
- State Objects represent the persistent state on which basic business objects are built at execution time.

IBM Component Broker supports the following services (see Figure 5.7):

Lifecycle services. Provide for basic object creation, copy, move, and destruction within the managed domains of CBConnector runtimes.

Identity services. Provide the mechanism for uniquely identifying object instances from their object references within the managed domains of CBConnector runtime environments.

Externalization services. Provide the mechanism for streaming out the contents of business objects to an external medium and for streaming them back again.

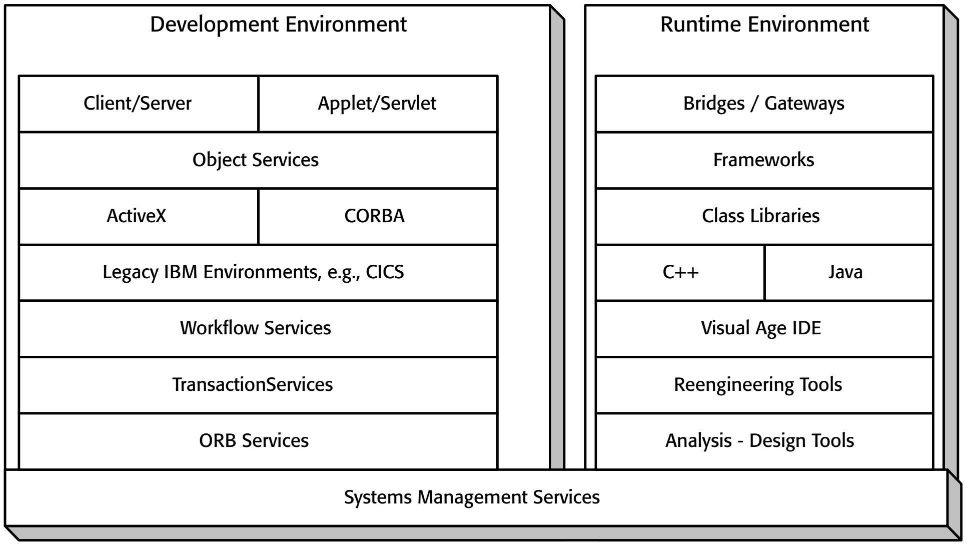


Figure 5.7 The IBM Component Broker architecture is a vendor implementation of CORBA.

Naming services. Provide a mechanism for locating named business objects or collections of them within a network. The Component Broker naming service is based on DCE naming standards.

Security services. Provide authentication checks for clients wishing to access business objects and authorization checks at the class, instance, or method level. The CBConnector implementation is based on DCE security standards.

Event services. Provide business objects with an asynchronous mechanism for communicating with other business objects and for automatic notification of changes from one object to another.

Persistence services. Provided through State Objects in Application Adaptor domains. State Objects provide the mapping between business object state and external persistent stores.

Concurrency services. Provide the locking mechanism that enables sharing of business object state between active clients. This service is used in conjunction with the persistence and transaction services.

Transaction services. Provide the mechanism for identifying sets of changes to business objects as atomic units of work that may be committed or rolled back as required.

IBM Component Broker can support a number of different types of clients (see Figure 5.8), including the concept of "thin" clients that can be realized by downloading Java applets and business object proxy interfaces to Web browsers on demand from end-user machines.

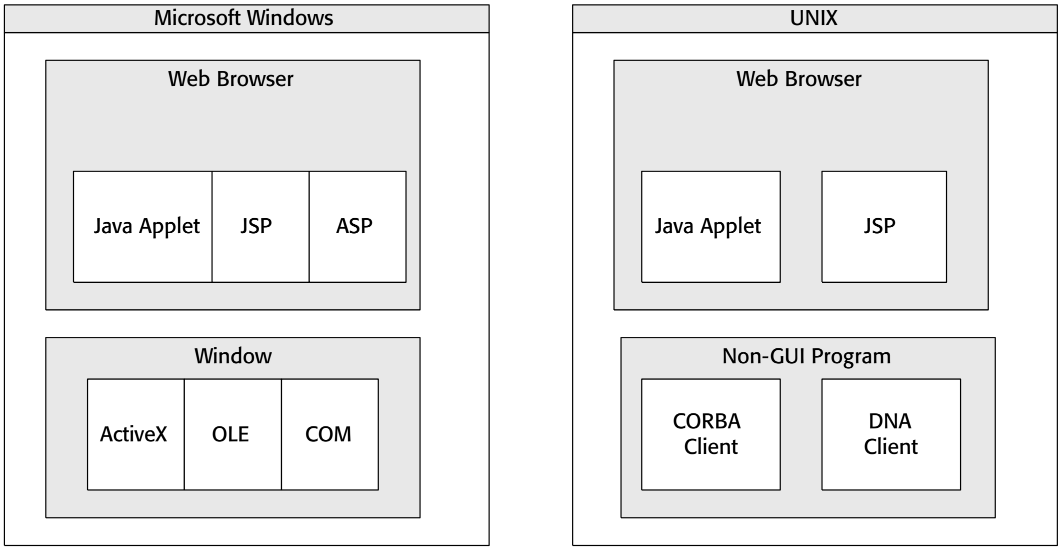


Figure 5.8 IBM Component Broker can support thin client implementations.

thin clients

Thin clients are described as such because the required client resident software is minimal. The best example of this is downloading an applet from the Internet that creates and runs a thin application on the desktop. The thin clients usually use "thick" servers.

IBM Component Broker enables access to server-resident business objects from Microsoft Windows by providing ActiveX interfaces to business objects that are callable from Visual Basic, word processors, spreadsheets, and other ActiveX component containers. This provides practical integration to business users between IBM-based software and Microsoft-based software by bridging the technology product architectures. This enables the integration of enterprise applications (EAI).

Microsoft Windows DNA 2000

Microsoft (www.microsoft.com) products include operating systems for personal computers and networks, server applications for client/server environments, business and consumer productivity applications, interactive media programs, and Internet platform and development tools. Microsoft also offers online services, personal computer books and input devices, and it researches and develops advanced technology software products. Microsoft products, available in more than 30 languages and more than 50 countries, are available for most PCs, including Intel microprocessor-based computers and Apple computers.

DNA is Microsoft's architectural framework for distributed object technology use for enterprise application integration. Microsoft Windows DNA (Distributed iNternet Architecture) is intended to support enterprise Web enablement and enterprise application integration. It primarily relies on the use of five key Microsoft products to achieve this:

Windows 2000 Server. The latest Microsoft Windows release with built-in Web and application services and Internet-standard security.

SNA Server. Gateway and integration platform.

SQL Server 7.0. Scalable relational database.

Site Server. Used to publish, find, and share information over an intranet.

Visual InterDev 6.0. Allows rapid server application development.

DNA 2000 defines that applications are logically separated into three layers (see Figure 5.9):

- Presentation services provided by Visual InterDev 6.0
- Business services provided by Visual Basic using COM+ for distributed components
- Data services provided by Visual Basic and SQL Server 7.0

In addition, Microsoft provides the underlying system and network services using Windows 2000 Server, SNA Server, and Site Server.

Microsoft specifies that these three tiers are logical separations within the application. The physical deployment of an application can span networks of computers.

Microsoft DNA 2000 is based on the COM+ programming model and Windows 2000 server services. The earlier COM model has come into widespread use, and it is an integral part of many Microsoft applications and technologies, including Internet Explorer and the Office suite of applications. The intent of COM+ is to further enable significant component reuse with programming language independence between Microsoft Visual Basic, Microsoft C, Microsoft C++, and future programming languages.

Microsoft DNA 2000 relies on the integrated services (see Figure 5.10) supplied as part of the Microsoft Windows 2000 Server platform to provide support for transaction processing, security, directory services, and systems management, with products such as MTS (Microsoft Transaction Service), MSMQ (Microsoft Message Queue), and SMS (Systems Management Services).

The downside to Microsoft DNA 2000 is that it is primarily a Microsoft Windows-based architecture. Microsoft, though, has been steadily increasing its level of integration with Unix. Also distributed object technology vendors have started delivering integration solutions for Microsoft and CORBA-based vendor products. Additionally Microsoft has released COM for Solaris and intends to deliver on other Unix platforms, such as: DEC Unix, HP/UX, and SNI Reliant Unix.

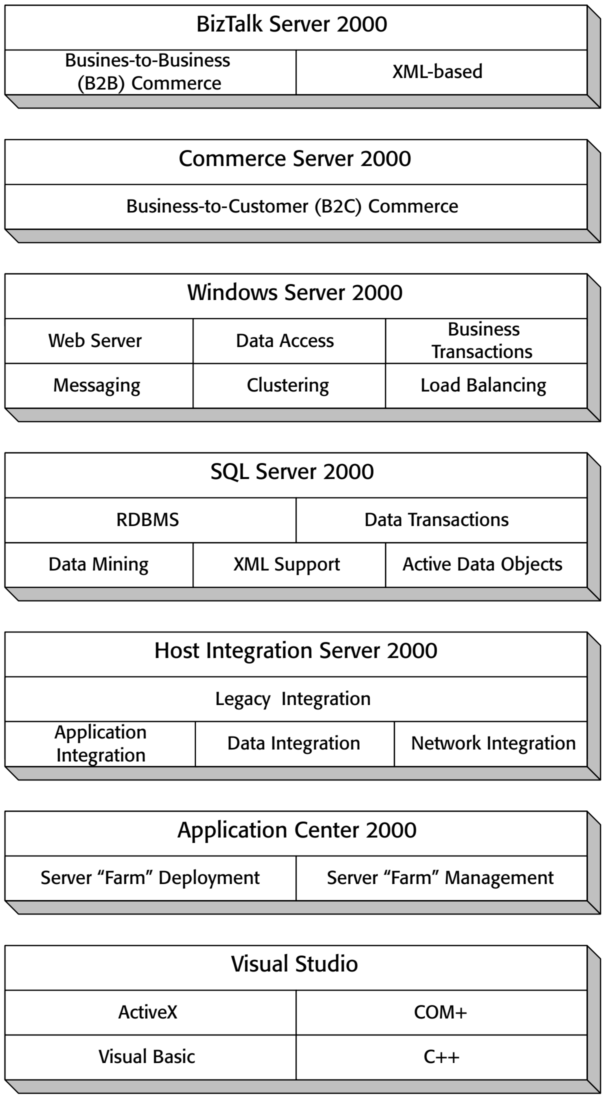


Figure 5.9 Microsoft DNA 2000 defines a distributed application and integration strategy.

Microsoft is already producing the next wave of DNA building blocks, built on Windows 2000 Server:

SQL Server 2000. Adds XML and Internet standards support, secure data access from a browser, through firewalls and online analytical processing.

Exchange 2000. A messaging platform that provides e-mail, scheduling, online forms, and the tools for custom collaboration and messaging-service applications.

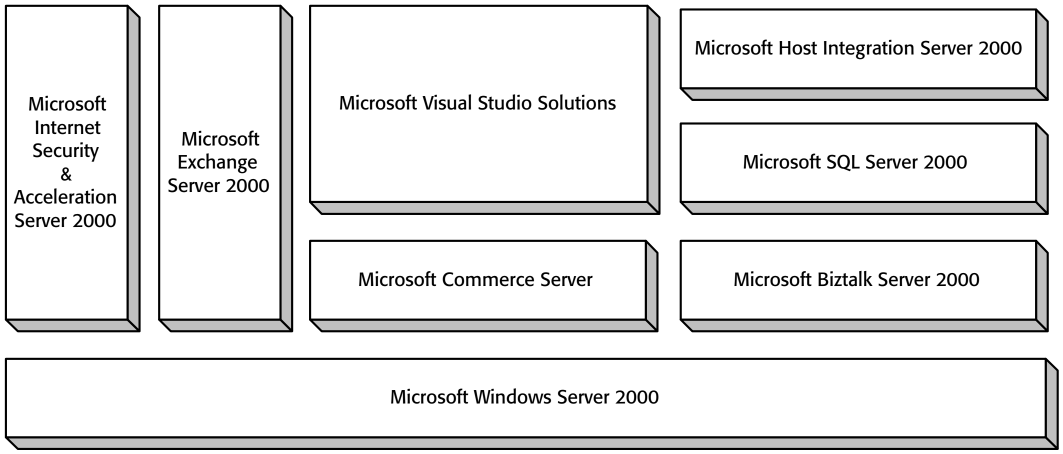

Microsoft Internet Security & Acceleration Server 2000
Microsoft Exchange Server 2000
Microsoft Visual Studio Solutions
Microsoft Host Integration Server 2000
Microsoft SQL Server 2000
Microsoft Commerce Server
Microsoft Biztalk Server 2000
Microsoft Windows Server 2000

Figure 5.10 The Microsoft Distributed iNternet 2000 application architecture provides a multi-tier distributed application framework.

Commerce Server 2000. Internet commerce server for online business.

Host Integration Server 2000. The follow-up to SNA Server, which provides application, data, and network interoperability.

Biztalk Server 2000. Infrastructure and tools to enable e-commerce business communities using a rules-based business document routing, transformation, and tracking infrastructure.

Application Center 2000. For scalable, distributed applications supporting large numbers of clients by servicing their requests concurrently using a group of servers.

Services for Unix 2.0. Key DNA 2000 services to run on the Unix platform.

Sun Microsystems Enterprise JavaBeans Architecture

Sun Microsystems is a global hardware, software and enterprise solutions provider (www.sun.com). In 1995 Sun unveiled the Java platform as a way to create applications that would run on any computer, regardless of the underlying operating system, and provide a cross-platform application development and deployment environment. Java enterprise technology makes it simpler for software developers, service providers, and device manufacturers to rapidly develop robust solutions. The current three editions of the Java 2 platform include: the Java 2 Platform, Micro Edition (J2ME) for small devices and smartcards; the Java 2 Platform, Standard Edition (J2SE) for desktops; and the Java 2 Platforms, Enterprise Edition (J2EE) for creating server-based applications and services.

Enterprise JavaBeans (EJB) technology defines a model for the development and deployment of reusable Java server components (see Figures 5.11 and 5.12). Components are predeveloped pieces of application code that can be assembled into working application systems [Patricia Seybold Group 1999].*

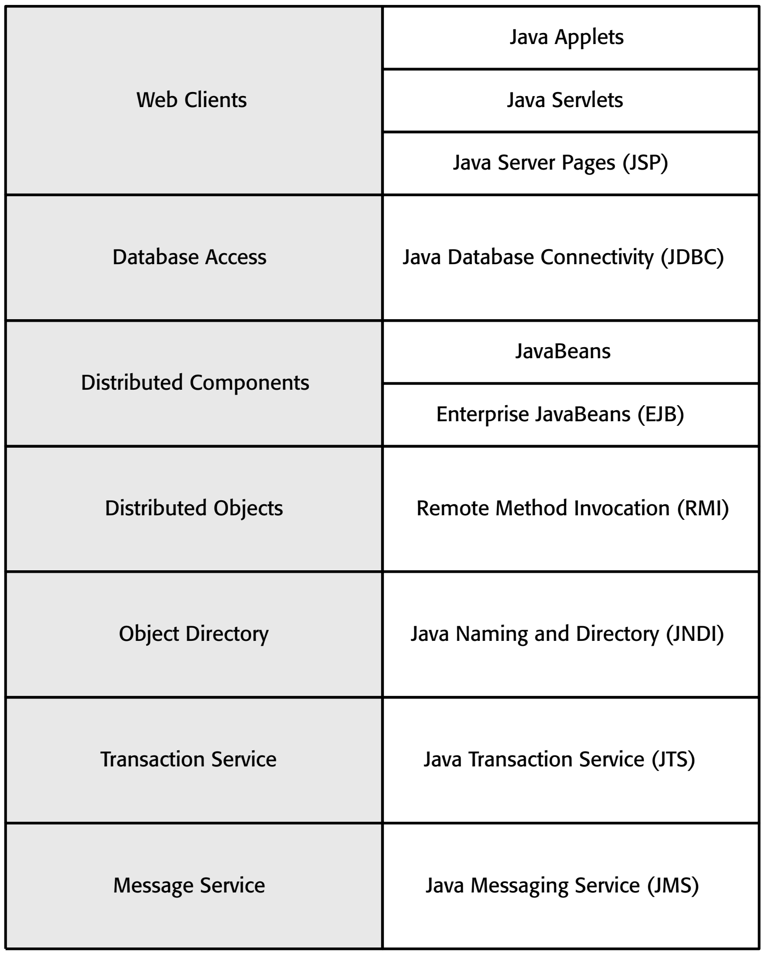
EJB technology is intended to support multitier, distributed object application development. This is particularly focused on thin clients, where the majority of the application functionality resides on the server side to enable enterprise, large-scale, distributed application support. A Java application server provides server-resident Java application components that combine online transaction processing (OLTP) and distributed object technology (DOT) services for the development of Internet-enabled application systems [Patricia Seybold Group 1999].

EJB provides a set of enterprise component interfaces (APIs) for standardized components on the Java platform:

- The Enterprise JavaBeans API defines a server component model that provides portability across application servers and implements automatic services on behalf of the application components.
- The Java Naming and Directory Interface API provides access to naming and directory services,such as DNS, NDS, NIS+, LDAP, and COS Naming.
- The Remote Method Invocation creates remote interfaces for distributed computing on the Java platform.
- The Java Interface Definition Language API creates remote interfaces to support CORBA communication in the Java platform. Java IDL includes an IDL compiler and a lightweight, replaceable ORB that supports IIOP.
- The Java Servlets and Java Server Pages APIs support dynamic HTML generation and session management for browser-based clients.
- The Java Messaging Service API supports asynchronous communications through various messaging systems, such as reliable queuing and publish-and-subscribe services.
- The Java Transaction API provides a transaction demarcation API.
- The Java Transaction Service API defines a distributed transaction management service based on the CORBA Object Transaction Service.
- The JDBC Database Access API provides uniform access to relational databases, such as DB2, Informix, Oracle, SQL Server, and Sybase.

Sun Microsystems has released the specification for EJB 2.0 and added three enterprise-ready services:

*The Patricia Seybold Group (www.psgroup.com) specializes in helping companies with their B2B strategies, e-business best practices and technology architecture decisions. Founded in 1978 and based in Boston, Massachusetts, the firm offers customized consulting services, an online strategic research service, executive workshops, and in-depth research reports.



Web Clients	Java Applets
	Java Servlets
	Java Server Pages (JSP)
Database Access	Java Database Connectivity (JDBC)
Distributed Components	JavaBeans
	Enterprise JavaBeans (EJB)
Distributed Objects	Remote Method Invocation (RMI)
Object Directory	Java Naming and Directory (JNDI)
Transaction Service	Java Transaction Service (JTS)
Message Service	Java Messaging Service (JMS)

Figure 5.11 Sun Enterprise Java Beans architecture provides a distributed application framework.

- Integration with Java Message Service
- Container-Managed Persistence
- Inter-Server Interoperability

Integration with the Java Message Service (JMS) allows enterprise beans to participate in typical asynchronous messaging operations. This enables enterprise beans to be used for asynchronous transactions, which eliminates the need to perform all transactions in strict sequence because all messaging is performed synchronously only.

Container-Managed Persistence (CMP) makes application development simpler and more rapid by allowing developers to implement portable applications that are database

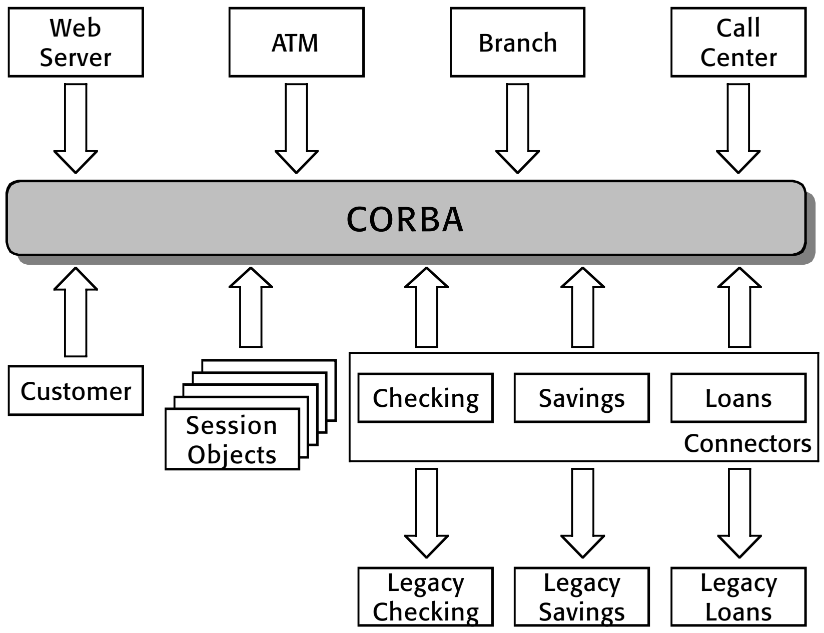


Figure 5.12 Sun Enterprise Java services provide functions to support the implementation of distributed applications.

independent, not requiring database-specific access code to be embedded in the application. CMP includes a standard query language for directly querying data in enterprise beans.

Inter-Server Interoperability is intended to support B2B (business-to-business) integration across heterogeneous environments by extending the cross-server application portability to interserver application interoperability. This eliminates the need for enterprise beans to have embedded knowledge of the application server on which they are running.

Applying Object Architecture to MegaMoney Bancorp

Let's return to the example that was introduced at the end of the last chapter and see how distributed-object technologies can be applied to the problem of MegaMoney Bancorp. Recall that MegaMoney was attempting to improve its Web site, which had been hastily thrown together, to offer its customers integrated access to all their accounts. The bank also wanted to simplify its IT architecture by developing a single customer application, which could be linked to various back-end systems using the connector-container model.

In Chapter 4, "Messaging Architecture and Solutions," we saw how an architecture based on message queues could be applied to MegaMoney's problem. We noted that, although a messaging solution was certainly workable, the messaging pattern did not fit well into the synchronous-interaction paradigm of a Web site.

It will be no surprise to learn that distributed-object technology deals very well with these problems. Figure 5.12 illustrates the solution. As we saw previously, a Web-based front-end application is linked to a customer application, which in turn is linked to back-end legacy applications. These links are now implemented using distributed-object technology. For this example, we'll assume CORBA, but the other technologies discussed in this chapter would work in much the same way.

One difference from the message-based pattern is worth noting. In the messaging architecture, a monolithic customer application supported an input queue of requests. The customer application had to be able to switch contexts to support numerous front-end clients. These clients could access the customer application through the Web, from a call center, or from one of MegaMoney's branches. The need to juggle multiple customer accounts makes programming the customer application more difficult than it would be if the logic had to deal with just one account.

The object architecture handles this problem by creating a separate object for each account. We call this object a *session object* because it is created at the start of a session dealing with an individual customer's account and is returned to a pool of available session objects when the customer's session ends, ready to be used for another customer's session. Recycling a session object through a pool is more efficient than creating a new object at the beginning of each customer session and destroying that object at the end of the session. Because the session object deals with only one customer's information for the duration of the session, the programming is somewhat simpler than for the monolithic customer application that we described in Chapter 4. The effort of supporting multiple concurrent customer sessions has been transferred from MegaMoney's application programmers to the ORB. Letting the middleware handle this kind of issue, and letting application programmers concentrate on business logic, is one of the hallmarks of effective EAI.

Object technology simplifies the programming of MegaMoney's applications in another important way. In the messaging architecture, it was the responsibility of the various applications to decode and interpret the messages flowing through the system. In the distributed-object architecture, the interfaces between the applications are defined in CORBA IDL. The Customer application (or the session object) can support distinct *ChangeAddress* and *TransferFunds* operations. Moreover, the parameters of these operations (the amount of money to be transferred, for example) are also defined in the IDL. MegaMoney's programmers don't need to extract this information from a message, which means they have less code to write and are less likely to make errors—all of which will help MegaMoney to roll out its new applications faster.

Is a distributed-object solution better than a messaging solution? No, not always. One drawback of distributed objects is that the technology is mainly synchronous. (Asynchronous capabilities have recently been defined within the CORBA community, but implementations are not available as of this writing.) Recall that the asynchronous nature of message queues provides a means of smoothing out peaks in demand for service. Moreover, some of the back-end systems with which we wish to integrate may be batch systems, for which synchronous integration is not feasible. One possible way of dealing with this problem is to define an object interface, whose implementation utilizes the messaging system to enqueue an asynchronous message for the back-end application. This approach combines the programming simplicity of the object model with the asynchronous support of the messaging system. Moreover, MegaMoney, like many enterprises, is trying to more toward more real-time processing of information and away from overnight batch processing, in order to make its information more timely. By defining an object interface for its back-end applications now, MegaMoney will be able to transition them to a real-time basis in the future without having to change the interface.

A more serious problem is that CORBA—at least, "plain vanilla" CORBA—is not transactional. Recall that for the service of transferring funds between accounts, it is crucial that the entire operation complete successfully. It is not acceptable to update one account and not the other, nor to update both accounts and not complete the process of notifying the front-end application that the operation has been successfully completed. Ensuring that the entire transaction completed successfully, even if one of the applications crashes or the communications links between applications fail, requires very complex programming. In the next chapter, we will see how transaction-oriented middleware can help us deal with this problem.

Final Thoughts

Each of the architecture approaches discussed are all aimed at integrating systems, applications, and components by providing framework services and integration technologies. They each achieve a similar result but use significantly different programming models to do so:

- Common Object Request Broker Architecture
- COM architecture model
- Java/EJB architecture model

These different distributed architecture models can be integrated but not easily because of specific vendor product differences. For instance, Microsoft Java (J++) is not interchangeable with other vendors' Java programming environments but provides DCOM, COM, and OLE support, which the others do not. Picking a technology-based architecture approach is only the starting point. Choosing vendors for ORBs is where the rubber meets the road; each of the examined architecture approaches has been proven successful but each brings its own built-in limitations.

CHAPTER 6

Transaction Architecture and Solutions

Key Facts

➤ A transaction is a single logical unit of work that is composed of subunits, all of which must complete successfully in order for the transaction to be successful.

➤ Transactions exhibit the ACID properties: atomicity, consistency, isolatability, and durability.

➤ The Open Group's Distributed Transaction Processing standard specifies how distributed transactions should be managed.

➤ CORBA-based OTMs, EJB implementations, and Microsoft's MTS combine the benefits of distributed object technology and distributed transactions.

➤ Transactional integrity may be difficult to achieve when integrating legacy applications. Messaging and soft rollback can mitigate some of the drawbacks.

In the previous two chapters, we have seen how distributed object and messaging systems allow applications to communicate with one another. In order truly to *integrate* applications, however, we need to do more. We need to ensure that the integrity of business processes is preserved, not just when everything works as it is supposed to, but also when failures occur in our applications, in the underlying operating systems and hardware, or in the communications networks that tie them all together. If we're moving money between accounts, we need to make sure that if one account is debited, the other is credited. It's not acceptable to perform only half the transaction.

Of course, we could try to anticipate all the contingencies and carefully code our applications to deal with them—and for simple or less critical applications, we might adopt this strategy. But this would still mean that the programmers are spending their time dealing with technology issues instead of business logic. Moreover, the problem of ensuring transaction integrity is complex, and few programmers have the skills to develop a complete solution.

For more complicated and mission-critical applications, we turn to another category of middleware, transaction processing monitors. *Transaction processing monitors (TPM)* are products that provide the tools needed to ensure the integrity of complex business processes. In this chapter, we look at TPMs and what they do.

transaction processing monitor

Transaction processing monitors ensure the integrity of business processes by providing atomicity, consistency, isolation, and durability of transactions.

The leading transaction monitors were originally designed to provide all the tools necessary to develop high-integrity distributed applications, independently of the distributed object and messaging technologies discussed in the previous chapters. More recently, products called object transaction monitors (OTMs) have begun to emerge that attempt to combine all three approaches. We'll discuss OTMs and show how they attempt to bring together the strengths of distributed objects, messaging, and transaction monitors, while mitigating their weaknesses. Finally, we return to our banking example and see how it is addressed by transactional technologies.

Transactions

What exactly do we mean by a *transaction?* Webster says a transaction is "the act of carrying on or conducting (business, negotiations, activities, etc.) to a conclusion or settlement." We're all familiar with the general notion of transaction in a business. Buying something, say a car, is a transaction. So is making a deposit to a checking account. In IT, we can define a transaction as a single, logical unit of work to support one or more business functions that must be completed in a single action to achieve a business purpose. If not all required business functions can be completed, then the transaction cannot be completed.

When we look at transactions from the business's perspective, they are somewhat more complicated. Suppose we're buying not simply a commodity product, but a specialized device that's custom built to order. Inside the enterprise then, the purchase transaction may involve numerous steps: checking parts inventories (and ordering parts if none are in stock), scheduling factory jobs, planning for transportation, and making entries into the accounting system. Each of these operations might be performed by a different system, but they all have to be performed if the purchase transaction is to be completed properly. None can be left out, or the overall transaction may fail.

The notion of two or more processes or steps, all of which must be successfully completed, is the basic idea of a *transaction* in the sense that the term is used in information technology. The key idea is that updates occur in two or more places, and these updates must collectively be treated as a single indivisible operation (see Figure 6.1). The individual steps may occur sequentially or concurrently based on the transactional structure of the system(s), but all of the steps must be completed before the transaction is determined to be completed.

Examples of IT transactions include the following:

- Updating two accounts to effect a transfer of funds
- Performing the necessary operations to process a purchase
- Telling an operational system to do something, while simultaneously making the appropriate entries to the accounting system
- Sending a command to a computer-controlled machine, while recording the fact that the command was sent in a log

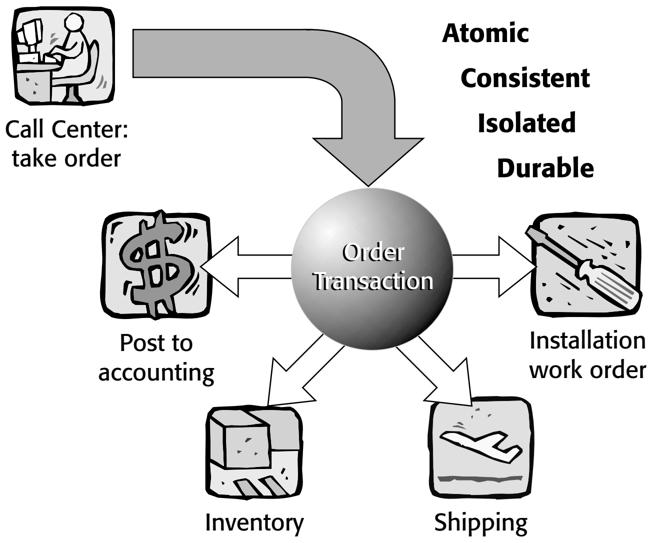


Figure 6.1 A transaction implements several business functions as a single unit.

transactions

A transaction is the implementation of one or more business functions based on associated business rules, where the transaction is completed only when all of the required business functions are completed as specified by the business rules.

A transaction, taken as a whole, must exhibit what are known as the *ACID* properties: atomicity, consistency, isolation, and durability (and a derived capability called roll-back).

Atomicity. Until twentieth-century scientists started splitting them, chemical atoms were thought to be indivisible. For us, atomicity means that the transaction is indivisible. The transaction consists of parts, to be sure, like the protons and neutrons of an atom, but those parts cannot be separated from one another. It's all or nothing.

Consistency. An IT transaction will normally update several files or databases, possibly in different subsystems or different parts of the enterprise. Because the different systems on which portions of the transaction are processed may be physically far apart, it isn't feasible to ensure that the updates occur at precisely the same instant. Nevertheless, processes outside the transaction, which may be looking at the same files or databases, should never be able to see an intermediate inconsistent state. For example, in the case of transferring funds between accounts, at some point one account may have been updated while the other has not been, as viewed from inside the transaction. Outside operations, though, must never see this temporary inconsistency. A transaction always takes a database from one consistent state to another consistent state; that is, it obeys all the integrity rules and moves to a new consistent state, or if it does not complete then it remains in the previous consistent state.

Isolation. In a big enterprise, there's always a lot going one. In a bank, hundreds, or even thousands, of customers may be simultaneously performing transactions at ATMs. A husband and wife may even be trying simultaneously to access the same account from ATMs in different parts of town. The isolation property says that all these simultaneous transactions must not affect one another; that is, they must be *isolated.* In the case of the husband and wife both accessing their account, isolation implies that the transactions must be treated as occurring sequentially, rather than overlapping.

Durability. A transaction's results must be durable, that is, safely recorded on some nonvolatile medium. A manufacturing company's systems can't "forget" that an order has been received, even if the computer crashes just after the transaction has been completed. As a practical matter, durability generally means that the information associated with the transaction is recorded on disk, rather than just residing in the computer's memory. In more critical transactions, to prevent the possibility of information loss due to a disk failure, stronger measures must be taken to ensure durability, such as writing separate copies of the information to separate disks or creating a journal entry from which the transaction can be re-created if need be.

Rollback. Taken together, the ACID properties imply that a transaction must have a *rollback* capability. If all goes well, all the parts of a transaction will complete successfully. The results will be atomic, consistent, isolated, and durable. But suppose all doesn't go well? What if one part of the transaction proceeds normally, but a second part fails? Or suppose we don't know what happened to some part of a transaction that is supposed to be handled on a remote computer system because the communication lines to the remote system went down? Because the transaction was supposed to be atomic—all or nothing—failure of one part means that the transaction as a whole has failed. Because we can't have all, we must accept nothing! And because the results must be consistent, the part of the transaction that appeared to have gone normally must be undone, or rolled back.

ACID properties

Atomicity, consistency, isolation, and durability are defined as the ACID properties of software.

The characteristics of a transaction can interact on complex ways. For example, because a transaction must be both consistent and durable, it must ensure that all durable stores are updated as a part of the transaction, not just one. Because transactions must be isolated and consistent, if an operation outside the transaction queries information involved in the transaction, the response to the query must report the state of the information either before the transaction began or after it completed. If the latter approach is taken (that is, the results reflect the state of the information after the transaction completes), the query must be suspended until the transaction completes. In addition, a system involved in a transaction must be able to restore the previous state of the information that it is responsible for, even if it has successfully performed an update—because a failure in some other system may cause the whole transaction to be rolled back.

The ACID properties are technical criteria that describe the requirements for a transactional system to operate reliably, but business requirements must also be taken into account in defining what constitutes a transaction. For example, let's return to the situation of the husband and wife simultaneously accessing their bank accounts from two ATMs in different parts of town (see Figure 6.2). Suppose the husband wants to withdraw $100 from their joint checking account. The wife intends to check the checking account balance, and if the balance is above a certain amount, transfer some funds to their savings account. From a business perspective, how should the bank manage these overlapping activities? Certainly the withdrawal and the transfer should each be transactions and exhibit the ACID properties. They can be treated as isolated operations that occur in some order. And assuming both the withdrawal and the transfer are made, it doesn't really matter what the order is; the net effect on the checking and savings balances is the same.

But what about the wife's checking the balance before she decides whether to make the transfer? Should that be part of her transaction, or not? From the wife's perspective, it

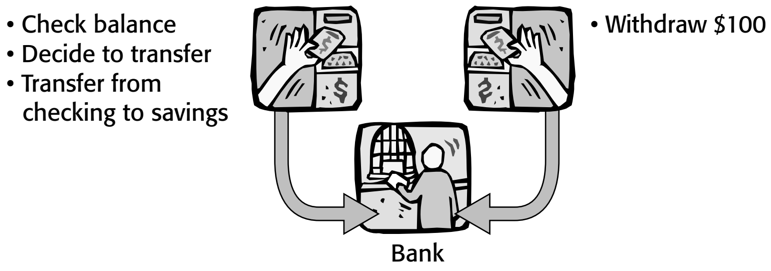


Figure 6.2 A transaction for an ATM system must exhibit ACID properties.

makes sense that the balance inquiry should be part of the transaction. If it's not, then it is possible (unlikely, but possible) that the husband's withdrawal transaction (a single atomic transaction) occurs exactly between her balance inquiry (a transaction) and her transfer (a second transaction). The final result will be mystifying to the wife, who will see a final balance in the checking account that is $100 less than she expected. Perhaps she would not have made the transfer if she had known!

The husband's perspective, however, is that he just wants to get his cash. He doesn't want his transaction to be slowed down because updates on his account have been inexplicably (from his viewpoint) locked out.

The bank's perspective is that it wants to maintain the integrity of its accounts. That integrity is ensured by the ACID properties of the withdrawal and transfer transactions. Of course, the bank also wants to keep its customers happy. And in this case, it may view the situation as being so rare that it's not worthwhile to develop special programming to handle it. Nevertheless, the analysis that the bank must perform in designing the ATM system illustrates the trade-offs that must be addressed in defining transactions.

Transaction Processing Monitors

Concern for the transactional integrity of systems is not limited to integration of existing applications or to distributed systems. Products that help ensure the ACID properties of transactions within a single application are a major part of the IT landscape. Applications based on database management systems (DBMS), for example, often need to update several tables or files in the course of a single transaction. Because each of these updates is a separate operation on the database, the DBMS must provide transaction management capabilities. DBMS products also often provide journaling capabilities. In the event of a severe problem like a disk failure, journaling allows the DBMS to be restored to a known state from a back-up file, and all transactions that occurred between the time of the back-up and the time of the failure can be applied.

Mainframe transaction monitors and distributed transaction processing monitors essentially perform the same task but they are based on different technical architectures.

Mainframe Transaction Monitors

Two transaction management systems are widely used in IBM mainframe environments to maintain the transactional integrity of applications:

- Information Management System (IMS)
- Customer Information Control System (CICS)

IMS tightly integrates database processing and transaction processing to achieve high performance. IMS includes its own hierarchical database, although applications utilizing the IMS transaction monitor can also utilize IBM's DB2 database. (And, conversely, applications using the CICS transaction monitor can access IMS databases.) IMS applications are based on queued message processing, and IMS incorporates its own message queuing capabilities. IMS integrates with IBM mainframe security capabilities such as Resource Access Control Facility (RACF). IBM has recently introduced capabilities for Web-based front-ends to access IMS applications on the mainframe; IMS applications themselves, however, are limited to the mainframe. The key advantage of IMS is its ability to support very high transaction rates. This factor, plus the inherent value of legacy IMS programs that would be difficult to convert to other technologies, continues to make integration with IMS an important EAI topic.

As the dominant transaction monitor on the dominant mainframe platform, CICS is the most widely used transaction monitor in IT today. CICS has a number of significant strengths. It is highly scalable: Many CICS applications support thousands of concurrent users. Response time in a properly designed CICS application is excellent. CICS applications leverage the extreme reliability of the mainframe, and CICS is itself highly reliable. This makes CICS well suited to "365 × 24" applications—applications that must be available all the time, every day.

CICS services and their application programming interfaces (APIs) significantly simplify the programming of large-scale applications. These applications are often complex, not because of the inherent complexity of each individual transaction, but because the application must multiplex many concurrent transactions in support of hundreds, or even thousands, of users. With CICS, programmers can focus primarily on the logic for a single transaction; CICS handles the multiplexing.

CICS supports a variety of databases. On the mainframe, CICS security capabilities are integrated with mainframe security services like RACF. CICS security provides control over the functions a given user can access. Authority to run specific transactions can be assigned, as can access to specific resources.

Originally developed as a host-terminal environment, CICS now supports transactions with peer and client applications on PCs, workstations, and midrange servers, both from IBM and from other vendors. It also supports numerous specialized devices, like ATM terminals. CICS servers can run on a variety of platforms, and they can coordinate transactions across platforms.

Due to its dominant position, CICS is often a factor in EAI projects. New applications must access the data and services provided by CICS-based applications. In addition, many mainframe-hosted packaged applications run under CICS control. EAI architects can choose two general approaches to dealing with CICS legacies:

- Remain within the CICS environment for transaction management
- Integrate CICS with a transaction monitor on a Unix or Windows NT platform

IBM offers versions of CICS on midrange platforms, but these versions do not necessarily offer all the services that mainframe CICS offers. Frequently, utilizing a bridge between CICS and another transaction monitor such as Tuxedo or Encina may be a preferred approach.

Distributed Transaction Processing Monitors

IMS and CICS originated on the mainframe and retain a strong single processor flavor. As might be expected, coordinating transactions across distributed systems raises significant problems that are not encountered in a mainframe-centered environment. As distributed, networked systems proliferate, TPMs specifically designed to cope with these problems have become more prominent. The two best known products in this category are BEA Systems' Tuxedo and Transarc's Encina. Before discussing these products, however, it is useful briefly to consider the problems of distributed transaction handling and the Open Group Distributed Transaction Processing (DTP) standard.

To achieve the ACID properties in a highly distributed environment, numerous difficulties must be overcome:

- A transaction may run across multiple computers.
- The environment may be heterogeneous, comprising platforms from a number of vendors.
- Processors may crash, or individual applications running on a platform may fail.
  - In the mainframe environment, when an application fails, the operating system knows that it failed; CICS can leverage this information to handle rollback.
  - In the distributed environment, when an application fails, it may simply "disappear." A distributed transaction monitor must discover the disappearance and initiate rollback. The rollback process itself is potentially complicated by the disparate nature of the platforms participating in the transaction.
- Mainframe transaction monitors were designed to run over dedicated networks running IBM's robust (and expensive) SNA protocols; transaction monitors designed for open distributed networks utilized the less robust TCP/IP.
  - Nowadays, however, mainframe networks often are based on TCP/IP as well, and the mainframe transaction monitors support the open-system protocol (and TCP/IP has become more reliable). So this last factor has become less significant.
- In addition to the technical differences, distributed transaction systems operate in a different commercial environment than mainframe transaction systems.

- On the IBM mainframe, the MVS operating system, CICS and IMS, and IBM's DBMS products were all designed to work with one another. Independent vendors must conform to IBM's interfaces.
- In the open-systems world, the playing field is more level, and vendors of, say, operating systems and DBMSs are more equal partners. As a result, the open-systems environment is more dependent on standards than the mainframe environment.

The *Open Group DTP standard* forms the basis for open-systems distributed transaction processing. The DTP standard defines four types of participants in distributed transactions (see Figure 6.3):

Applications. Programs that participate in transactions.

Resource managers (RM). Control shared resources such as databases.

Transaction managers (TM). Coordinate transactions.

Communications resource managers (CRM). Control the communications between the other participants.

Distributed Transaction Processing standard

An open industry standard that specifies how transactions are to be coordinated. The DTP is widely supported by databases and TPMs.

An application initiates and completes transactions by interacting with the TM, using the TX protocol. RMs handle resources (such as database records) that are utilized by applications in the course of a transaction; interactions between applications and RMs are carried via the normal native interface to the RM (e.g., SQL for a database). A CRM provides a means of communicating with participants in the transaction; three protocols are available for coordination between the application and the CRM: XATMI,

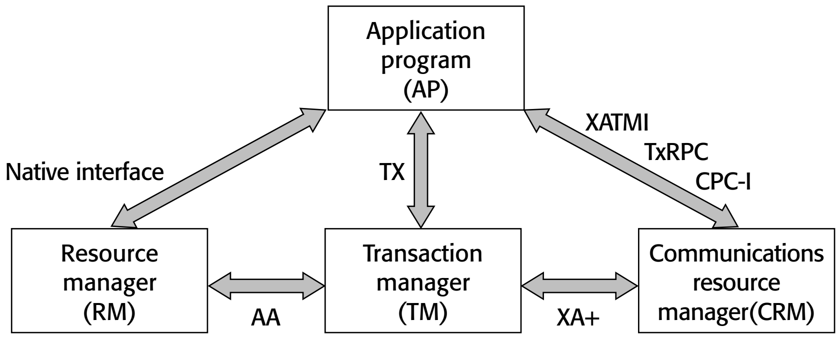


Figure 6.3 The Open Group Distributed Transaction Processing model defines standards for distributed transactions.

txRPC, and CPI-C (which reflect the designs of Tuxedo, Encina, and CICS, respectively). A CRM communicates with the TM via the XA+ protocol. Finally, the TM coordinates with the resource manager via the XA protocol. The most important aspect of these protocols is the concept of the *two-phase commit.*

Recall that, in order for a transaction to exhibit the ACID properties, no participant in the transaction can complete (that is, commit to) updates unless all participants can do so. Coordinating this process across systems that may be running on widely separated platforms, connected by possibly unreliable communications links, is the essential problem in distributed transaction management. The solution in the DTP standard is the *two-phase commit*. The details of the two-phase commit protocol are beyond the scope of this book, but it's worth taking a brief look at the protocol to appreciate some of the issues involved.

two-phase commit

The mechanism used by the DTP to ensure the integrity of distributed transactions.

In the two-phase commit, a transaction manager is responsible for coordinating the transaction. The transaction manager interacts with resource managers. The transaction manager starts the two-phase commit process by issuing a *prepare-to-commit* instruction to all participants in the transaction. Each resource manager responds by indicating whether it can commit its portion of the transaction. If all goes well, all the resource managers will respond that they can commit. In this case, the transaction manager instructs the resource managers to commit. If a resource manager cannot commit, it will roll back its part of the transaction and inform the transaction manager that it cannot commit. The transaction manager then instructs all the other resource managers to roll back their portions of the transaction.

Transaction Monitor Services

In addition to support for transaction management via the two-phase commit protocol, transaction monitor products typically offer a variety of other capabilities. Taken together, these capabilities are intended to provide the system integrator with all the features needed to build high-integrity distributed systems.

Programming models. In the course of a transaction, the participating applications may need to have significant interactions with one another. Transaction monitors can support a variety of programming models to mediate these interactions. These may include the following:

- *Synchronous request/response*. In this model, a client application requests a service from a server application. Processing within the client is suspended while it waits for the server to respond. A client may interact with a server several distinct times, or with several distinct servers, within the context of a single transaction.

- *Asynchronous request/response*. In this model, a client sends a request to a server, then proceeds with other processing without waiting for a response. After the server completes its processing, the client can retrieve the results using a special interface. This model allows a client to interact with several servers in parallel, thus possibly reducing the processing time for the transaction as a whole.
- *Conversational.* In this model, two applications exchange messages in a "conversation." This mechanism treats the two applications as peers, rather than as a client and a server. Because the overhead associated with setting up a connection is not repeated, a conversational mode interaction can be more efficient than a series of request/reply interactions.
- *Events*. In this model, applications communicate by declaring that certain named events have occurred. Other applications can register, or subscribe, to the event name and are informed whenever the event occurs. Additional information can be carried with the event. The event model provides one-way communication, as opposed to the two-way communication of the request/reply and conversational models. Events are highly flexible because the declarer of an event does not need to know the identities of event subscribers, and because declarers and subscribers for a given event type can be added or removed dynamically. In addition, because notification of an event can be sent to many subscribers, this model is useful for broadcast communications (such as a notification to prepare for system shutdown).

Message queuing. Message queuing was described in Chapter 4, "Messaging Architecture and Solutions." Transaction monitor products sometimes provide message queuing in addition to their other services. A transaction monitor can easily support the transactional aspects of queuing. For example, a messaging-based architecture for a mission-critical application must treat the *enqueue* and *dequeue* operations as transactions; that is, the action of adding a message to, or removing a message from, a queue must be atomic, consistent, isolated, and durable. Moreover, if an application fails while enqueuing or dequeuing a message, the messaging system should roll back the enqueue/dequeue operation.

Security. Transaction monitors can provide various security services. A transaction monitor may control which users can access certain servers or certain functions on a server. A transaction monitor may also provide a built-in encryption capability or an interface to an external encryption utility.

Conversion and translation. Because distributed applications often tie together different types of computers, which may employ different data representations, a capability to translate between representations is sometimes provided by transaction monitors. This capability is similar to the marshalling provided by ORBs, which was described in Chapter 5, "Object Architecture and Solutions."

Server management. Some transaction monitors provide capabilities to start servers when their capabilities are requested or restart servers when they fail. They may also provide a capability to assign servers to selected machines or to balance loads by running servers on less heavily utilized machines.

Administration. Distributed transaction systems are complex, so transaction monitors provide administrative tools to help users set up and manage these systems. Transaction monitors also provide interfaces to system management frameworks such as Hewlett-Packard OpenView, Computer Associates Unicenter, or Tivoli Management Environment (TME). Because the applications involved in a large enterprise may be extensive, transaction monitors also provide mechanisms for dividing administrative responsibilities. Tuxedo, for example, allows distributed systems to be allocated to administrative groupings called domains.

Gateways. For EAI, we need to connect disparate systems. The systems to be integrated often were implemented using an existing transaction monitor—for MVS mainframe applications, usually IMS or CICS. Transaction monitors intended for EAI usually include some capability to integrate IMS or CICS applications with newly developed applications.

Distributed Transactions: Encina and Tuxedo

Two major products, BEA Systems' Tuxedo and IBM/Transarc's Encina, are the most prominent products oriented to open distributed transactions.

Encina supports a variety of Unix platforms as well as Windows NT. Encina is layered on DCE and is closely tied in many ways to DCE technology. Structurally, Encina is configured as a toolkit, giving designers a great deal of flexibility in architecting their transactional systems. Encina provides load balancing, scheduling, and fault tolerance across platforms. It integrates with Kerberos to provide security capabilities. Large-scale Encina installations can be structured as a set of *cells*, which are similar in concept to DCE cells. Each cell can be managed independently. Management capabilities include starting, stopping, configuring, and monitoring servers. Encina applications can be integrated with CICS via a peer-to-peer communication gateway. Through its support for the DTP XA protocol, Encina can manage transactions involving a variety of third-party XA-compliant resource managers, including most major RDBMS implementations. A variety of programming models is supported, including RPC, object invocation, and peer-to-peer. Message queuing is supported via a queuing service toolkit. To provide across-the-board transaction services, IBM first bundled Encina with CICS under the name TXSeries; more recently, Encina has been integrated with IBM's WebSphere product.

BEA Systems' Tuxedo is probably the best known and most widely used open system transaction monitor. Tuxedo is offered on most of the popular Unix platforms and on Windows NT. Tuxedo is build around queues; various programming models are implemented on top of the queuing capability, including RPC (synchronous and asynchronous), conversational, and event-based. Information sent between clients and servers via Tuxedo can include strings, structured types such as C structures or COBOL records, and name/value property sets (which Tuxedo calls *fielded buffers*). Tuxedo does not include an integral security capability, but it does support a Kerberos-like security interface that allows an external security service to be integrated. A full set of management capabilities is included, including monitoring, load balancing and scheduling, application configuration, and application restart. Tuxedo supports distributed administration and management via *domains*, which communicate with each other through gateways. CICS and IMS applications can be integrated with Tuxedo applications via a BEA Connect gateway.

Tuxedo and Encina provide comparable features, and they are being aggressively integrated into the BEA Systems and IBM product lines, respectively. Thus, the choice between the two is often based not so much on their specific features as on an enterprise's preferred vendor and on the capabilities of the entire product suite.

Object Transaction Monitors

In earlier chapters, we discussed application integration based on messaging and distributed objects. In this chapter, we have presented transactional messaging and shown how many transaction monitors incorporate messaging. The question naturally arises, do object technologies integrate with transactional technology? Indeed they do, and a new class of products called *object transaction monitors* (OTMs) has arisen that provide this capability.

object transaction monitor

An object transaction monitor is a product that provides distributed objects with transactional integrity.

In principle, the concept of an OTM is straightforward. An object, as understood in CORBA or Java, is viewed as a participant in a distributed transaction. The ACID properties must be maintained across the entire transaction, which may involve a number of objects. The state of an individual object is considered part of the overall state managed by the transaction, and it is this aggregate state that changes atomically, must remain internally consistent, and must be isolated and durable. Thus objects must be able to coordinate their activities, and if a transaction is aborted, the objects must be capable of being rolled back to their former states.

There are several types of object transaction monitors available:

- CORBA OTS
- EJB
- MTS

Let's take a look at each of these.

CORBA Object Transaction Service

The CORBA Object Transaction Service (OTS) specification defines how CORBA-based systems should support transactional behavior. OTS is based on the Open Group DTP model, and it is intended to be consistent with it (see Figure 6.4). OTS envisions inter-operation between objects and non-OO servers in a transaction, and it expects that an XA-compliant resource manager such as a relational DBMS should be integratable into a transactional CORBA-based application.

Object Transaction Service

Object Transaction Service is the CORBA standard for distributed object transactions. OTS defines how the Open Group DTP model can be applied to CORBA objects.

The OTS "objectifies" the transaction process. OTS identifies the following types of objects:

Transactional client. An arbitrary program that can invoke operations within a transaction context on a transactional object.

Transactional object. An object that can participate in a transaction. As always with CORBA, a transactional object can become a client and invoke another transactional object.

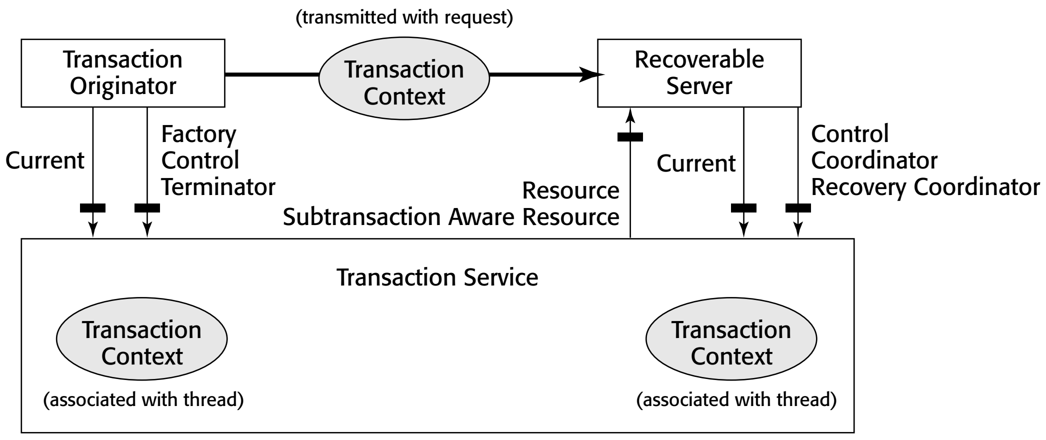


Figure 6.4 The CORBA Object Transaction Service combines distributed transaction and object technologies.

Recoverable objects. Transactional objects that participate in transaction completion (either commit or rollback). A recoverable object maintains some state information that is updated by the transaction.

Transactional server. A server that supports a transactional object.

Recoverable server. A server that supports a recoverable object.

The client that begins a transaction is called the *transaction originator.* As calls propagate from object to object, the OTS must also propagate a *transaction context*. Transaction context can be propagated either explicitly, that is, by passing a parameter that represents the transaction context, or implicitly by the ORB's handling the transaction context as part of object invocation. Note that it is possible for a transaction object not to be a recoverable object if its state is not affected by the completion of the transaction.

As an example, consider an order processing integration application in which inventory and customer account applications are integrated. A transactional client application may run on a user's workstation. This client may invoke a processing object that handles the transaction by invoking the inventory and customer account applications. This processing object may not have any persistent state; it is simply a transient object that comes into existence to coordinate a sale and then no longer exists. It is a transactional but not a recoverable object. In contrast, the objects that represent a given customer's account and the inventory do have persistent state, and they are recoverable objects. The transactional and recoverable objects execute within transactional and recoverable servers, respectively.

The OTS extends the Open Group DTP model in one major way, in that it defines *nested transactions*. As the name implies, a nested transaction is a transaction within a transaction. Nested transactions increase the flexibility of distributed transactions because in the event of failure in a subtransaction, only the subtransaction needs to be recovered, not the whole transaction. OTS implementations, however, are not required to support nested transactions.

nested transactions

A nested transaction is a transaction within a transaction.

Virtually all CORBA vendors provide an implementation of the OTS, in most cases building on existing transaction monitor products. The BEA WebLogic Enterprise Server, for example, integrates Tuxedo with BEA's ORB. The IBM WebSphere Application Server Enterprise Edition integrates with Encina. Iona Technologies' OrbixOTM is based on an earlier version of the Encina transaction manager, which Iona licensed from Transarc and has since developed independently. (Note that the OrbixOTM does not include the entire Encina product.) Hitachi TPBroker integrates its Open TP1 transaction monitor.

An OTS implementation that is built on top of a transaction monitor based on the Open Group DTP model facilitates integration of distributed object transactions with resources or applications that are not object-oriented, as illustrated in Figure 6.5.

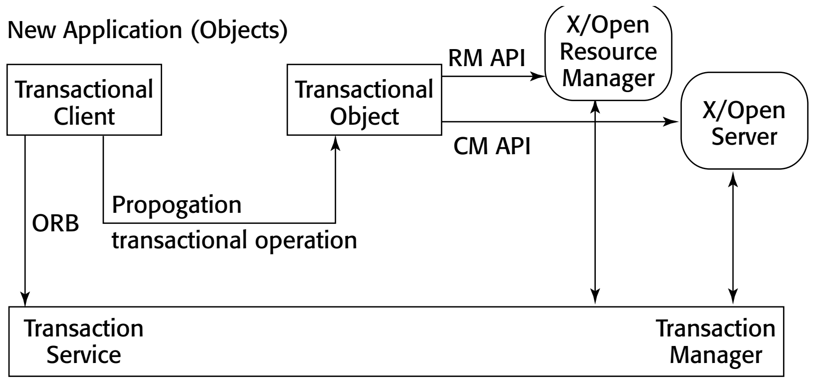


Figure 6.5 Integrating CORBA OTS with non-OO transaction participants.

In this case, a CORBA transactional client initiates a transaction through the OTS, which is implemented on top of a transaction manager (not necessarily a full-fledged transaction monitor). The client then invokes a transactional object; OTS manages the transaction context. The transactional object then updates a relational DBMS that supports the XA protocol. Database operations that are part of the transaction are then coordinated between XA and the transaction manager.

Enterprise JavaBeans

The Enterprise JavaBeans (EJB) component model is strongly transactional. Indeed, compliant EJB implementations are required to support transactions, although they do not necessarily support all four ACID properties. EJBs may be either *session* beans, which are not persistent, or *entity* beans, for which persistence is required. A transactional session bean may not be recoverable following a system crash, so it is not durable in the usual sense of the term; it could, however, support the atomicity, consistency, and isolatability properties.

session bean

Session beans are EJBs that do not maintain a persistent state.

entity bean

Entity beans are EJBs that maintain a persistent state.

Java and EJB are based on the idea of leveraging existing standards, and their handling of transactions is consistent with this approach. EJB implementations are encouraged (although not required) to use the Java Transaction Service (JTS), which is a Java binding to the CORBA OTS. Similarly, Java Remote Method Invocation (RMI) can be implemented on top of CORBA IIOP. An EJB implementation utilizing JTS and RMI/IIOP should be interoperable with CORBA and OTS, which will allow EJBs to access existing CORBA services or CORBA wrappers of legacy applications.

JTS

Java Transaction Service specifies the implementation of a transaction manager that supports the Java Transaction API and implements the Java mapping of the OMG Object Transaction Service at the low level. JTS uses the standard CORBA ORB/TS interfaces and Internet Inter-ORB Protocol (IIOP) for transaction context propagation between JTS transaction managers.

RMI

Remote Method Invocation allows Java programs to call C and other native programming languages and vice versa.

IIOP

Internet Inter-Orb Protocol defines how applications that support CORBA communicate with one another. Java applications can also support IIOP, thus allowing them to interoperate with CORBA services.

The EJB specification tries to minimize the amount of time that programmers spend writing support for complicated system services like transactions, as opposed to writing business logic. The specification therefore requires EJB containers to provide transparent support for transaction management. Clients cannot access EJBs directly; instead they access an EJB object wrapper, which intercepts operation invocations before the bean-specific code is invoked. The wrapper is capable of handling all transaction management, although the programmer can assume control if necessary.

The wrapper/interceptor design of EJB containers allows for another key feature, *declarative specification* of bean properties, including the transaction rules applicable to the bean. The transactional mode for a bean need not be programmed; it can simply be declared when the bean is installed in a container. Declarative information is stored in a Deployment Descriptor. The EJB container then automatically manages transaction initiation, commitment, and rollback. The EJB model supports six transactional modes:

Bean managed. The programmer assumes responsibility for transaction management.

Not supported. The bean cannot execute within a transaction. If the bean is invoked within a transaction, the transaction is suspended.

Supported. The bean can run with or without a transaction context.

Required. The bean requires a transaction context. If the bean is invoked outside a transaction, a new transaction is started.

Requires new. The bean must execute within a new transaction. If the bean is invoked within a transaction, the existing transaction is suspended. In any case, a new transaction is initiated.

Mandatory. The bean requires an existing transaction context. If the bean is invoked outside a transaction, the request fails.

As with the CORBA OTS, it is expected that most EJB implementations will leverage existing capabilities by building on either an existing transaction manager or a data management capability that supports transactions.

Microsoft Transaction Service

Microsoft Transaction Service (MTS) is rapidly becoming a major contender in the field of object transaction monitors. MTS supports, and is tightly integrated with, Microsoft's COM+, which is evolved from its Distributed Component Object Model (DCOM). MTS is also highly integrated with the MSMQ message queuing service. MTS supports integration with CICS.

Like EJB, MTS/DCOM provides a container system for COM components, which is similar to EJB containers (although not as function-rich). MTS provides a Distributed Transaction Coordinator (DTC), which provides similar capabilities to the transaction manager in a transaction monitor based on the Open Group DTP model. MTS supports XA-compliant resource managers, so MTS applications can integrate most major relational DBMS products.

Like EJB, MTS allows transactional behavior to be specified declaratively. Transaction management can be handled by MTS through an interceptor, minimizing or eliminating user-written transactional code. Transactional modes are similar to those in EJB.

Distributed Transactions in EAI

TPMs have both strengths and weaknesses for EAI. Among their strengths are the following:

- Ensuring the integrity of business transactions
- Handling complex coordination that very few developers would be able to program on their own
- Depending on the specific transaction monitor, providing a variety of additional services that reduce developer effort and shorten time to market

Using a transaction monitor in an EAI application also has some drawbacks. These include the following:

- Part of a transaction may be too tightly coupled.
- A transaction may take too long to complete, preventing other processes from accessing resources that are tied up by the transaction.
- It may be difficult to introduce the transactional paradigm to some legacy applications

The first two of these three drawbacks are a general concern in working with transactional systems. The last is more specific to the problem of integrating existing legacy applications.

It is in the nature of a transactional system that the various parts of a transaction are very tightly coupled. If any portion of a distributed transaction cannot commit, the entire transaction must be rolled back. The transactional application as a whole is therefore only as strong as its weakest link. That weakest link may be a relatively unreliable server or a failure-prone communications link that makes it likely that a server will be inaccessible when it is needed for a transaction.* If a transactional application involves many participants that are themselves not extremely robust, the overall application may have very poor availability.

The situation of the husband and wife accessing their bank accounts from ATMs in different parts of town, which we discussed at the beginning of this chapter, illustrates the problem of a transaction taking too long to complete and tying up resources. Recall that the husband simply wanted to get some cash from the ATM, whereas the wife wanted to check a balance, then possibly make a transfer based on the results of the balance check. The resource in contention in this case is the checking account, and if this resource is tied up by the wife while she is considering whether to make the transfer, the husband's response time will suffer. Although this example may seem artificial, the general problem of a transaction tying up resources for too long is not. Due to this kind of problem, as a general rule designers should minimize the scope of transactions to minimize the effects of lockout on other processes that might be trying to access the same resource. Note that in many cases, as in our example of the husband and wife, the design decision must be evaluated at the level of business rules as well as at a technical level.

When integrating legacy applications, the system architect must consider carefully how difficult it will be to have the legacies adhere to the rules of distributed transactions. Servers that are to participate in transactions must be able to play their role in the two-phase commit process. They must be able to roll back their activities when necessary. But the legacy application may have been designed with no notion of rollback, and it can be extremely difficult to introduce this concept into an existing application. Even if the legacy application itself is transactional, as in the case of an application that uses transactions to coordinate updates to an application-specific database, it may not

*Some transaction systems support a concept called heuristic commit, which allows a transaction to complete even when a resource is unavailable. For example, if communications were lost, an application could decide to commit its portion of a transaction. This allows transactions to complete, at the risk that the integrity of the overall system could be compromised.

easily support accepting instructions from a higher-level transaction manager. Full transaction management is a laudable goal, but in the real world it is often difficult to achieve.

Alternatives to Full Transaction Management

What alternatives to full transaction management are available to an architect in designing EAI systems? Two approaches merit consideration when it appears there may be problems either with the robustness of the integrated application or with getting one or more of the legacies to play by the rules of the transactional game. The first is to adopt a messaging approach; the second is to implement "soft rollbacks." Note that these are not alternative techniques; either or both could be used as the situation warrants.

Messaging

The distributed transaction model assumes that all elements of the transaction have to be fully completed before the entire transaction can complete. If any part of the transaction cannot commit, the entire transaction must be rolled back. Often, however, it is sufficient to be sure that each part of the transaction either has completed *or is certain to be completed.* In an order processing environment, for example, it is necessary to be certain that an order will be fulfilled, but it is not necessary for every step in the fulfillment process to have been completed before the initial order transaction completes. A messaging system can provide the necessary assurance because a robust messaging system guarantees that once a message has been enqueued, it will eventually be delivered. This allows us to, in effect, substitute sending a message to a server, knowing that it will eventually complete its part of the transaction, for it is actually completing its part. In effect, we decouple the single transaction into multiple transactions: a front-end transaction, part of which involves placing one or more messages on appropriate queues, and one or more back-end transactions, each of which dequeues and handles a message (see Figure 6.6).

This approach works because the messaging operations are themselves transactional. That is, when enqueuing a message is considered part of a transaction, we know that if that transaction commits, the message will definitely have been placed on the queue and that this operation is both consistent and durable. If the computer on which the messaging system runs subsequently crashes, we are confident that when the messaging system is brought back up, the message will still be in the queue—because the enqueue operation was durable. Similarly, if the dequeue operation is part of a back-end transaction, we know that it will be atomic, consistent, isolated, and durable. So if the back-end application crashes after (apparently) dequeuing the message and its transaction is therefore rolled back, we know that when the application is rerun, the message will still be in the queue because the dequeue operation will have been rolled back. And the message will remain in the queue until the back-end transaction fully completes (i.e., commits), at which time it will be permanently and appropriately removed from the queue.

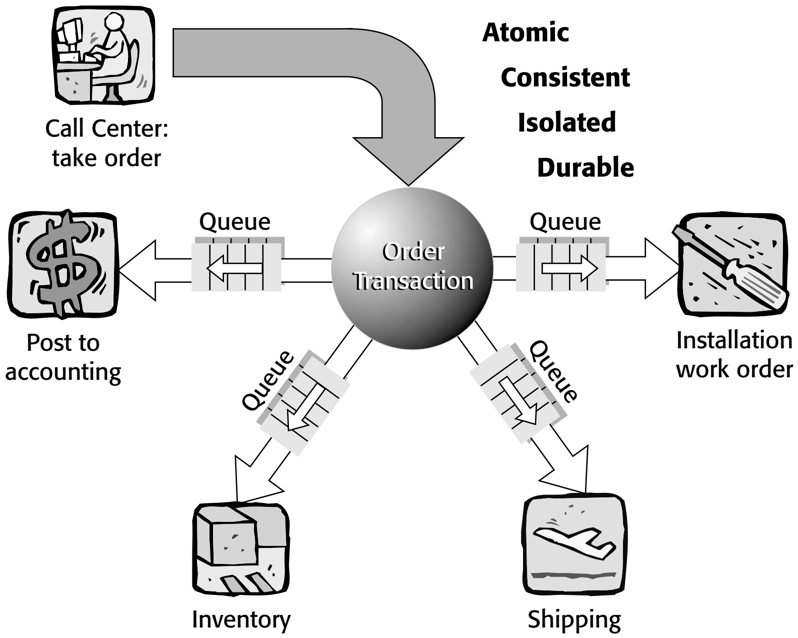


Figure 6.6 Messaging-based transactions increase system flexibility.

In order to use the approach of splitting a transaction into multiple message-linked transactions, we must first verify two conditions. The first is that the enqueue/dequeue operations are, in fact, transactional. That is, the messaging system must be robust enough for our purposes, and it must support participating in transactions—at the "ends." In addition, the messaging system must be instructed to handle the message properly because some messaging systems have both a fully transactional mode in which the message is durably stored on disk and a less robust—but also less expensive—mode in which the ACID properties are not guaranteed.

The second condition that must be verified in order to use messages is that the integrity of the overall system will not be compromised by deferring the completion of some parts of the business process.This is ultimately a business judgment. Suppose, for example, that one part of an EAI project is to ensure that customer address information stored in several legacy systems is consistent. Updating an address might be considered a transaction, or we might consider using messaging. If messaging is used, then there could be a period of time when the address has been updated in some legacy applications, but not in others. Strictly speaking, consistency has been compromised. But this may be judged an acceptable compromise in order to achieve acceptable performance.

Soft Rollback

A second approach to mitigating the drawbacks of full transaction management is the soft rollback. In this case, a task that might logically be considered part of a transaction is not included, because to do so might unacceptably affect performance. Perhaps the excluded task is handled at a remote location, and the communication links to that location are poor. Including the task as a formal part of the transaction might frequently cause entire transactions to be aborted.

In a situation like this, it may be possible to exclude the problem task from the transaction, provided the results of the other tasks can be "undone" at a later time. We refer to such "undo" operations as a *soft rollback.* Soft rollback is not a novel concept. In traditional manual accounting methods, for example, postings to an account, even if erroneous, are never simply erased. Instead, a later journal entry is made to reverse the effects of the error. The soft rollback applies a similar concept to the effects of a transaction.

Several conditions must apply for the soft rollback strategy to be effective. First, there must be a viable undo operation for all the other tasks in the transaction. Some operations simply cannot be undone. Second, there must be a reliable means of determining that the task that was excluded from the transaction could not complete and that therefore the other tasks must be undone. Finally, we must ensure that no other activities were initiated as a result of the erroneous completion of the original task; that is, there must be no ripple effects that will themselves have to be undone. It is difficult to ensure reliably that all these conditions hold, so the soft rollback strategy should be used with extreme caution. Nevertheless, because it is sometimes very difficult to get legacy applications to function in accordance with the rules of distributed transactions, this strategy should be part of the EAI designer's toolkit.

Applying Transaction Architecture to MegaMoney Bancorp

Let's return now to the problem of MegaMoney Bancorp. In Chapter 4, we showed how MegaMoney could integrate their applications using message queues, and in Chapter 5, we discussed how the same problem could be solved using distributed-object technology. Transaction monitors provide yet another way of solving this problem. Indeed, because TPMs such as Tuxedo and Encina support request/reply and messaging paradigms, the architectures that we discussed in the earlier chapters can be implemented using TPMs as well. And OTM middleware provides both the benefits of object technology and those of transactional technology.

Using an OTM, MegaMoney can do the following:

- Provide an object-oriented architecture, which minimizes the amount of code that MegaMoney's programmers have to write. This, in turn, reduces the chance for errors and allows MegaMoney to bring its applications to market sooner.
- Ensure the transactional integrity of its services.
- Utilize messaging to interface with back-end applications that cannot support synchronous integration or to smooth out loads.

Given these benefits, why not simply adopt an OTM-based solution in all cases? While OTMs are indeed likely to be the wave of the future, there are several reasons for caution. One is that the technology is complex, presenting tricky design decisions. Architects should use care to avoid exercising every feature of an OTM product, just "because it is there." Another concern is that because the technology is new, skilled practitioners are scarce. Enterprises will have to emphasize training and carefully monitor projects to ensure that they don't outrun the skills of the development staff. Finally, because the technology is new, the maturity of products can be problematic. Products that claim to support a standard may, in fact, support only a subset of the standard's features. Rigorous, hands-on technology evaluations should be conducted before development projects commit to using new OTM products.

Final Thoughts

In this chapter we have examined the role of transactions in EAI applications. A transaction is a single logical unit of work that is composed of subunits, all of which must complete successfully in order for the transaction to be successful. Transactions exhibit the ACID properties of atomicity, consistency, isolatability, and durability. In order to preserve these properties in the face of the possible failure of one of a transaction's subunits, the transaction must have a rollback capability.

Transactional systems are usually implemented via a transaction monitor. These middleware products handle the details of ensuring that applications support the ACID properties. Transaction monitors like IMS and CICS are widely used in mainframe applications.

In distributed applications, transaction monitors that support the Open Group's Distributed Transaction Processing standard are usually employed. Leading products in this category are BEA's Tuxedo and IBM/Transarc's Encina. These transaction monitors implement a two-phase commit process to manage distributed transactions. They also provide a variety of other services including support for multiple programming models, transactional message queuing, security, conversion and translation, management and administration, and gateways to CICS or IMS.

Object transaction monitors are becoming increasingly prominent. CORBA-based OTMs, EJB implementations, and MTS offer most of the strengths of traditional transaction monitors, with the added value of distributed object support. In EAI applications, OTMs make it feasible to wrap legacy systems to present object-/component-based interfaces while providing the high levels of reliability and robustness needed for mission-critical applications.

Even though distributed transaction systems are extremely important to effective EAI solutions, they have some drawbacks that must be handled carefully in developing an EAI design. Use of messaging and soft rollback strategies can mitigate some of these drawbacks.

In the last three chapters, we have described how EAI architectures can be developed using three technologies: messaging, distributed objects, and transaction monitors. To create EAI solutions within the enterprise, technology alone is far from sufficient. In the next several chapters, we discuss how to apply EAI. We will begin in the next chapter by discussing the topic enterprise IT architecture and considering why establishing and managing an enterprise architecture is critical to effective EAI. Then in Chapter 8, "Effective EAI Development: The SAIM Methodology," we describe a methodology called the Secure Application Integration Methodology (SAIM), which has been specifically developed to facilitate EAI development.

CHAPTER 7

Enterprise Architecture

Key Facts

- Architecture is critical to the enterprise.
- Architecture must be enterprise-wide.
- Enterprise architecture must model existing and planned applications and systems.
- Enterprise architecture must indicate reusable components and points of reuse.
- Enterprise architecture must provide information sharing in a seamless manner across the enterprise.

To understand the rationale for EAI it is necessary to understand both the enterprise requirements for a solution and the options offered by EAI that can make the requirements achievable. No one EAI approach (messaging, distributed objects, or transaction monitors) is likely to meet all the needs of a complex enterprise. If individual projects choose their own technologies, the result is a hodge-podge of incompatible technologies. The solution is an enterprise architecture, which assesses integration needs across the enterprise and establishes technology standards and guidelines for the enterprise.

This chapter discusses the requirements of an enterprise architecture. Because an architecture can mean different things to different people, this chapter also covers why it is important to define an architecture and how to make sure that your architecture meets the enterprise needs.

Enterprise Requirements

Many organizations are incapable of sharing key business information across the enterprise because they suffer from stovepipe information systems. Stovepipe systems have no interoperability, which inhibits key business information sharing (see Figure 7.1).This reduces both the business's ability to service customers and the business's

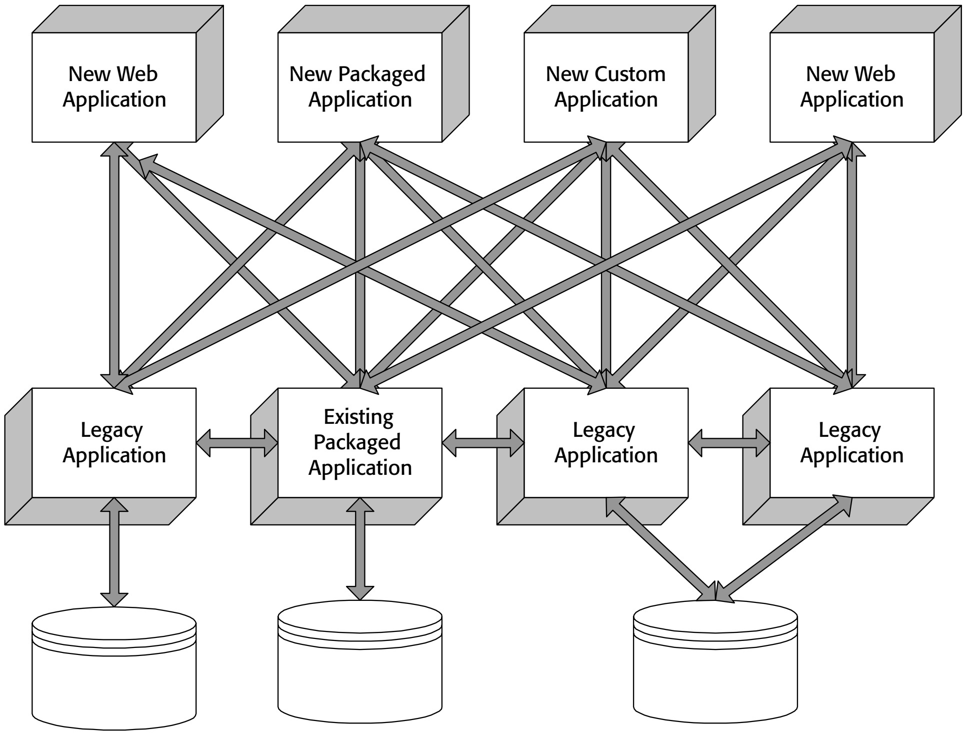


Figure 7.1 Typical enterprise information systems are stovepipes.

competitiveness in the marketplace. Many companies talk about enterprise approaches, such as a "single view of the customer," but few know how to implement it. The reality is that although the company may not intend to, it has taken, and continues to take, a stovepipe approach to the information management problem, which results in the existence of stovepipe systems in legacy technologies such as COBOL, Ada, or C, as well as in the more advanced software programming environments such as Java and Enterprise JavaBeans (EJB). The solution is to develop and implement an enterprise architecture approach to application and systems development.

Because the amount of legacy technology in use is so high, the sheer number of systems is a strong rationale to do nothing more than attempt to connect a few critical stovepipe systems. Companies rationalize that this EAI approach is more favorable; to change all the legacy systems is an unlikely solution that has all of the following drawbacks:

- It involves too much risk.
- It is too expensive.
- The compay suffers from a lack of IT skills.
- It takes too long to realize benefits.

These are rational arguments if the solution to achieving an enterprise-wide information system is to replace all major systems; this is both unnecessary and impossible. Each of these key concerns can be addressed with a pragmatic enterprise-wide architecture that promotes information sharing.

Achieving the Architecture

It has taken a couple of decades of hard experience for major corporations to realize that replacing entire large-scale stovepipe systems is not an effective enterprise solution. With the advent of connector, gateway, and messaging technologies, however, it has quickly become apparent that the stovepipe systems can be interconnected with new components to achieve critical information sharing.

There are three ways for information systems to provide an enterprise solution, as shown in Figure 7.2:

Extend existing applications. Normal advancement of business practices will always require existing applications to be extended to support the new business rules and data.

Add new technology solutions. New technologies can enable new ways of doing business, such as Internet technology for online application access to improve information flow to and from customers and staff.

Connect existing applications and new technology solutions. To share information across the various corporate applications and systems, the various gateways and message queuing software provides a practical approach.

From an enterprise perspective, where information sharing with minimal software development and expense is the key goal, the only time an application would need to be replaced is when one or more of the following criteria apply:

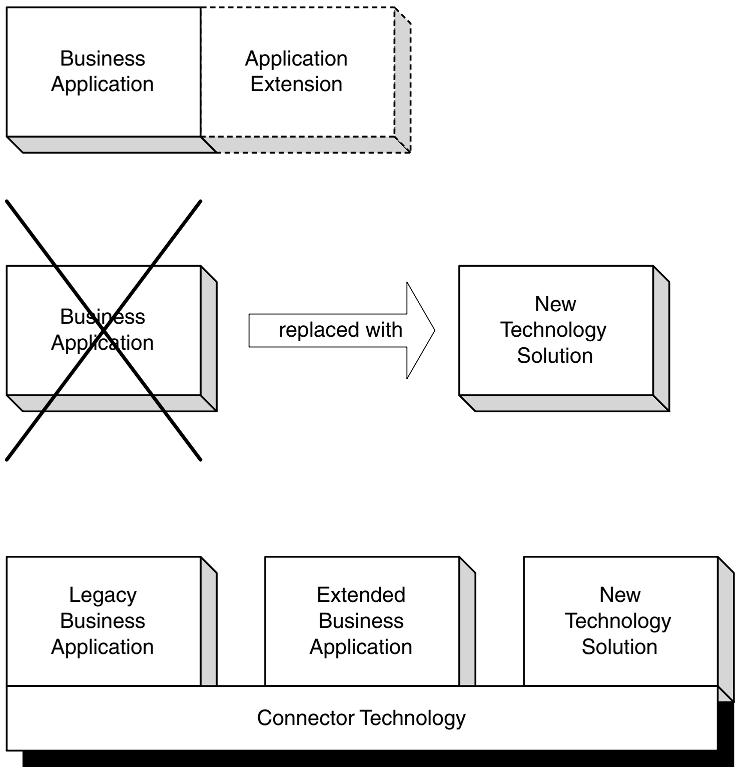


Figure 7.2 There are three basic ways in which to provide an enterprise solutions.

The existing application technology no longer fits the rest of the business support system technologies. For example, this situation occurs when a stovepipe application is written in Visual Basic and C++, but the remainder of the newer, Web-enabled applications all are written in Java.

The interoperability with the other enterprise systems is poor because of the legacy architecture. A mainframe COBOL customer management system is accessible only from a 3270 emulator that has to be screen scraped for data to be shared with client/server systems.

The degree of extension required makes it cheaper to replace than extend. A client/server insurance system provides only new customer and prequotation functionality. It needs to be extended to include quotations, policy renewals, and insurance claims.

Reuse is another criteria that adds to the complexity of achieving enterprise architecture. While reusing current systems and applications is favorable, reuse can be expensive and elusive and does not always bring business benefits. It is an important part of enterprise architecture when it can provide a critically needed central business function. The additional cost of developing reusable software, however, can exceed the benefits if its use depends on extensive rewrites of existing systems.

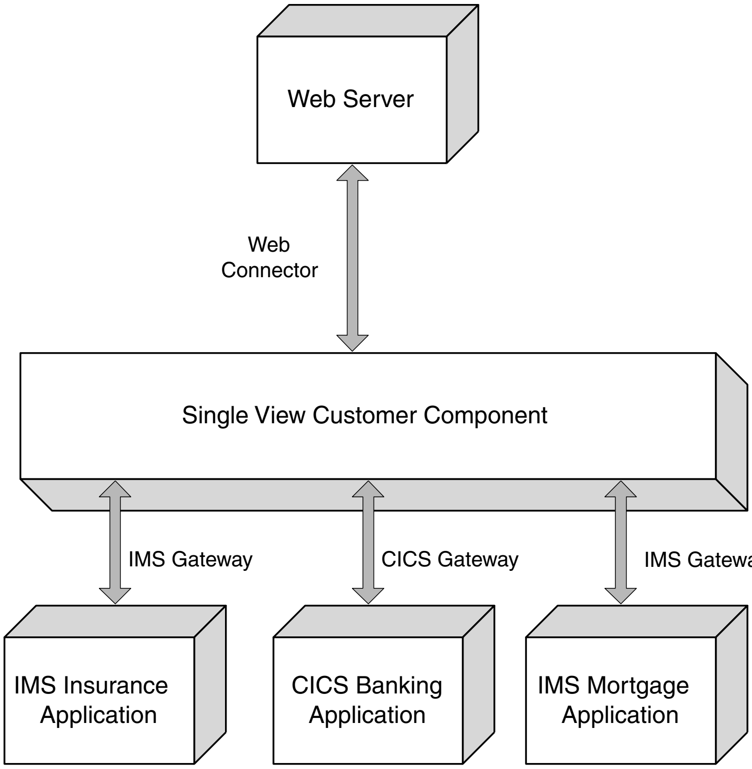


Figure 7.3 Component-based reuse should be aimed at the enterprise.

Reuse is more pragmatic when aimed at the enterprise rather than within a line of business. For example, accounting applications exist in most lines of business of an enterprise. Reuse will be most effective when introducing new components that add functionality and complement the functionality of existing applications (see Figure 7.3). This type of reuse can take the form of an application-layer component, which provides business support functions such as handling customer information, or a middleware-layer component, which provides basic services that applications use, such as a data conversion service.

EAI is highly complex because it should provide solutions that fully support the enterprise business processes in a seamless manner. This would be straightforward if all applications were to be defined and built from scratch. But the reality is often that only some of the business processes are supported by applications and that many of the systems are stovepipes, built using varying technologies that do not easily interoperate. This means that EAI solutions must support the following:

- Implementing new applications
- Implementing reusable enterprise components
- Extending existing, often legacy, applications

- Interconnecting all enterprise-critical systems, regardless of their technology or interoperability

To manage the complexity of the EAI task an architectural approach is critical for success. This means being able not only to clearly state what applications and systems already exist, but to produce the architectural blueprints for the new applications, enterprise components, and required interconnectivity.

Defining the Architecture

Architecture can mean different things to different people. The perspectives vary between user and development roles at different levels within the enterprise, such as the following:

- CEO and line-of-business executives, who see architecture as the business processes required to effectively run the enterprise
- Application and system users, who see architecture as individual applications that enable their work within the enterprise
- CIO and enterprise architects, who see architecture as software that collectively supports the enterprise business processes
- Software developers, who see architecture as the solution structure and the integration policy for one or more applications

It is critical to define the enterprise architecture at an appropriate level of detail. There are many perspectives that can define and constrain an architecture and that are limited only by the roles within IT and the business. For example, a user may want new functionality to make her job easier; this requires an IT analyst who can identify the need for an extension to an existing application. A CEO may need summarized information that must be extracted and filtered from multiple systems; this requires an architect who can define connections from the specified applications to a new application.

Consider the objectives of an architecture that must support the perspectives taken by both the user community and development:

- Understanding what exists in terms of business functions and data
- Defining extensions to what exists
- Defining new components and applications
- Defining connectors between applications

It is vital that the enterprise architecture define the business and technical architecture at the enterprise level and still drive the production of individual application architectures. Implementing an enterprise architecture should allow you to do the following:

- Scope implementation sets
- Define outline interfaces and specification

- Select specific Commercial Off-The-Shelf (COTS) systems
- Identify an implementation sequence

COTS

COTS are packaged applications that are available off–the-shelf for end-user use, such as Microsoft Office.

The enterprise architecture itself must not only identify existing software, extensions, new components, and applications, but also provide a common definition for the following:

- The structure of the enterprise in terms of these software artifacts
- The interaction of these artifacts to support the business objectives
- The interfaces specification to support the required interactions

The following sections discuss the role of architecture in solving business problems.

Considerations with Modern Architectures

In recent years the approach to architecture has moved from client/server to n-tier. N-tier systems contain layers of software that provide incremental layers of abstraction for more flexible and adaptable systems, as shown in Figure 7.4. These systems may contain the following components:

Thin clients. Reduces the need for powerful desktops.

Application servers. Provide centralized business logic.

Web access. Enables e-business.

Legacy interconnectivity. Makes stovepipe data sharable.

Centralized systems management. Reduces the complexity of applications by providing centralized services.

Secure transactions. Ensure privacy of information.

An effective modern architecture will have many layers of software, each of which offers a particular type of service to the layer above, as shown in Figure 7.4. Once the intermediate infrastructure layers have been built, they can be reused across all new developments in order to save time and money.

The architecture must then integrate all the layers. Recall from Chapter 2, "Types of Integration," that there are three types of integration that are available to an EAI architect, depending on the integration requirement:

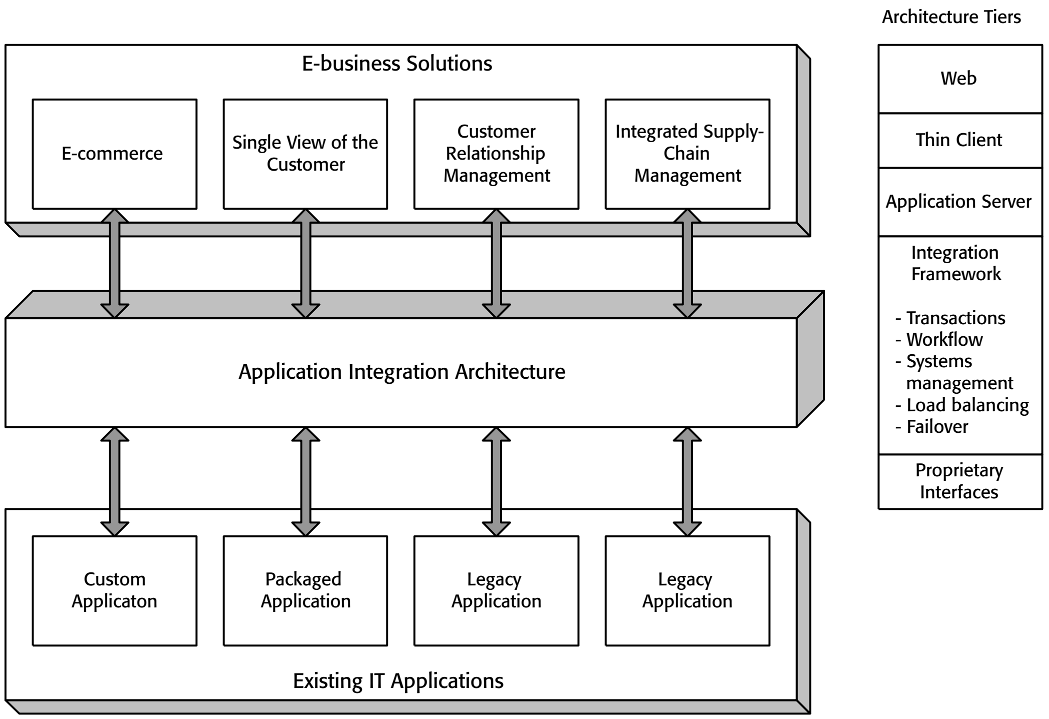


Figure 7.4 N-tiered architecture is becoming the norm.

Presentation Integration. Provides a single view of multiple applications to business users.

Functionality Integration. Provides full interoperability between applications for business users.

Data Integration. Provides centralized data access for business users.

Table 7.1 is a summary of integration requirements for each of these integration types.

The integration options must be combined within the architecture as part-solutions to support the business goals. The architecture should use integration wherever possible because the legacy applications are then leveraged to the advantage of the business, as shown in Figure 7.5.

Modeling the Enterprise as an Architecture

It is crucial that everyone understand the architecture. This may be difficult because an architecture can have a variety of views, for example, conceptual, physical, structure, or flow. It is critical to clearly define the notation by which you express the enterprise architecture to ensure that the various aspects can be defined in a single and useful

Table 7.1 Summary of Integration Requirements

TYPE OF INTEGRATION	REQUIREMENT FOR USE
Presentation	Shared front-end
	Need to update different data sources from single front-end
Functionality	Application processing logic required to interpret data from different applications
	Addition of processing logic required to integrate functionality from different applications
	Transactional integrity between applications is required
Data	Need to update data from multiple sources
	Data needs to be synchronized between databases

manner. Modeling the architecture with a combination of pictures and descriptive text is the easiest way to define the architecture and depict each of the aspects of the architecture.

What is modeled, and to what degree, varies due to the nature of the requirement. For example, existing applications are usually represented architecturally as large grained solutions arrived at by various technology approaches because it is important to know their functionality and data handling. Large grained refers to the high level of abstraction

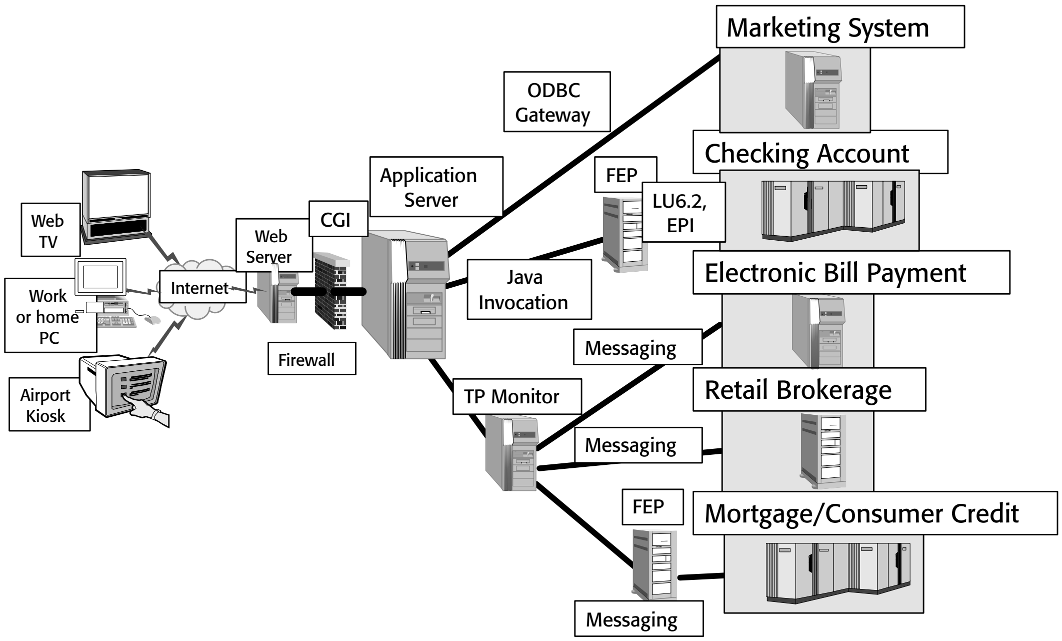


Figure 7.5 Legacy integration is part of the EAI solution.

that can be used to define the application data and functionality. This is fine for the internals of existing software but not for the interfaces, which should be represented in a fine-grained manner.

large-grained

Large-grained refers to the ability to provide a view of software as a collection of components, with only the primary data and functions identified.

Connectors require interfaces providing functionality and data and should be modeled in a fine-grained manner. In other words, the architectural details should be captured at the object level, clearly stating the data (attributes) and functionality (methods), as well as the interface sequence. New technology, including connectors, usually have very abstract and logical architectures and code-level interface documentation. This is difficult to model because of the extremes of information available; ideally an intermediate specification format is required.

fine-grained

Fine-grained refers to the ability to provide a view of software as a cooperative set of objects, with object-level attributes (data) and methods (functions) identified.

The modeling techniques you use to state the existing and the new software solutions must be compatible with each other and useful to those expected to validate and work with the models. For example, if older applications are defined using data models and new ones using object models then it is very difficult to have a similar understanding of both to enable architectural assessment; it would be like comparing cabbages and popcorn. All architecture should use a consistent modeling technique to enable and demonstrate effective understanding. This is extremely important. It is critical to use a standard notation to model architectures in a clear and consistent manner. The formalized methodologies of past and present have been primarily useful for analysis and design phases of software development. Most existing approaches that have some support for architecture modeling do not go far enough in addressing the various architectural modeling requirements identified in Table 7.2.

Currently there is only one standardized way that provides the ability to model the varying views and degrees of granularity required to be stated by a software architecture: the Unified Modeling Language (UML). The UML is a standardized approach, adopted by the Object Management Group (OMG), for visually modeling software from concept to implementation specification. The UML has grown to be an accepted technique for modeling architectures at enterprise, application, and middleware levels because it is

Table 7.2 Enterprise Architecture Criteria

MODELING CRITERIA	SOFTWARE ARCHITECTURE ITEM	ARCHITECTURAL MODELING REQUIREMENT
Existing applications and COTS solutions	Existing functions Existing data Existing interfaces	Coarse granularity Not modular Not object-oriented structure Proprietary interfaces
Enhancement to existing applications	Extension of functions Extension of data Extension of interfaces	Fine granularity Modular Mix of object-oriented and existing structure Proprietary and object-oriented interfaces
New technology applications	Components	Modular Object-oriented structure Object-oriented interfaces
Technology integration	Connectors	Modular Unspecified structure Object-oriented interfaces

extendable enough to represent components, interfaces, and objects at the varying degrees of granularity required as defined by an architecture. This makes UML the obvious choice for business analysis, technical architecture modeling, and detailed design.

Enterprise Components and Architectural Layers

The basic unit of the architecture model must be able to represent different views of software. The best unit for this is the component. A component is a software entity that provides a cohesive set of functional capabilities through a specified interface. A significant attribute of a component is its *granularity*—that is, its size and the scope of the capabilities that it provides. Very fine-grained components (perhaps resulting from extending the decomposition tree above several additional levels) could be classes in C++ or Java. From an enterprise architecture perspective, we are interested in extremely coarse-grained components, which are significant systems in their own right. For example, an enterprise's entire financial information system could be considered a component.

component

A component is a software entity that provides a cohesive set of functional capabilities through a specified interface.

Components are not necessarily developed using object orientation (although as a rule we think newly developed components should be), and enterprise architecture components are usually too large to be viewed as a single object. A component does, however, have a well-specified interface, and distributed-object technology provides the best available means of defining and controlling component interfaces.

An important characteristic of components is that they are reusable. We use the term *reusable* in two senses. The most important sense from the perspective of the enterprise architecture is a reusable common service, which is an executing service that persistently maintains some data of interest to multiple applications (e.g., an inventory component). The second sense is that of traditional software design and code reuse, in which a given piece of code is reused, but executing instances are separate (e.g., a stack component). In some cases, both senses of reuse are applicable. For example, a separate instance of a reusable inventory component could exist at each warehouse in an enterprise, or a component may be reused by embedding it into several otherwise unrelated components.

The better modern architectures allocate components between layers, or tiers, each of which serve a specific purpose to which the components within must conform. Figure 7.6 demonstrates how component layers are used in an enterprise architecture. Such components can support a very wide range of applications; that is, each layer represents a subset of the enterprise architecture that focuses on providing a cohesive set of capabilities to end users or system developers.

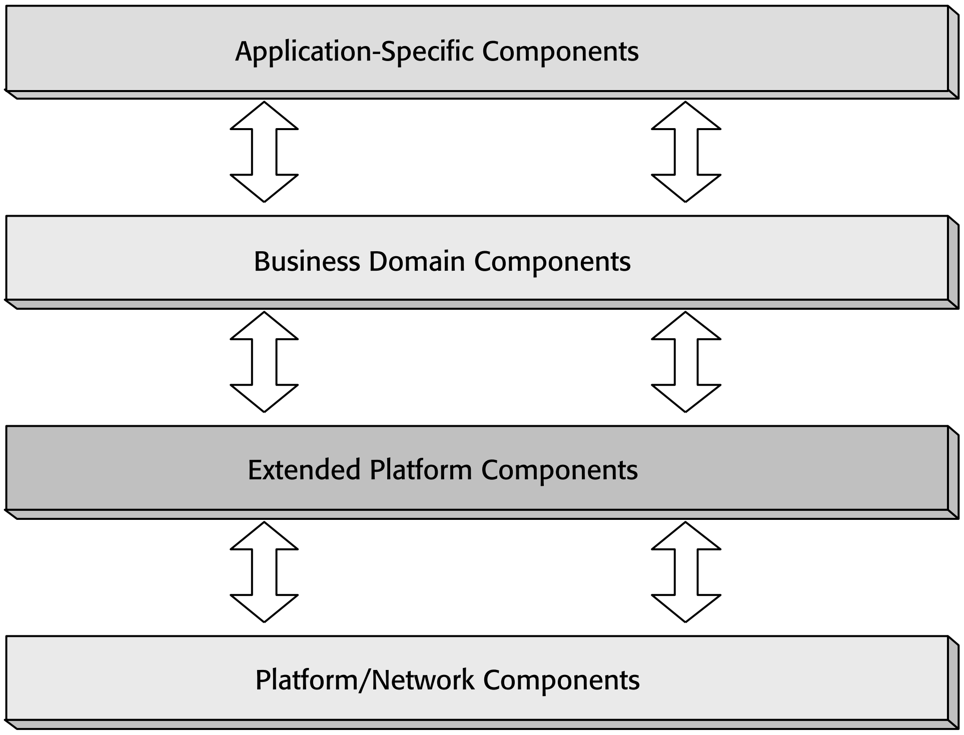

Application-Specific Components
Business Domain Components
Extended Platform Components
Platform/Network Components

Figure 7.6 Component layers are essential in an enterprise architecture.

There are three component layers of enterprise architecture:

- Business domain
- Extended platform
- Platform

Business domain components are the heart of the enterprise architecture. They represent the fundamental business entities and processes of the enterprise. The key property of business domain components is that they are *shared* across applications.

The layers from the business domain components down are the focus of the enterprise architecture because they define areas where the enterprise can most effectively reuse capabilities and because they determine how well the architecture will be able to evolve and adapt to change.

Platform components are the basic services on which applications rely to support their business logic. These are numerous and varied in nature, ranging from system administration services to persistence mechanisms and data conversion services. There is no finite list of platform components. They are defined by the common services required by a set of applications in the enterprise.

Extended platform components include Object Request Brokers, database management systems, Web servers, application servers, and transaction monitors. While more specific than platform components, extended platform components are still independent of the enterprise and the business domain. Over time, successful extended platform components become regarded as essential and tend to migrate into the platform components. For example, a few years ago a TCP/IP communications stack might have been considered part of the virtual platform, but now it is tightly integrated with the operating system.

The set of enterprise architecture components, such as business domain components, extended platform components, and platform components, are typically involved in most applications. Some of these components are shared with other applications, and some of these are not; the shared components are called *application-specific components*.

The vertical partitioning of the architecture into sets of components reflects generality, but more importantly reflects the concept of dependence. That is, components lower in the architecture are *independent* of higher-level components, and vice versa. Independence of the lower layers from the upper ones is a key architectural consideration because it helps to ensure that the architecture as a whole is adaptable and modifiable. When proper dependency relationships are maintained, upper-layer components can be modified easily without requiring any changes in lower-level components. In addition, lower-level components can be modified or replaced without any impact on the upper layers, provided that the new or enhanced components support the same interface as the components they replace or upgrade.

Commercial lower-layer products will almost always be independent of higher-level custom code, but a number of dependency relationships must be carefully considered to ensure a robust architecture:

Dependencies between platform and extended platform components. For example, is an ORB supported on only certain operating systems? Similar considerations apply to dependencies between commercial application packages and platform/extended platform components.

Dependencies between business domain and application-specific components. Perhaps the greatest barrier to reuse of supposedly multipurpose components is that they are too tightly tied to a particular application.

Layer skipping (or "tunneling"). This refers to a case where a component interfaces directly with components further down in the architecture, not just with components in the layer immediately below itself. Some tunneling is unavoidable (for example, application-specific components often have to depend to some extent on the operating system), but it must be limited if the architecture is to be adaptable.

The enterprise architecture needs to represent the multiple layers of software and their purposes. Each architecture must be defined in terms of these layers, as shown in Table 7.3.

Existing Applications as Components

Before defining required application extensions it is critical to clearly model what exists. Existing application models should represent the applications at a coarse grain,

Table 7.3 Defining Architectural Layers

ARCHITECTURE	LAYERS DEFINED
Existing system	Application Platform/network
Extensions to existing system	Application Platform/network
New technology component	Application Business domain Extended platform Platform/network
Connector	Business domain Extended platform Platform/network
Security	Application Business domain Extended platform Platform/network
Enterprise	Application Business domain Extended platform Platform/network

or a high level of detail, as shown in Figure 7.7, because there is little return on investing significant effort to model them in great detail because the detail is never needed or used. The objective is to define the architecture at an enterprise level, not an application level.

A legacy application should have two views depicted at the same time:

- Logical
- Physical

The logical view shows the key software purposes of the internal functionality and their primary interrelationships. The physical view shows available software interfaces for interoperating with the application (note that some interfaces will be human interfaces and these entry points should also be modeled).

Extending Applications

Extending applications is often less expensive than replacing them. Assuming that the existing business functions and the associated enterprise data for an application have been architecturally documented, it is important to identify the nature of the extension:

- Additional functionality using existing application data, such as to produce derived data, such as a "customer financial score"
- Additional data, such as adding data fields to handle product information
- Additional data and functionality, such as new processing logic that handles data not already supported

The architecture should detail the extensions, differentiating the future from existing aspects of each application, as shown in Figure 7.8. Modeling at any greater level of detail, such as an object model, at this stage will return no advantages or benefits but waste time and money.

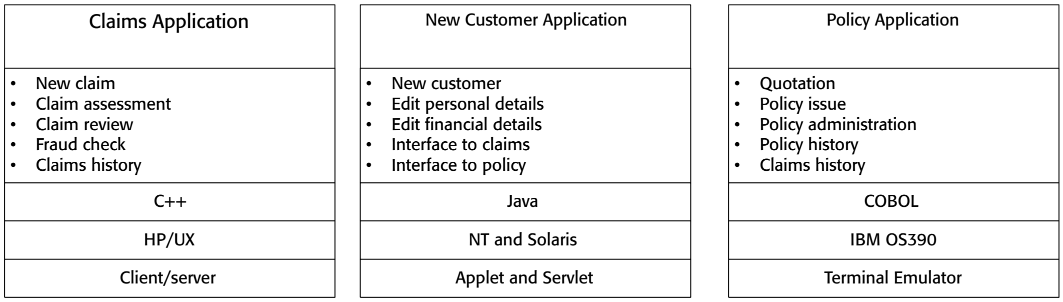


Figure 7.7 Existing applications should be modeled as components.

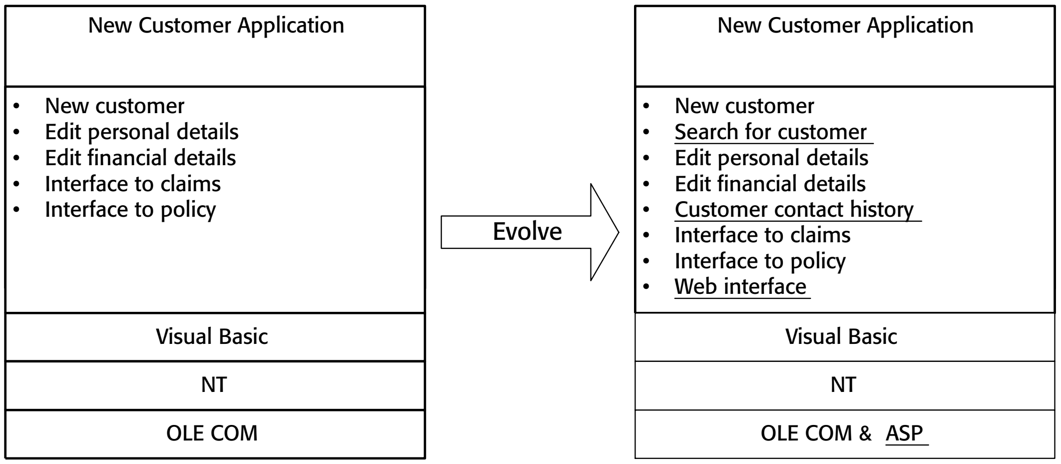


Figure 7.8 Extensions to existing applications are part of the required EAI model.

Identifying Component Interface Specifications

The interface specifications will be abstract in nature when dealing with applications, components, and connectors at an enterprise level. Each interface specification should state the following:

- The specific functionality that can be invoked
- The specific data supported by each function
- The interface invocation sequence

The notation should be simple and clear to allow a snapshot of the existing applications and extensions. Figure 7.9 gives an example of interface specifications.

Adding New Components

The architecture needs to identify the new technology components. These will deliver new functionality and data that is not appropriate as part of a legacy application extension. Keep in mind that any projects that are in progress must be included as future solution components. These new components should preferably be defined as object-oriented components, which will ensure a more flexible architecture. The components should be allocated to layers or tiers, as shown previously in Figure 7.4. The layers should clearly identify the interconnectivity between the components, both within the same layer and between layers, as shown in Figure 7.10.

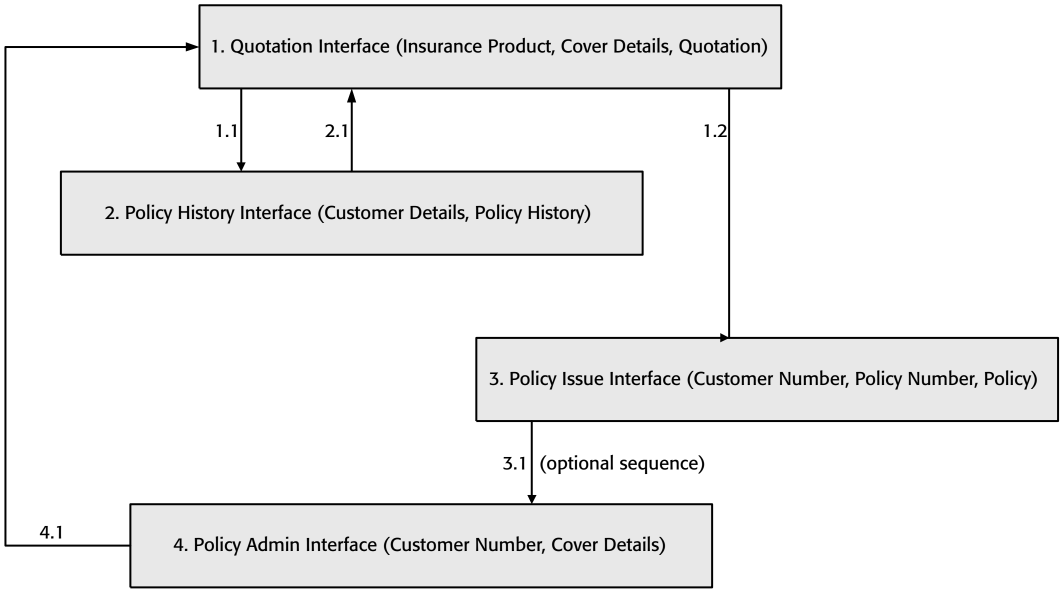

1. Quotation Interface (Insurance Product, Cover Details, Quotation)
1.1
2.1
1.2
2. Policy History Interface (Customer Details, Policy History)
3. Policy Issue Interface (Customer Number, Policy Number, Policy)
3.1 (optional sequence)
4.1
4. Policy Admin Interface (Customer Number, Cover Details)

Figure 7.9 Interface specification is critical to EAI.

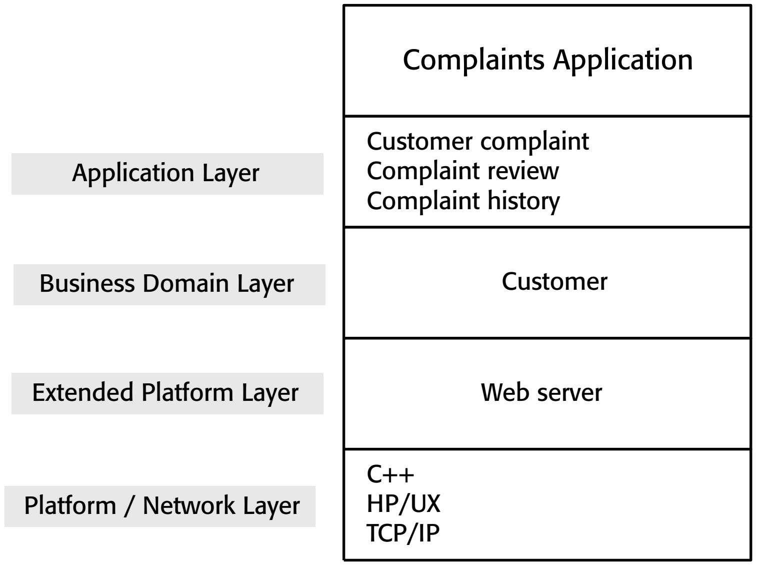


Figure 7.10 EAI architecture must identify interconnectivity between component layers.

Wrapping Applications and COTS

Often the lack of technology compatibility between legacy applications, new technology components, and COTS is sufficient to prohibit their interoperability unless an abstraction layer is implemented. A wrapper is a small component that hides a software implementation, such as a COTS product or legacy application, which provides an abstracted interface that simplifies integration. Effectively mapping an application or COTS product should make it appear as another enterprise component. A *wrapper* should do the following:

- Provide consistent access
- Transparency of the technology of the wrapped functionality and data
- Hide complexity
- Provide a foundation for frameworks and enterprise objects as reusable components
- Provide a connection point for linking to other software

In the case of COTS, particularly where it is immature, the wrapper must be designed to cope with a level of abstraction that reduces the impact of lack of stability and frequent bug-fix releases (see Figure 7.11).

Adding Connectors between Components

In reality, *connector* is a generic term for some middleware technology that can enable connection of data and functionality between two different technology applications, or a client and server.

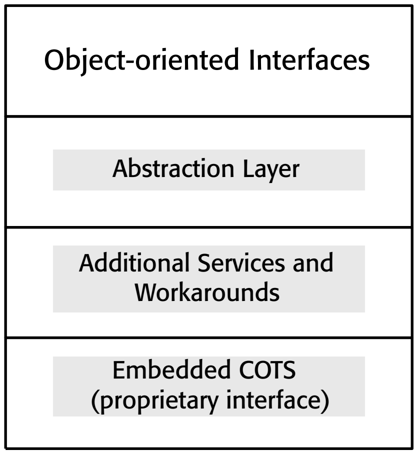

Object-oriented Interfaces
Abstraction Layer
Additional Services and Workarounds
Embedded COTS (proprietary interface)

Figure 7.11 COTS and their wrappers must be modeled within the EAI architecture.

connectors

Connectors are software components that enable interconnection between applications. They provide the connection to each application (adapter) and the "pipe" between the connectors.

There are three basic parts of each connector, as shown in Figure 7.12:

- Two physical adapters for the specific technology of each application, or the client and server. These can be off-the-shelf adapters or customized for specific technologies.
- One middleware framework that supports the interconnection. This is usually in the form of gateway; either language-based, such as Java-CICS, or operating system-based, such NT and AIX.

Off-the-shelf adapters are technology-specific and application/programming language-specific. Custom adapters can be either for a new technology / application / programming language or business domain-oriented, supporting a set of business objects. Custom adapters are usually built by end-user organizations using adapter toolkits. These should be treated as custom interface components that enable interconnectivity of the application for which they are built.

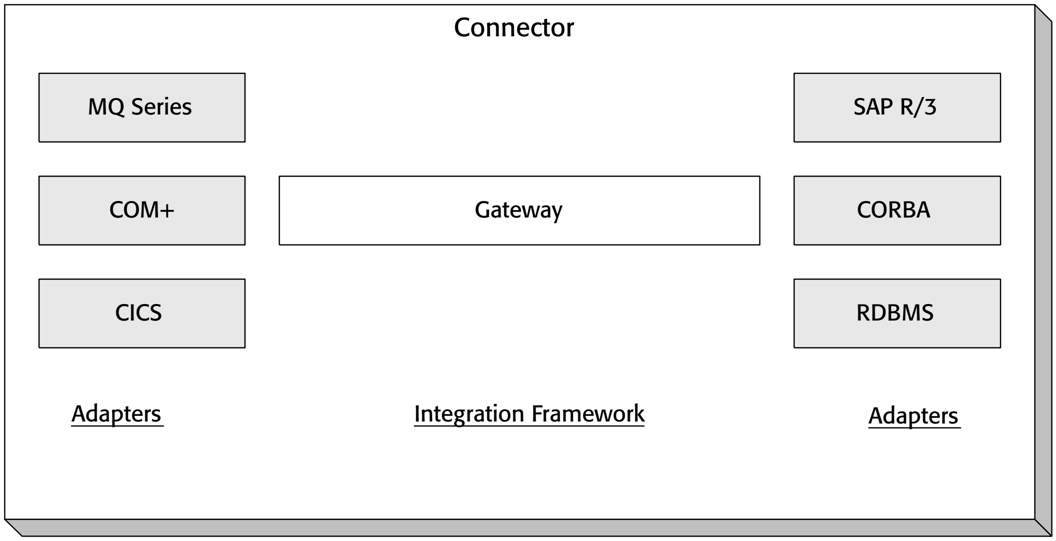


Figure 7.12 A connector interconnects applications as part of an EAI solution.

Dealing with Enterprise Security

Security is a critical line item in EAI. As soon as information is to be shared, the risk of information theft and system attack is increased by the level of sharing. Every company is predisposed to take either a strong stand on the importance of security to the business or a more relaxed view. This inclination is driven by a number of factors, such as these:

- The nature of the business
- The potential for risk (is the company a likely target for attack?)
- The company's heritage and culture (open or closed community)

Determining the priority given to information security is necessary in order to judge the commitment of the company to follow through with security as a guiding principal in system development. It is also necessary in order to specify the scope and overall strength of the Information Security Enterprise Policy. Some companies will want to dictate a very detailed policy, where waivers will be required for any exceptions and where there is little room for interpretation. Other companies may prefer to dictate a more general policy where those delegated specific responsibilities will be required to determine the specific policy details for their organizations or area of responsibility.

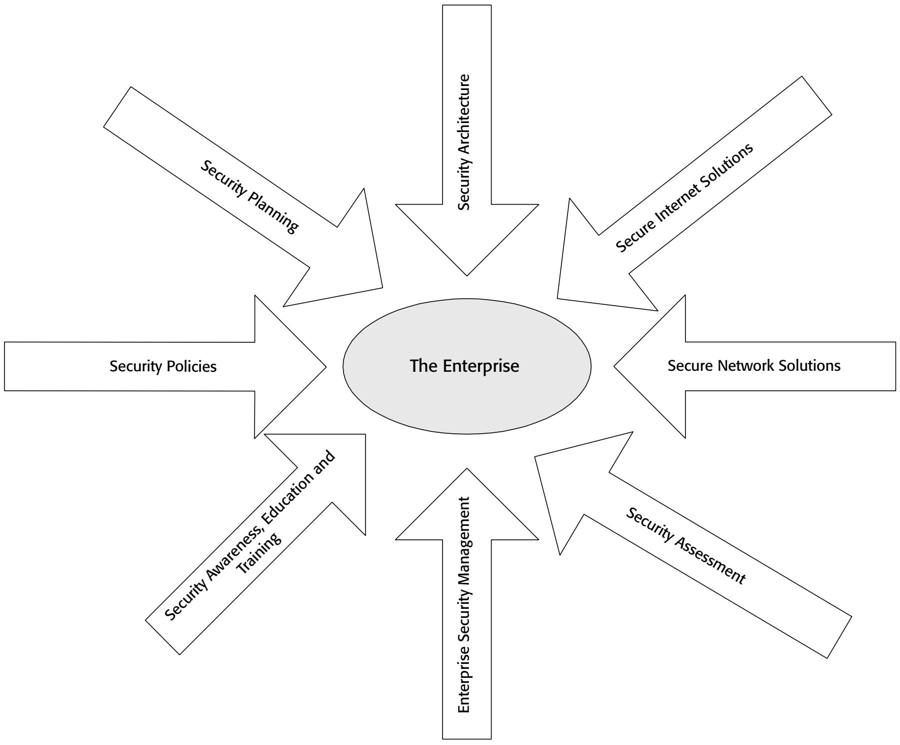


Figure 7.13 Security considerations for Web-enabled enterprise are a critical part of EAI.

Security is often disparate because of the nature of stovepipe legacy systems. This means that there is little consistency of security across the enterprise. Security is coming more to the corporate forefront with access to key business systems through Web technology, as shown in Figure 7.13. This magnifies the risks of the following:

- Release of personal or company proprietary or confidential information assets
- Unauthorized modification of information
- Legal liability
- Loss of customer or shareholder confidence
- Correctness of information
- Accountability for any modifications of information
- Timeliness and availability of information or service
- Loss of revenue
- Theft of service
- Opportunity for fraud

Even when it has been decided to provide enterprise-wide security there are options for what levels of security are provided. The least would be to provide a single secure interface to access existing systems and then rely on their existing security but suffer the lack of consistency. The most would be to provide a rich secure layer in front of all existing systems to ensure that security is all of the following:

- Complete with compliance across all operational systems
- Isolated to reduce access exposure and control damage in the event of a breach
- Centrally managed to ensure ease of control and reduced administration

Final Thoughts

It is critical to understand the business goals for any software to provide even a partial solution that will meet the business expectations. The business perspective should drive all IT activities to ensure the business goals are met. In addition to understanding the business drivers it is also critical to understand all aspects of the existing systems and applications, whether modern or legacy in nature. This will enable a delta to be established that identifies the additional software required to achieve the business goals. This software will fall into three types:

- Enhancements to existing software
- New technology components
- Connectors

The combination of these three types of software additions will enable new business functionality and information sharing without having to replace the legacy stovepipe systems that are the lifeblood of most corporations.

CHAPTER 8

Effective EAI Development: The SAIM Methodology

Key Facts

- A robust methodology is essential to effective use of EAI.
- A good methodology must ensure that the enterprise IT architecture satisfies business needs, describe how to manage the EAI activities, describe how to integrate legacy systems and packaged solutions, guide technology selection and standardization, and promote reuse.
- SAIM is based on the principles of aligning IT with the business strategy, building on a solid enterprise architecture, leveraging legacy and commercial software, and focusing on security.
- SAIM comprises five activity areas: Enterprise IT Strategy, Enterprise Architecture, Application Architecture, Component Development, and Application Integration and Deployment.
- Effective risk management is essential. Dealing with unprecedented technologies is almost always a major risk in EAI projects.
- To perform effective EAI, an IT organization should have an organizational focus for enterprise architecture and security.

We've discussed the business drivers for EAI, reviewed the key EAI middleware technologies of messaging, distributed objects, and transaction systems, and seen how these technologies can fit together in an enterprise architecture. The question remains, how do we accomplish all this? How do we decide on the technologies to use, how do we ensure that application development projects are using them appropriately, and how should we actually develop applications? The answer is to apply a sound methodology. Effective EAI requires more than just technology. Systematic and repeatable processes are also required, and those processes should be specifically designed for EAI.

In this chapter, we present the *Secure Application Integration Methodology* (SAIM), which we have developed at Concept Five Technologies. SAIM focuses on the following:

- Developing an EAI strategy that supports the enterprise's business goals
- Developing an enterprise architecture that supports the EAI strategy and establishes standards and guidelines for individual projects
- Developing applications that adhere to the EAI strategy and enterprise architecture

Before we get to the details of SAIM, we will discuss what an EAI methodology should do and why a methodology designed to facilitate EAI differs from a more conventional software development methodology. We will then present the principles on which SAIM has been designed and review each of its major activity areas. A detailed presentation of individual SAIM activities is beyond the scope of this book. Instead, we will present the objectives of each activity area and discuss some of the key activities that are conducted within each area.

Risk management is a key consideration in any engineering methodology, and SAIM is no exception. In SAIM, a special focus of consideration is the set of risks associated with unprecedented technology and processes. We will review this issue and discuss some mitigation techniques.

A final topic within the area of methodology is organizational issues. Although we do not think that SAIM demands a particular organizational structure, we do believe that EAI requires that certain areas of responsibility be clearly defined and allocated within the organization. We will address these issues, and we conclude the chapter with a summary.

Requirements for an Effective EAI Methodology

A methodology is to the development process as the system design is to the system itself. The methodology explains how the parts fit together to accomplish the goals of the whole. And like a system design, the methodology must respond to requirements. Specifically, a methodology for EAI should do the following:

- Ensure that the enterprise IT architecture and developed applications satisfy business needs

- Describe how to manage the EAI process
- Describe how to work with legacy systems and packaged solutions to integrate them
- Provide guidance on technology selection and standardization
- Ensure that the methodology promotes reuse

The methodology must ensure that the enterprise IT architecture and developed applications satisfy business needs. In Chapter 1, "The Business Drivers of EAI," we discussed some of the business needs that motivate an enterprise to undertake EAI, as well as some of the barriers that need to be overcome. The needs included taking advantage of the Web revolution, improving customer and supplier relationships, reducing time to market, leveraging existing legacy applications and packaged commercial applications, and so on. Some of the barriers included chaotic existing IT architectures, lack of appropriate skills in the IT organization, and lack of a comprehensive security architecture addressing the new threats that come along with EAI. An effective EAI methodology will facilitate achieving the business goals, while showing how to overcome the barriers.

methodology

A methodology defines a coordinated set of activities that are applicable to solving problems in a specified domain. SAIM describes activities related to the effective use of EAI.

An EAI methodology must confront the fundamental tension that is inherent in EAI. On one hand, we want to develop new applications rapidly, rolling out functionality in three to six months. On the other hand, we want a well-designed enterprise architecture that supports continual evolution because we know that EAI is doomed if each application project "does its own thing" with respect to middleware and legacy application interfaces. It's very hard, though, to develop a robust architecture in a short period of time.

An EAI methodology must describe how to work with legacy systems and packaged solutions to integrate them. This is perhaps the clearest difference between an EAI methodology and a conventional development methodology. Conventional methodologies, by and large, take a "green field" view of development: They assume everything is being made from scratch. EAI is fundamentally about leveraging legacy and packaged applications, so an EAI approach must address how to design the interfaces, how and when to wrap existing applications, and how to match existing applications with appropriate integration technologies.

Because middleware is so fundamental to EAI, an EAI methodology has to provide guidance on technology selection and standardization. Of course, it should suggest criteria for evaluating technologies, but it should also do more. It should provide a reference model or profile of relevant technologies and choices. It should map problem spaces into technologies, providing guidance about whether a given technology is appropriate for a certain problem class. And it should provide guidance on standardization: how should the enterprise decide what to standardize and what to leave to individual projects, whether to standardize on a particular vendor's offerings or rely on open-system standards, and how to manage and evolve the enterprise's standards.

A final requirement for an EAI methodology is to ensure that it promotes reuse. The methodology should establish a component-based architecture. It should ensure that component specifications and designs take into consideration the possible reuse of the component. Provision should be made for the necessary supporting facilities for reuse, such as an appropriately configured reuse library.

SAIM Principles

SAIM builds on a set of basic principles. We believe these principles are essential for an enterprise to succeed with EAI. The principles are as follows:

- Align IT with the enterprise business strategy
- Build on a solid enterprise architecture
- Leverage legacy and commercial software
- Focus on security

The first principle of SAIM is that EAI development must be aligned with the enterprise business strategy. It may seem surprising that this even needs to be stated, but the fact is that lack of alignment is often cited by CEOs as a key problem in their IT organizations. It is felt that IT organizations are more interested in technology for its own sake than for the sake of the business. SAIM recognizes that the enterprise's overall business strategy is the ultimate driver for IT development, and it provides explicit measures to ensure that the IT strategy is responsive to business goals.

The second SAIM principle is that mission-critical systems must be built on a sound foundation—a solid enterprise architecture. The overall architecture, more than the design of individual applications, determines how quickly applications can be built and how well the enterprise will be able to adapt to constantly changing requirements. Using clean, layered standard interfaces, as described in Chapter 7, "Enterprise Architecture," dramatically reduces complexity and lowers maintenance costs.

The focus of SAIM's approach to enterprise architecture is on business components and infrastructure. Reusable business components capture the core functionality of the enterprise's business processes. Because they are reusable across applications, new applications can be developed much more quickly—much of the most difficult functionality has already been developed. The core infrastructure services in an enterprise architecture operate like the telephone wiring in a building architecture: They provide the means of communication between diverse applications. By standardizing services across applications, the enterprise promotes interoperability, saves money, and cuts time to market.

The third key principle of SAIM is to leverage the use of an enterprise's existing legacy software and of packaged applications. Not having to write new code promotes rapid development. In addition, the legacy and commercial systems have already been largely debugged, so new applications will be more reliable. Most importantly, by using existing functionality for most of an application, developers are free to focus on unique capabilities that can create competitive advantage. Business components are the focal point for exploiting the enterprise's legacy systems and purchased application software.

The final key SAIM principle is that security is paramount. In SAIM, security is a constant concern and is thoroughly integrated into the development process. Comprehensive and cohesive security analysis, risk mitigation, and integration techniques and tools are applied throughout the project life cycle. The potential for loss is balanced against the cost of security countermeasures to achieve business goals within an acceptable level of risk.

SAIM Activity Areas

SAIM comprises five activity areas:

- Enterprise IT Strategy
- Enterprise Architecture
- Application Architecture
- Component Development
- Application Integration and Deployment

These five areas build logically on one another, but it is important not to think of them as sequential steps in a "waterfall" process, or even as steps in a single iteration of an iterative process. Instead, consider these five areas as concentric rings. As we move from the center outward, each ring builds on the previous one. Several individual projects or tasks in each area can proceed in parallel, moving outward through the rings. What is important is to maintain the linkages to ensure that the overall enterprise architecture remains consistent. The biggest management challenge in EAI is ensuring that the enterprise architecture remains under control while tasks in all five activity areas proceed concurrently.

Enterprise IT Strategy

The first of SAIM's activity areas is Enterprise IT Strategy. To develop an IT strategy, the relationship between business needs and IT activities must be established and endorsed by enterprise management. Strategy activities must have two main thrusts: first, to identify strategic IT initiatives and develop the business case for those initiatives; second, to ensure that the enterprise, and particularly the IT organization, is prepared to effectively perform EAI. Of course, not all strategic IT initiatives will necessarily be EAI initiatives, but for most enterprises, the vast majority of initiatives will involve some level of legacy application or packaged application integration.

The IT strategy should be subject to continual review, to ensure that alignment with business objectives is maintained and that IT projects conform properly to the strategy.

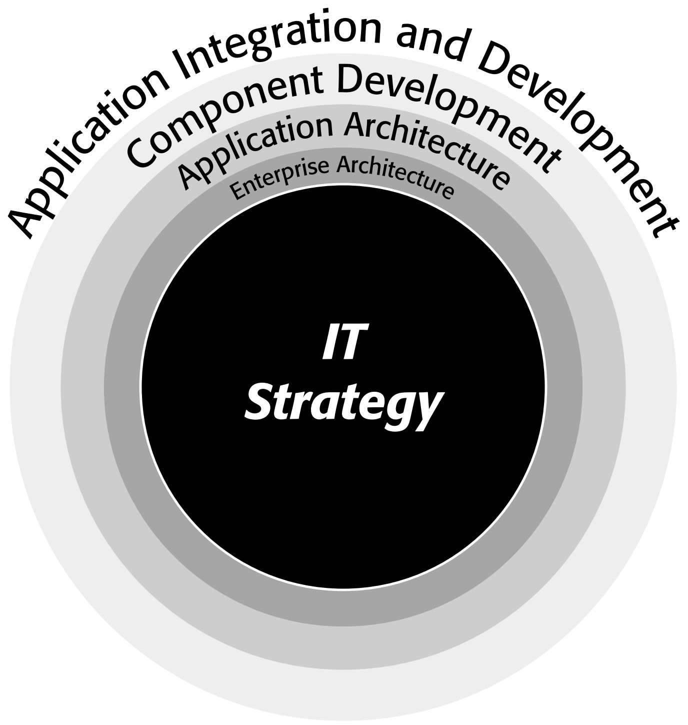


Figure 8.1 SAIM has five activity areas.

The major effort of developing the strategy will be a periodic effort that will usually be synchronized with budget planning cycles.

Identifying Strategic IT Initiatives

By strategic IT initiatives, we mean initiatives that clearly and directly support the overall business goals and strategies of the enterprise. The process of identifying these initiatives is one in which stakeholders throughout the enterprise should participate, not just IT personnel.

A variety of techniques can be used to define the enterprise IT strategy. We have found a strategy workshop to be particularly effective. A workshop gives stakeholders an opportunity to establish an overall framework for the IT strategy. It also creates a forum in which stakeholders can identify and resolve differences of opinion.

The IT strategy must be driven by the overall business strategy. It is critical that the business strategy be available, either in written form or embodied in the personal knowledge of the participants in the process. For example, the following factors need to be considered:

- What are the enterprise's competitive requirements?
- What is the basis of competition in the enterprise's market space?
- How is the enterprise positioning the business within that environment?
- How does the enterprise deliver value to its customers?
- What is most important to each of the enterprise's customer segments?
- What are the specific business goals and strategies of the enterprise?
- How does each goal and strategy address the competitive requirements?

It is also important to recognize that changes will not be limited to IT. Do not allow current business processes to limit the potential. Instead, identify ways in which business processes might change, enabled by IT changes, to provide new or improved services to users.

In analyzing the relationship between business strategy and IT strategy, you should perform both "bottom-up" and "top-down" analyses. The bottom-up analysis proceeds from an identification of customers' needs and the competitive requirements in various business units. The top-down analysis proceeds from the business goals and strategies of the business units. Together these analyses lead to the development of the Strategy versus Competitive Requirements matrix shown in Figure 8.2, which shows how requirements support business strategies and allows the requirements to be prioritized. IT initiatives can then be analyzed against requirements in a similar fashion, leading to an IT Initiatives versus Requirements matrix, shown in Figure 8.3.

After the IT strategies are prioritized, assess the impact of technological change in a brainstorming session involving IT personnel familiar with emerging technologies as well as those familiar with the enterprise's existing technology environment. The tech-

		Requirements				
		Requirement 1	Requirement 2	Requirement 3	Requirement 4	Requirement 5
Strategies	Strategy 1	•	•	❍	•	❍
	Strategy 2	•		•		•
	Strategy 3		•	❍	•	
	Strategy 4	•		•		
	Strategy 5		❍	•	❍	•

Key:
• Reguirement supports strategy
❍ Requirement partially supports strategy

Figure 8.2 The Strategy versus Competitive Requirements matrix shows how requirements support business strategies.

		IT Initiatives				
		Initiative 1	Initiative 2	Initiative 3	Initiative 4	Initiative 5
Requirements	Requirement 1	•	❍	•	•	
	Requirement 2	•		❍	•	•
	Reauirement 3		❍	❍	•	
	Requirement 4	•	•	•		
	Requirement 5		❍	❍		•

Key:
- • Initiative supports requirement
- ❍ Initiative partially supports requirement

Figure 8.3 The IT Initiatives versus Requirements matrix identifies the technology support required.

nology options that support the IT strategies should then be evaluated and ranked by their expected benefits, costs, risks, and ability to respond to changes in the business environment. This approach will identify the best technology enablers for the IT strategy as well as new and emerging technologies that make previously unacceptable high-value business strategies feasible. For example, a strategy of doing business over the Internet might have been rejected because it was believed that a customer could easily renege on an order. The evaluators of this strategy might have been unaware of digital signature technology with a nonrepudiation feature. Consideration of this technology could lead to a more favorable evaluation of this business strategy.

The steps that we have been discussing lead to a qualitative identification of IT initiatives. In some cases, this may be enough. Usually, however, a quantitative return on investment (ROI) analysis will be required. Details on performing such an analysis are beyond the scope of this book, but a few factors should be kept in mind. First, both costs and benefits of EAI projects are notoriously hard to estimate. This is particularly true in the case of revenues from Web-based electronic commerce initiatives. Second, it must be recognized that the costs of an EAI infrastructure must be allocated across current and future initiatives, not simply attributed to the first project that utilizes the infrastructure.

In analyzing ROI, here are some factors to keep in mind.

Revenue impacts

- ➤ Accelerated time to market for new products and services, due to shorter development times of EAI-based applications
- ➤ Attracting new customers, and increased revenue from existing customers, due to EAI-enabled improvements in service quality

Cost impacts

- Reductions in operations cost: reduction or elimination of manual integration activities, elimination of checking and reconciliation from current business procedures
- Marketing cost savings as a result of higher customer satisfaction and retention levels
- Reduced selling costs as a result of improved sales processes
- Reduced cost of working capital due to faster payment cycles
- Reduced inventory cost levels due to tighter supply-chain integration
- Reductions in non-IT staff due to improved processes
- Reductions in development time due to ease of integration with EAI
- Initial investment: EAI products, vendor consulting services, IT staff time to select, acquire, and install hardware platforms
- Ongoing costs: hardware and software maintenance for EAI software and hardware, IT staff time for support

Assessing Readiness for EAI

EAI requires different disciplines than development of traditional IT applications. It is a good idea to formally assess your enterprise's readiness to undertake large-scale EAI. The readiness assessment should address business, organizational, and technical issues. Business issues are those that relate to the enterprise's competitive position and the products and services it wishes to provide via EAI. An organization that has developed an IT strategy along the lines we discussed in the previous section will be in good shape with respect to business issues.

Organizational issues relate to the enterprise's management, organization, policies, procedures, and culture. The enterprise's management style can be an important determinant of readiness. For example, a firm in which line-of-business managers and IT personnel routinely collaborate on IT planning is more ready to take on EAI than one where IT planning is considered the sole responsibility of the IT staff. The enterprise's processes for budgeting IT projects and motivating IT staff are also relevant.

Organizational structures can influence readiness to undertake EAI in several ways. A key issue is whether there is an existing organizational structure, with well-defined roles and responsibilities, that is responsible for synchronizing and coordinating across IT projects. For example, is there a business steering committee, architecture or technology review board, or cross-business roles? Conversely, are project organizations highly independent and disinclined to share information?

In addition, because an EAI approach demands a high degree of integration between projects, the maturity of key software planning and engineering functions is critical to readiness. For example, you should examine whether smoothly operating quality assurance and configuration management functions are in place.

A final organizational issue with respect to EAI readiness is the organizational culture. Does the organization have a process for joint decision making and a good track record for follow-through? Is information freely shared in the interests of the enterprise, or is it routinely hoarded? Are disciplined processes encouraged, or are they regarded as limiting creativity? Does the organization demonstrate a commitment to growth and change by providing training as needed in new technologies and methods?

Technical issues relate to the IT infrastructure that is in place, as well as to staff expertise. Clearly, an enterprise will be more ready to adopt EAI if some of the necessary elements of the technical infrastructure are already in place and familiar to the IT staff. For example, a readiness assessment should address whether technologies like distributed objects or message brokers are already in use within the enterprise. Most importantly, EAI will depend on a highly componentized, standards-based architecture. It will be critical to evaluate the maturity of the architecture evolution and the existence of a well-defined target architecture. From a security perspective, the EAI readiness assessment should address the security architecture and components already in place.

Although an enterprise may not have addressed EAI as such, a strong existing methodology for systems development is a significant readiness indicator. An organization that is accustomed to following a well-defined methodology will have processes in place that will enable it to instill new EAI-oriented concepts and techniques in its work force.

Similarly, staff expertise is an important issue. The assessment should address the staff's general background (e.g., academic training, years of experience) as well as specific experience with EAI technologies. Does the organization have people who have worked both the IT and business sides of the enterprise? Does the organization have a high turn-over rate or do staff tend to come and move around in the organization, bringing knowledge and history with them? Does the organization have an adequate mix of technical and management skills?

The criteria against which the enterprise will be judged should be defined early in the assessment process. This will confirm the definition of the scope of the assessment and help avoid misunderstandings later. Figure 8.4 presents a list of possible criteria. Of course, these criteria must be tailored to the individual enterprise.

Enterprise Architecture

The second of SAIM's major activity areas is Enterprise Architecture. Managing the enterprise architecture is important in any IT context, but it is especially critical for EAI. The enterprise architecture identifies the components of the enterprise's systems, describes how those components relate to one another, and presents standards, including principles and constraints, by which the components can be refined into more detailed designs. The role of the enterprise architecture is to constrain the designs of constituent components in order to achieve overall architectural goals, such as standardization, simplification of interfaces, and the ability to accommodate change both in the business environment and in supporting technologies.

EAI ASSESSMENT CRITERIA

Management

- ☐ Is there a coherent IS strategy or master plan?
- ☐ Is there a plan to transition from stovepiped applications to an EAI framework?
- ☐ Do line-of-business and IS personnel collaborate on planning?
- ☐ Are projects synchronized to leverage common functions and data?
- ☐ Is the culture collaborative?
- ☐ Is development of reusable components, and reuse of these components rewarded.

Organizational

- ☐ Are roles, responsibilities, and relationships related to EAI clearly defined?
- ☐ Is responsibility to plan, coordinate, manage, and execute IS activities enterprise-wide allocated to a Program Management Office?
- ☐ Is responsibility for the EAI architecture clearly defined?
- ☐ Is there a smoothly functioning CM function in place?
- ☐ Is there a smoothly functioning QA function in place?

Methodology

- ☐ Is a repeatable life cycle methodology in place?
- ☐ Does the methodology address application integration specifically?
- ☐ Does the methodology provide for definition and control of an enterprise IS architecture?
- ☐ Does the methodology provide a means to integrate unprecedented technologies into the enterprise mainstream?
- ☐ Are metrics collected and analyzed?
- ☐ Are risks explicitly managed?

Infrastructure and Technical Expertise

Are EAI technologies currently in use?

☐ CORBA	☐ MQ Series	☐ Tuxedo
☐ DCOM	☐ Other messaging	☐ Other transaction monitors
☐ EJB		
☐ Translation/Conversion Services	☐ Data integration (Extract/Translate/Load tools)	

- ☐ Is an EAI training program in place?

Figure 8.4 EAI readiness assessment criteria addresses business, organizational and technical aspects of the enterprise.
Source: Concept Five Technologies, Inc.

For the enterprise architecture to be an effective basis for application development, the following issues must be adequately addressed:

- Is an enterprise security policy defined?
- Are high-level enterprise requirements for business functionality and infrastructure services understood?
- Have legacy and potential packaged applications been evaluated with respect to integration?
- Is the architecture defined and documented?

We will address these issues in the sections that follow.

Developing a Security Policy

The security policy establishes guiding security principles for the enterprise. Its purpose is to reduce risks to the enterprise's information assets. The policy will define the information security practices to which projects must adhere in order to protect the company's valuable information assets, its competitive position, and its reputation. Inadequate protection of information assets could result in loss of these assets, loss of competitive position, or loss of reputation.

The objective of an enterprise information security policy is to specify the high-level protection goals required to achieve the enterprise's business and IT strategy. The enterprise-level security goals must be sufficient to achieve the goals of the business, meet legal and regulatory requirements, and be acceptable to the risk tolerance level of the company's management. The information security policy must be persuasive and acceptable to the key stakeholders who will implement the business strategy and who will administer and manage the IT and business systems. The policy must also be understandable to the users of the system who must adhere to the security policy guidance and avoid attempts at circumvention.

The security policy in principle should drive all IT requirements and design. In practice, however, compromises can be made in defining the policy to reflect existing practice. The security policy depends strongly on the business and IT strategy initiatives.

All legal and regulatory requirements must be reflected in the policy, as well as corporate policies for proper business ethics and conduct. The policy must also identify the roles and responsibilities of individuals who deal with sensitive information. These roles will likely be tightly correlated to the organizational structure and division of responsibilities generally found in a business.

Critical information resources must be identified. Critical resources are not always data. The business logic embedded in applications may be just as important as the data managed by those applications.

Based on the risks to the enterprise, appropriate protection goals must be identified. These protection goals should cover the information asset throughout its life cycle. The protection goals should target all applicable forms of threat (e.g., eavesdropping, spoofing, masquerading, denial-of-service attacks, data corruption, destruction, and theft of information or service).

The following steps are required to develop a security policy:

1. Identify legal, regulatory, and business protection requirements that apply to the enterprise. Be sure to obtain guidance from appropriate experts concerning the firm's legal obligations to secure confidential information.
2. Define roles and responsibilities. These may include the roles of IT technical staff, as well as management and operational personnel.
3. Identify critical information resources. Information resources should be placed in categories based on their sensitivity, for example, confidential personal information, intellectual property, and sensitive operations data.
4. Identify protection goals for critical information resources. These goals should ensure that the level of protection is commensurate with the sensitivity of the information and the risks of compromise. Identify *who* should be allowed access to various types of information and *what* they are allowed to do with it (i.e., create, update, destroy, copy).
5. Determine the applicability of the policy. Is the policy applicable to all users of information (e.g., employees, contractors, customers, etc.)? Are there special considerations concerning from where information is accessed? Some enterprises differentiate between access from a PC directly connected to the enterprise's LAN and access via dial-up, on the grounds that dial-up access could be compromised more easily.
6. Define compliance requirements. Describe acceptable and unacceptable use of information, and determine the consequences of unacceptable use.

Analyzing Business Component Requirements

Requirements analysis at the enterprise architecture level has two objectives:

- To determine the high-level business functionality to be shared across applications
- To determine the infrastructure services that should be available to applications, based on currently identified strategic initiatives

The business functions suggest candidates for reusable business components. At the enterprise level it is not necessary to identify detailed operations on business components, or even detailed attributes and relationships of business entities. It is sufficient to identify the fundamental entities that will be reused across applications.

The implications of strategic business and IT initiatives on business processes must be assessed. High-level business process models should be developed, particularly for new functions that will be introduced by the initiatives (e.g., customer self-service over the Internet). Extremely detailed models are unnecessary for this activity because the objective is to understand how the business processes will affect the enterprise architecture, not to define detailed requirements for implementation. More detailed models can be created if needed as part of the application development process. The business process analysis should be refined via interactions with line-of-business staff.

Based on the process models, high-level business components can be defined. The primary basis for identifying business components should be the fundamental business entities of the enterprise. These can be traced from basic business functions (e.g., for a manufacturing enterprise: R&D, Production, Marketing, Sales, Distribution, Service, Finance and Accounting, Personnel). Closely related entities can be grouped to identify components. Entities such as *person* and *organization* are cross-functional; they can identified as such and not associated with a specific function.

The high-level business components should be grouped into clusters with the business functions that create them, to form the basis for defining components. The requirements analysis team should write a brief description that summarizes the function of each cluster. Before being finalized, these descriptions should be analyzed for consistency and cohesion.

Analyzing Infrastructure Requirements

The purpose of analyzing infrastructure requirements at the enterprise architecture level is to identify the characteristics that the enterprise architecture must have in order to provide acceptable levels of service. Enterprise architecture requirements must not only address the current needs of the enterprise, but specify the characteristics that the architecture must possess in order to ensure that the enterprise's IT systems can support its changing needs. This requires a different approach from traditional requirements analysis, which focuses on well-specified and bounded needs. On one hand, you must address the question, "How can we position the enterprise's IT infrastructure so that it can accommodate requirements that have not yet been conceived?" On the other hand, few enterprises have the resources to build capacity into their IT infrastructure for all conceivable eventualities. The enterprise architecture requirements must strike a balance between wasteful "gold plating" and an inflexible architecture that is unable to accommodate change.

To begin to determine infrastructure requirements, planned applications should be grouped into categories based on the following characteristics:

User community. Customers, employees of partner or allied enterprises (e.g., supply-chain businesses), and in-house employees.

Scale of use. Handful of users, a hundred, thousands, tens of thousands.

Criticality. Mission-critical, non-critical.

Within each category, analyze applications with respect to the following:

- Security requirements
- Maintainability and adaptability
- Reliability and availability

- Performance
- Scalability
- Integrity
- Manageability
- Usability
- Recoverability

By grouping applications using these criteria, the necessary infrastructure services can be identified. It is important to be realistic in this assessment. Ideally, every application would have high integrity, have outstanding performance, be fully recoverable in case of failure, and so on. But each of these characteristics comes at a cost, both in terms of development time and additional infrastructure products. Service requirements should reflect the true needs of the applications.

Assessing Legacy and Packaged Applications

Assessing legacy and packaged applications to determine how they fit into the overall design is a critical aspect of defining the enterprise architecture. The approach to the two categories is essentially the same. Evaluation of packaged applications that are not currently installed should be based on analysis of detailed documentation (and preferably hands-on evaluation), not on sales literature. In the balance of this section, we will refer to both categories as "legacy systems," except where it is necessary to make a distinction. SAIM focuses on two major dimensions of this problem, assessing how the legacy systems fit functionally into the enterprise's needs and assessing the integration facilities afforded by the legacy systems.

An asset that should not be ignored in assessing legacy systems is the material that may have been developed as part of the enterprise's Year 2000 efforts. Y2K has generated significant documentation on many applications that previously had been undocumented or poorly documented. Moreover, documentation resulting from Y2K efforts should be reasonably up to date. Analysts assessing an enterprise's legacy systems should always review the results of any Y2K efforts on the legacy systems.

Analysis of legacy applications begins by developing a mapping between the legacy systems and the business component framework. It is necessary to perform a gap analysis; that is, to identify capabilities and functions that either do not exist or are inadequate in the legacy systems. Two possible relationships could exist between the legacy systems and the business component framework.

Legacy system completely subsumes business component. In this case, analysis reveals that the legacy system provides all functionality necessary to support a business component. Alternatively, the legacy system can be modified to provide additional functionality with relatively little effort. Further, the legacy functionality is reasonably accessible; that is, there exists an interface, such as a sequence of screens, a function initiated by a message, or a CICS transaction that implements the behaviors associated with the business component. The legacy system should probably assume responsibility for the component. A wrapper can be developed to expose an appropriate component-based interface to newly developed applications.

Legacy system partially supports business component. In this case, the legacy provides some but not all of the functionality required for a business component, or several legacy applications support the component. Use of a container to manage links to several legacy systems may be considered. Alternatively, a complete new implementation of the component may be considered, in which case the legacy systems (assuming they are to continue to support the component) will need to be modified to interface with the new component rather than their existing data stores.

If wrappers are to be developed, they should conform to the boundaries in the framework. For example, a given legacy system may support both application-specific functionality and business domain functionality. Or it may support several distinct components at the business domain level. In such cases, several distinct wrappers should be developed to maintain the framework structure.

Frequently a strategy of *orderly migration* or incremental replacement of legacy systems is adopted. In this strategy, the legacy system is first wrapped, and new applications are developed that depend on the wrapper. Then pieces of the legacy system are replaced in an incremental fashion "behind the wrapper," while maintaining the interface for the new applications. When the migration has been completed, the legacy system will have been totally replaced by new component-based functionality.

Packaged applications, such as commercial ERP packages and customer relationship management (CRM) packages, are analyzed in much the same way as internally developed legacy applications. In the case of packaged legacy systems, however, there may be no opportunity to modify or enhance the legacy system.

In addition to analyzing the functionality provided by legacy systems, it is necessary to consider how easy it is to get at that functionality. Does the legacy application provide an API? For packaged applications, is the API accessible via standard programming languages or via the vendor's proprietary language? Does the application support messaging, file-based input/output, or a database? For a packaged application providing integration via a database, it is important to determine under what circumstances the vendor will support customer access to database tables and whether the vendor is willing to commit to keeping the table definitions stable. In the worst case, legacy systems can be integrated via their user interfaces, using screen-scraping technology.

Legacy systems should be investigated to determine whether they reflect a one-tier, two-tier, or three-tier model. Older CICS applications tend to be monolithic, or one-tier; newer ones often have a three-tier structure, with accessible CICS interfaces between the tiers, even though all three tiers run on the same platform. If the design is multitier, are the internal interfaces well documented and supported?

Security capabilities of legacy systems should also be investigated. Does the application implement its own security scheme? If so, how difficult will it be to replace the existing security code with calls on standard authentication, authorization, and auditing services?

Specifying the Enterprise IT Architecture

The previous steps will have developed the information necessary to define the enterprise architecture. It is necessary to capture that information in an appropriate form for it to be most useful. The enterprise architecture must be defined and documented in sufficient detail so that development of applications can proceed. The specification must be detailed enough that applications can be developed concurrently and independently while still being able to interoperate efficiently.

An enterprise architecture is a complex entity. The temptation must be avoided to reduce the architecture to a single piece of paper or single block diagram. We have found that in order to completely specify the enterprise architecture, it is necessary to present a number of independent "views." These views can be used to organize an architecture specification. There are 10 views:

Component view. Decomposes the architecture into components, which are then further decomposed into subcomponents, and so on. It is the basis for managing the development and maintenance of the architecture.

Entity/object view. Describes the business entities that are made available through reusable components in the business domain layer of the architecture.

User view. Describes the user interface services that are available.

Data view. Describes the databases maintained in the architecture.

Legacy view. Describes how legacy (and packaged) applications are integrated into the architecture.

Security view. Describes the security services that are available, and provides guidance for how they should be used in applications.

Physical view. Describes how the functionality and data are mapped onto physical devices.

System management view. Describes how the various elements of the architecture are managed.

Disaster recovery view. Describes how the architecture is transformed in the event of possible disasters, in order to maintain essential business capabilities.

Development view. Describes how applications and components are developed.

Application Architecture

The objective of the SAIM Application Architecture activities is to establish the foundation for the development of a single application. The application architecture ensures that the application corresponds to business requirements, that it provides an appropriate quality of service and security, that it meets time-to-market needs, that it can evolve to deal with changes in the business and technical environments, and that its business domain components will be reusable in future applications.

We view an *application* as a software entity that focuses on providing a cohesive set of capabilities to end users. Examples of applications might be a line-of-business system (e.g., casualty insurance) or a front-end system that supports multiple back-end systems (e.g., a customer service system). In the Application Architecture activities, the application is analyzed as a whole, and high-level design is taken to the point where requirements are allocated to components, which can then be developed concurrently and independently.

Application Requirements

We recommend developing application-level requirements in two ways, using business process models and use cases. Business process models are used when a new application will have a significant impact on how business is conducted. Use cases are used to capture technical requirements for IT components.

It is essential when developing a business process model to capture information about how end users will work in the future environment using the strategic application. Because the future work environment may be different from the current environment, the user community should make available a set of end users who have the authority to speak for the entire user community. You may also want to include business representatives and stakeholders who participated in the Enterprise Business Modeling activity. Likewise, new business participants will contribute by bringing a new perspective.

Use cases describe the application and contain scenarios in which the application will be used. They are the starting points for identifying functional business requirements. Use cases can also influence the architecture design by identifying scenarios that require transactional capabilities or error recovery capabilities. The primary objective of a use case analysis in the context of an application architecture is to capture the expectations for the application's functionality. This is achieved by understanding how the system will be used by end users, or if there is no human end user, how the system will respond to external stimuli (e.g., events, messages, or data from external systems). A secondary objective is to provide a mechanism for ensuring that nonfunctional requirements are met. By formulating use cases that focus on infrastructure-related concerns, nonfunctional requirements are given priority instead of being buried within functional use cases. Because these use cases can affect a significant number of system components, they are usually implemented before use cases that primarily support business domain behaviors. It is also important to ensure that use cases explicitly deal with security policies. For example, systems that involve user interaction should have a "login" use case.

use case

A use case is a description of a scenario in which an application will be used.

Use case analyses create a bridge between nontechnical users and technical architects, developers, testers, and system maintenance engineers. This activity provides a stepping stone for translating business requirements into a working system.

Analyzing Application Security Requirements

An application security requirements analysis evaluates system threats and security risks associated with a proposed application. The application security analysis will capture potential security vulnerabilities in the application. It covers the range of relevant security goals defined in the enterprise security policy. Significant emphasis should be directed at analyzing any weaknesses resulting from combining components into a system of systems, where each component individually may appear to provide adequate security, but the integrated application is vulnerable. An additional goal is to reduce the number of security-relevant components overall to minimize the amount of security critical code that must be developed, tested, and maintained.

Threats are any circumstances with the potential to cause harm to a system in the form of destruction, disclosure, modification of data, software or hardware, or denial of service. They can include human attacks, inadvertent human errors, software or hardware flaws that lead to security problems, and natural disasters. Vulnerabilities can be any weakness in system security procedures, system design, implementation, or internal controls, which could be exploited to violate system security policy. Threat agents typically take advantage of system vulnerabilities to compromise security.

Effective security requires managing the security-relevant components in the setup and operation of the application. This means managing the strength of security mechanisms over time and ensuring ease of use so the administrators can make changes to the security of the application easily.

Using proprietary security attempts to achieve "security by obscurity" creates a false sense of security. Because the mechanism is not well known, it is thought not to be vulnerable to attack. This is a bad practice because security components do not gain assurance except through extensive use (this is why we trust operating systems today), and it does not support an open evolvable system as new security technology advances are available. The analysis should look for the use of standards-based products, architecture, and services with published mechanisms and APIs.

Security must be made easy to use, or users will attempt to circumvent the safeguards or unknowingly misuse the security mechanisms. A common example is use of passwords for authentication. If a user is expected to remember too many passwords, the user will likely write them down or use the same password for several systems, thus making all systems vulnerable should the password be compromised. Providing a unitary login capability or a smartcard-based solution alleviates this type of problem.

Developing the Application Architecture

The core of the application architecture is a high-level allocation of application requirements to components. The components act as black boxes, potentially with large numbers of elements. The set of enterprise architecture components used in a particular application must be called out in the application architecture. These will include both application-specific components and reusable components at layers below the application-specific layer. The application architecture must detail those components at a sufficient level of specificity so that they can be designed and developed in subsequent steps.

Components, particularly at the business domain layer, can be shared by several applications. They are usually developed, however, with a particular application development project. The development project must take into account that the components are intended to be shared by other applications in the future, and it must adhere to appropriate design principles for reusable components. In addition, certain component requirements must be laid down from the component's initial development because they are very difficult to address once the component has been initially implemented. For example, if a component may take part in transactions, it is necessary that appropriate transactional protocols be adhered to from the very beginning. It can also be the case that an existing component needs to be modified to meet the needs of a new application. Care must be taken that the existing applications that depend on the component are not adversely affected by the change.

In the 1990s, the computer science discipline codified a set of approaches and techniques as *patterns*. The intent of the pattern movement is to develop a common vocabulary and set of approaches to design problems that are encountered in a broad range of applications. Initial pattern work has primarily focused on the design of specific components rather than the application level, but a set of architectural-level patterns also exists in the industry. A range of architectural patterns supports systems, from transaction processing to analytic to process control.

pattern

A pattern defines an approach to the solution to a broad class of related problems. Patterns capture what is common to the solutions in a form that is useful for solving newly arising problems within the class.

Patterns describe common solution features while suppressing details of solutions that are specific to a particular problem. Business domain components required by the application may or may not have been previously developed. For newly defined business domain components, the architect should carefully review the application's specifications as they relate to new components and ensure that other applications that are expected to use those components are represented in their specification. For existing business domain components, care must be taken to ensure that they are properly specified and supported for the application's use. Existing interfaces may require some alteration to support the new application.

The application architecture should identify the services the application requires that are provided by the infrastructure layer of the enterprise architecture. The enterprise architecture will have specified particular standard approaches for particular infrastructure services. The application architecture must follow those approaches, for example, for distribution services. If the enterprise architecture specifies a particular approach to distribution services, for example, CORBA, the application architecture must follow that unless there are specific reasons that the application cannot.

Selecting Commercial Products

Commercial products may be included in the application architecture at any level. Assessment and selection of products may take place at a variety of times during the development of an application, but selection is best considered as part of the Application Architecture activity because the selected products will likely be introduced in conjunction with the development of a new application.

The commercial market is first explored in order to identify all possible vendor products that are available. Very few explicit requirements are actually needed at this point, just enough to identify the type of product being sought. The intent is to develop as complete a list of products as possible so that viable candidates are not overlooked. An initial set of high-level evaluation criteria is developed based on the system requirements. Once the list of potential products has been assembled, an initial screening is performed against the major or most critical criteria—these are the requirements drivers that the product must satisfy in order to be a viable candidate. The investigation of the products against these drivers can usually be done by an examination of vendor literature—that is, commonly available sales and marketing materials—to weed out those products that are clearly unsuitable candidates.

Viable candidates are next evaluated more thoroughly. At this point it is necessary to have a fairly good understanding of the application requirements. The requirements are translated into specific criteria against which each candidate will be assessed, and a weighting/scoring scheme is developed. The next step is to more thoroughly understand the behaviors and technical characteristics of the products in order to be able to select a winner. This is best done through hands-on investigation accompanied by a review of the technical and user manuals and discussions with vendor staff. Sometimes there is not a clear winner, such as when there are several suitable products; in this case, the deeper investigation will help reveal product differentiators that can be used in trade-off analyses.

Component Development

A component is a software entity that provides a cohesive set of functional capabilities through a specified interface. In SAIM, we are interested in extremely coarse-grained components, which are significant systems in their own right. For example, an enterprise's entire financial information system could be considered a component. An application potentially consists of many components, and a component that is part of the business domain or infrastructure layer of the architecture may support many applications; components in the application-specific layer of the architecture by definition support one application.

component

A component is a software entity that provides a cohesive set of functional capabilities through a specified interface.

Component development activities in SAIM are not that dissimilar from more traditional development methodologies. Our approach does have some unusual aspects, however, which we'll highlight here.

There are a number of distinct types of components, each with its own development strategy. They include the following:

Custom components. These components are developed from scratch, at least initially. Because components can be reused across applications, any given component in an application development project may be built from scratch, be a modification of an existing component, or be reused without change.

Wrapped legacy application components. These components are built on top of in-house legacy applications. The legacy may or may not have to be modified. As with custom components, the wrapper component may be new in a given application development or a (possibly modified) existing wrapper.

Wrapped packaged applications. These components are similar in most respects to wrapped legacy applications, but it is usually infeasible to modify the underlying legacy.

Wrapped databases. These components are wrapped databases which are a special category that serves database information through a distributed object, messaging, or transactional interface. The wrapper is intended to relieve the client programmer of the burden of knowing the details of the database structure or how to access the database directly.

Infrastructure components. These are usually off-the-shelf components such as ORBs and databases.

Because components are usually intended to be reusable, the specification for a component should address the long-term requirements, not just those of the current application.

Application Integration and Deployment

The final SAIM activity area is Application Integration and Deployment. The most distinctive feature of the SAIM approach in this area is the emphasis on iterative deployment and piloting. Hence an important activity is pilot evaluation. Because of SAIM's continual emphasis on security, we also include conducting security penetration tests as part of this area. The Application Integration and Deployment area also includes a number of other activities, such as integration testing and deployment planning, but those activities are handled fairly conventionally, and they will not be discussed here.

Evaluating a Pilot

The objective of a pilot assessment is usually to determine whether the application meets its overall goals. For example, if the goal of the application was to provide a new or improved service to customers, the pilot evaluation might determine whether customers find the service useful. If the objective was to enable employees to access and find job-critical information more easily and quickly, the pilot might investigate how easily employees can find critical information. A pilot assessment will also allow IT or operational personnel to identify flaws or deficiencies and to recommend improvements.

The activity may require considerable coordination with, and support from, the using organization in order to carry out the evaluation, such as conducting interviews or collecting and monitoring system data. Overuse of pilot evaluations can cause loss of focus and resource drain. Evaluating a myriad of minor and inconsequential things about the application generally has little benefit.

It is essential that an evaluation plan be written that articulates the exact purpose of the evaluation and that describes the evaluation methodology. For example, "to determine if this is a good system" is not a well-defined evaluation objective. On the other hand, "to determine the percentage and types of employees who regularly use the system to quickly find information" is much better defined.

Performing Security Penetration Tests

The security penetration testing activity demonstrates that the security services and mechanisms are complete, correct, and consistent with the approved design. It also validates that the application satisfies critical security policies and requirements. Security penetration tests are intended to expose security flaws.

We distinguish between *security probing* and *penetration testing*. Security probing identifies vulnerabilities in specific system components. Penetration testing identifies and confirms system vulnerabilities. Probing can be performed as part of penetration testing.

security probing

Security probing tests specific system components for security flaws.

Penetration tests attempt to circumvent the application's security features. The tests seek to access unauthorized component data or functions, to observe, extract, modify, or add unauthorized data or functions, or to deny component services to other users.

penetration testing

Security penetration testing tests an architecture as a whole by attempting to defeat its security features. Penetration testing employs the same tools and techniques that hackers use.

At a minimum, the penetration test should cover these areas:

- All perimeter access points should be checked to ensure authentication and access controls function properly.
- All network communications paths should be tested for vulnerability to disruption, corruption, or eavesdropping.
- All data entry, for example, forms and fields, should be tested for the ability to withstand bad entries or attempted form changes.
- All security mechanisms should be tested to ensure they support the security service described in the security policy.
- Common flaws should be checked, such as failure to remove default login accounts or test accounts.
- Known bugs in vendor products should be checked to ensure vendor patches have been applied.
- Session hijacking should be attempted for any session management techniques used, such as cookies or URL-encoded session IDs.

Risk Management and Unprecedented Technology

The rapidity with which new technologies emerge prevents an organization from getting up to speed on one technology before something else emerges. Yet the inability to incorporate these technologies into a new application can mean the difference between simply keeping up with the pack and competitive advantage. Thus EAI-based solutions invariably contain unprecedented development, involving risks that require specialized mitigation strategies.

SAIM's approach to risk management focuses on the unprecedented. Unprecedented elements that must be addressed include the following:

- New classes of applications (e.g., customer self-service Web site)
- New business domain components. For many organizations embarking on component-based architectures for the first time, the entire concept of a business domain architectural layer may be unprecedented. In this case, entirely new processes to

ensure both the reusability and the actual reuse of software may need to be implemented.

- New infrastructure services. Again, entire components may be unprecedented.
- Enterprise architecture guidelines. Such guidelines may be unprecedented. That is, an enterprise may have had some experience with all the application and technical elements but have not attempted to standardize their use across the entire organization.

Because the use of unprecedented technology inevitably incurs some learning costs, it is important to consider whether the unprecedented elements are of high value. Consideration should also be given to functional or technological replacements; for example, replacing a legacy system with a new custom-developed component, or replacing a given element with an equivalent element from another vendor.

Situations that require special concern and management attention include the following:

- Business applications with extremely tight or near-term time-to-market constraints that depend on unprecedented elements.
- Mission-critical applications that depend significantly on unprecedented technology. In general, experience should be gained with new technologies on applications that are not deemed mission-critical.
- Any applications that depend entirely on unprecedented elements (i.e., there is no identified contingency plan).
- Elements that are unprecedented not only within the enterprise, but in the marketplace as whole (i.e., "bleeding edge" technologies).

Each of these cases represents significant risk.

Unprecedented elements are typically managed using prototypes. Small "run-ahead teams" that specialize in rapid prototype development to gain knowledge of the process behaviors can be brought in to attack unprecedented tasks. Through rapid iteration, the run-ahead team converges on stable, robust solutions on compressed time scales. This method contains costs, ensures schedules, and provides early notification of problems that need special attention.

Organizational Considerations

EAI and SAIM have many impacts on the IT organization. In this section, we briefly address two of the more important impacts: the role of the enterprise architect and associated roles and an Information Security Steering Committee.

Enterprise Architecture Organization

SAIM is an architecture-driven methodology. To effectively utilize SAIM and, we would argue, to effectively apply EAI, IT managers must develop a unified approach to managing the enterprise's architecture. Many organizations do not have an enterprise architect or any organizational focal point responsible for enterprise architecture.

A number of organizational patterns are possible to ensure that there is an appropriate architectural focus within the enterprise. A successful model includes the following elements:

- An Enterprise Architect, who reports to the CIO.
- An Enterprise Architecture Steering Committee, which is responsible for setting architecture policy and making major architecture decisions. This committee should include the Enterprise Architect (although not necessarily as the chair) and have representation both from the IT organization (development, maintenance, and operations groups) and from the business units that IT supports.
- A small Enterprise Architecture staff, who are responsible for maintaining the enterprise architecture specification, for analyzing future requirements and associated architectural changes, and for evaluating new technologies. The staff should not be a permanent elite; instead personnel from various parts of the IT organization should rotate through its positions. In addition, a significant portion of staff members' time should be spent working on application development projects. Working on projects serves two functions: It helps project personnel understand and apply architectural rules and guidelines, and it gives the architecture staff a realistic sense of the impact of architecture standards on the application projects.

Information Security Steering Committee

Security considerations require a consistent management approach in order to ensure that the security policies are appropriately interpreted and applied. We have found that an Information Security Steering Committee (ISSC) represents a good approach to managing security issues. The ISSC will be responsible for defining the enterprise's information security policy and for ensuring consistency with the business goals. The membership of the ISSC should include executive-level management in order to ensure that the committee's recommendations have sufficient clout.

An ISSC may fulfill its responsibilities through many types of activities. Some examples are these:

- Reviewing the enterprise's security policy in light of emerging threats
- Investigating any breaches of security that may occur within the enterprise, and adopting rules to ensure that they are not repeated
- Reviewing security characteristics of critical applications, both at the stage of requirements definition and before the applications are placed in production
- Sponsoring the development of security education and training programs for the enterprise

Final Thoughts

In this book, we have presented EAI technologies and described how you can apply them effectively in your enterprise. In this chapter, we have described, at a high level, a methodological approach that we have developed and applied to bring the benefits of EAI to our clients.

We are nearly at the end of our journey. In the next chapter, we will conclude with some thoughts on how you can develop an EAI-oriented culture within your organization.

CHAPTER 9

Building Future EAI Business Solutions

Key Facts

- EAI is an approach to developing information systems that is based on the use of products that prepackage integration and related services.
- Integration is a broad topic that spans a wide spectrum of activities including data consistency, multistep process, and component integration.
- Organizations should move through four stages to effectively apply EAI.
- The four stages are getting started, EAI architecture, EAI organization, and EAI enterprise.
- Getting started tests the use of EAI through a pilot application.
- The EAI architecture stage begins to broadly apply EAI to other projects.
- The EAI organization stage makes EAI a part of the organization's process of implementing information systems.
- The EAI enterprise stage puts all of the infrastructure in place to continuously apply and improve the organization's EAI architecture.
- The EAI architecture should be planned and evolved in incremental steps.
- EAI is a way to structure the architecture of your systems.

In this book you have learned about the critical role that EAI will play in the enterprise. EAI is neither a trend nor a product but an approach to developing information systems. This approach is a combination of architecture, technology, and methodology. It is differentiated from prior approaches to developing information systems by the availability of software products that enable good integration and contain prepackaged integrations. It is necessary because of the significant investments made in existing systems that are difficult or expensive to replace but must be used in new configurations within the organization.

The types of problems that require integration cover a broad spectrum. Integration has moved from simpler presentation and data to functional integration. Within functional integration we have identified three general patterns of integration:

- Data consistency
- Multistep process
- Component integration

They differ in the technology and the process required to develop a solution.

There are four different basic building blocks for EAI architecture: communications model, method of integration, middleware, and services. The core element, however, is the middleware technology. MOM, DOT, and TPMs are the source of middleware for EAI architecture. They are combined with the other building blocks to establish an EAI architecture.

The technology by itself, though, is not enough. EAI architecture requires an approach to structure an application to ensure the proper application of the architecture. This also must be coupled with methodology to form the total EAI solution for the enterprise.

This chapter ties together all of the material discussed in this book in a series of steps that an organization can take to apply EAI technology effectively. It provides the reader with an approach to becoming an EAI-enabled enterprise.

Stages of EAI Evolution

EAI technology has the potential to improve the development time and efficiency of an enterprise's systems dramatically. Like any other technology, it can be abused, mishandled, or misapplied. The application of EAI in an enterprise can exhibit any and all of these attributes. Therefore, it is important to apply it in a manner that leads to a successful result and minimizes all of the potential negative aspects. We believe that, to best meet this need, an enterprise should go through a process of bringing EAI into the organization.

In our experience an enterprise should go through four stages to become effectively EAI-enabled:

1. **Getting started stage.** Organizations must plan, learn, and architect with the intent of achieving early success.

2. **EAI architecture stage.** Organizations must create an architecture that can be understood by designers and developers and that can promote future reuse.
3. **EAI organization stage.** Organizations must apply the results from the first two stages across the organization and expand EAI services.
4. **EAI enterprise stage.** The organization has mastered EAI and is effectively employing and measuring the effectiveness of its EAI architecture.

These stages reflect the lessons we learned in working with this technology across a broad spectrum of problems.

Getting Started with EAI

The goal of the first stage of EAI is to begin down the path of becoming EAI-enabled. The best way to do this is to select an appropriate application that needs to be developed and apply EAI technology and methodology. The organization should use this pilot application to develop experience and to demonstrate success in an application while at the same time creating an EAI architecture that can be broadly employed.

This stage should be initiated by thought leaders in the organization who have the support of the CIO or the management one level below the CIO. It should be done in conjunction with a project manager who is responsible for the pilot application development. It should be scoped to an activity that takes between three and six months.

Although the SAIM methodology described in Chapter 8, "Effective EAI Development: The SAIM Methodology," may seem to suggest that the strategy and architecture be done before embarking on any development, this is not our intention. In fact, the methodology encourages the parallel execution of the strategy and development. It is important, though, to have some elements of strategy and architecture in place prior to developing a pilot because these elements form the basis for certain decisions in the pilot. These key strategy and architecture elements can be done quickly with the recognition that the results from the pilot may necessitate a change in the architecture. This initial version is referred to as the strawman strategy and architecture.

strawman strategy and architecture

The strawman strategy and architecture are versions of the strategy and architecture that have been rapidly developed for the purpose of fostering discussion and comment. They are used to allow a broader audience to participate in the definition and should create a positive debate.

An important outcome of the pilot is to guide the strategy and architecture. This stage might also be called the "learning as you go" stage. At this point, the organization will have little experience with the technology or its application, and the collection of results is important. The strawman strategy and architecture should be compared to the results and lessons learned from the pilot. The differences between the strawman and the actual implementation should be noted and considered in the next version.

The pilot itself should be a functioning application where the use of EAI is central to the application's requirements. It is important to note at this stage that the use of EAI technology is a low risk, and it should not add significant costs, if any, to the pilot. Because the integration is required, the specific technology used might change, but the work was required independent of EAI. A key to success is training and expert assistance because the misapplication of technology is often the reason for failure.

Finally, out of this stage a roadmap should be developed that sets the stage for the organization to move from the "getting started" plateau to the next stage. The roadmap should focus on getting the architecture right, taking it out to a few additional applications, and developing a plan for rolling it out across the organization. This should not be taken lightly nor at a pace that the organization cannot accept.

Once these things are accomplished the enterprise has reached the first plateau and is ready to move on to the next stage. At this point the EAI architecture, methodology, and technology can be applied by other applications with little risk and great value.

The EAI Architecture

The goal of the second stage is to solidify the architecture and begin to apply it to new applications and enhancements to old applications. The reason the EAI architecture is required is simple: Without the architecture there will be no reuse, and organizations will continually duplicate interfaces even if they use only a single technology. A key to the architecture is to provide visible interfaces and ensure that they are considered in the course of application design. The cost of this duplication will have a tremendous impact on an enterprise's resources. Figure 9.1 shows this impact.

The team that has responsibility for the EAI architecture in conjunction with other project managers in the organization should initiate this stage. It should occur over a 6- to 12-month period.

Figure 9.1 shows that if an enterprise has not organized an EAI architecture to ensure interface reuse, then the result is an exponential growth rate in the cost of integration. This means that for every application that is added you will need to add interfaces that total one less than the total number of applications. A good EAI architecture should require only one additional interface. In the beginning, it will be argued by many in the enterprise that custom building the integrations is fastest and easiest. The future costs will be hidden by the fact that only two or three applications are integrated. As the number of custom integrations begins to grow, it will become an increasingly slow and frustrating experience for the developers. Because integration evolves it is critical that the entire organization understands this fact. The selection of technology is considerably less important than this single concept.

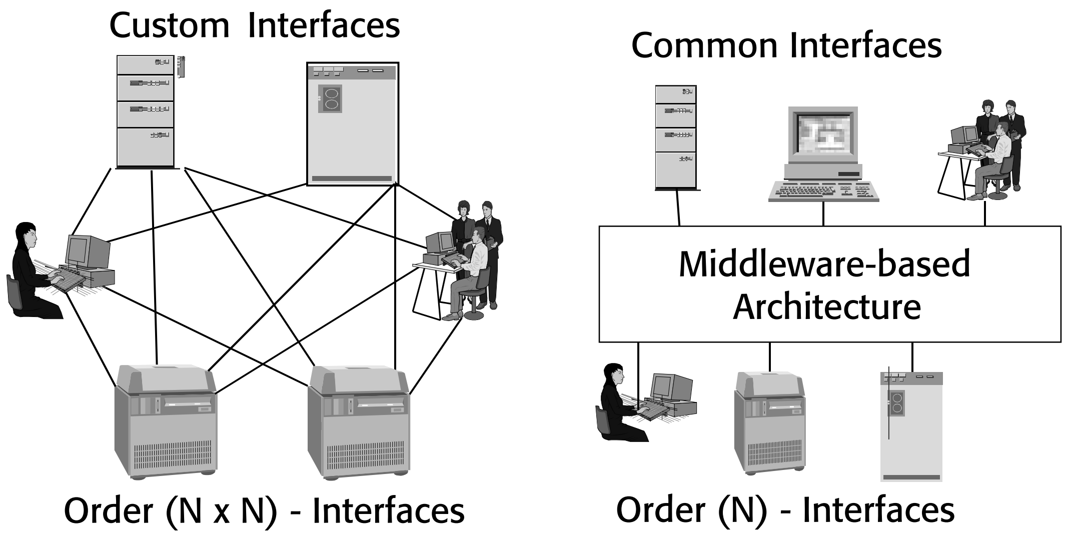


Figure 9.1 The impact of EAI architecture on enterprise resources can be significant.

At this stage there should be continued application of the EAI architecture and methodology as defined in stage 1 by other applications. Furthermore, the strategy and architecture should be extended to cover the multitude of open issues that remained at the end of stage 1. In conjunction with this work, the group responsible for EAI should be ensuring that adequate documentation is prepared to support the people who will be responsible for its use. Without adequate documentation acceptance will be impossible. In addition, training in the architecture, methodology, and technology is critical.

Finally, a bit of advice on the scope of the architecture definition. We say, with apologies to Albert Einstein, the architecture should be as simple as you can make it and no simpler. Architects have a tendency to make architectures very large and expressive. Trying to ensure that every possible aspect is defined and designed can lead to a very large and complicated structure. The results will be that the really important aspects are hidden behind many relevant but minute details. The problem that many architects will have is knowing what is simple enough but no simpler. Finding this balance is difficult but important. There is no recipe—it comes with experience and application. Some guidelines to consider are these:

- Assign a weight or value to every element of the architecture that describes its necessity, then focus on the most important elements. For example, you might use "required in architecture," "important to most applications," "requested by numerous projects," "important in long term," and "highly desirable" as your value descriptions.
- Take a sampling of the opinions of designers and architects in which you rank features and functions in terms of importance to them.

- Create two versions of the architecture—the full set and a light version. Analyze the differences between them.
- Evaluate how long it takes a sampling of designers to read and understand the architecture enough to apply it to a project.

When a simple but complete EAI architecture is defined and has been applied to several applications the organization has reached the second plateau and is ready to move to stage 3.

The EAI Organization

The goal of the third stage is to take the EAI architecture, methodology, and related technologies out across the organization. It is at this stage that the reach is to the enterprise. At this point there should have been enough success, documentation, and training to achieve significant results in efficiency and time to develop new applications. The focus is now on the organization.

This stage should be initiated by the team who has responsibility for the EAI architecture in conjunction with the support of the CIO. It requires both the education of key members of the IT organization and their buy-in to this approach. It is critical to have early successes in implementing EAI to achieve credibility across the enterprise for the EAI initiative. It should occur over a 6- to 12-month period, although some projects might be of a longer duration.

Perhaps the most important step in this stage is to have executive support for EAI. Without executive support, the architecture, methodology, and technologies are all optional. Within any organization there exist many experiences, beliefs, and political aspirations. Each of these can be overcome only by executive support (and then not always!).

The executive support can be achieved only by convincing the executive management team that the results of stage 1 and 2 were significant. This can come in the form of any of the following:

Return-on-investment analysis. An analysis of the cost of the implementation compared to the value received from the use of the application.

System feasibility analysis. An analysis of a system that demonstrates that the implementation was not feasible without EAI.

Implementation analysis. An analysis of the time frame required to implement a system with and without EAI.

These analyses are more powerful when there is a demonstrable system to back up the report.

With executive support and the other items discussed, the organization should begin to apply EAI to each and every new project and enhancement underway. In some organizations there may be a sudden urge to retrofit every application. Resist this temptation at all costs. The important rule of thumb at this point is to let nature take its course. Good interfaces come with good requirements based on real applications. EAI is not a sprint but a marathon.

Finally, some time and energy should go into EAI support technology. Specific areas that should be targeted because of their importance to deploying EAI-based applications effectively in the existing infrastructure of the enterprise are these:

- Directory services
- Security services
- Integration with systems management

As an organization begins to apply EAI technology broadly, the designers and developers should be given tools that allow them to find and reuse EAI interfaces and related information like meta data. EAI directory services should be implemented and well managed. Security will become a much larger issue at this point. Everyone usually ignores it up to now except the most sophisticated of organizations. Security is another area where costs can get out of control. The EAI architecture should be enhanced to make security practical, easy to understand and apply, and applicable to the needs of the business. Last, EAI is an extension of the network and should be managed. A key integration to be considered at this time is the integration with the network and systems management capabilities of the enterprise.

At the end of this stage the organization has reached a new plateau where it is applying EAI in a structured and disciplined manner. It is at this stage that time to market and efficiency can be measured. The enterprise is now ready to move on to the final stage.

The EAI Enterprise

The goal of this stage is to have the enterprise EAI-enabled. What differentiates this stage from stage 3 is the continuous learning, updating, and measuring that needs to be put in place to have a continually improving organization.

At this stage the entire organization is involved. The EAI architecture team is working with all elements of IT to continuously improve. This stage should go on for as long as EAI adds measurable value to the enterprise.

Once the organization begins to apply EAI as a normal course of business the enterprise will need to become more disciplined about the management of its architecture, methodology, and technology. There needs to be a feeling that learning is critical and will be fed back into the next version of the enterprise's EAI architecture. These changes should not be viewed as rewrites but as enhancements or corrections. A problem that tends to crop up at this point is the "not invented here" syndrome. If the person responsible for the changes does not feel a sense of ownership for the prior version and wants to totally rewrite it, then the enterprise is back to stage 1. Changes are good, but rewriting from scratch needs to be considered very carefully. Too much change can unbalance the whole enterprise.

How the changes are managed should be driven by the style best suited to the organization. Whether that is to have a dedicated group or distribute responsibility for sections and components doesn't matter. It does matter, though, that the mechanism is clearly defined.

Changes should be identified through two mechanisms:

Discussions with application architects, designers and developers who will provide subjective and objective assessments. These discussions should focus on where connectors needed to be developed, how difficult it was to train the staff, what tools did not meet project needs, and specific changes the staff would like to see implemented in the EAI architecture. Scalability and flexibility should also be discussed.

Metrics that objectively measure critical aspects of the application of EAI. These include application performance, resources required to develop an application, reliability, reuse of existing capability, and percentage of reuse versus new code developed.

The discussions will raise many issues. Some of these will conflict or seem unsolvable, but all should be captured and documented. The group responsible for the stewardship of the EAI environment should prioritize and assign the most critical for assessment of action.

Metrics should provide subjective data on the effect of EAI on the organization. The key metrics to be considered are these:

- Reuse of interfaces
- Time to build an interface
- Usage of directory services
- Performance of interfaces

Obviously, many more metrics are possible, but these are critical elements to be understood and managed. Because reuse of interfaces is where the greatest return on the investment will occur it is important to see how much is occurring and tune the organization around it. The time to build an interface is directly related to the quality of the technology used to create it. Organizations may try different technologies to reduce this time. The less time spent on integration and interfaces, the faster the time to market. The metrics will need to be compared with use of directory services. If the personnel are not looking, they can never find the services. Therefore, the efficiency problem may be with people, not technology or architecture. Finally, the greater the reuse of an interface or message, the higher the chance of having performance bottlenecks. Often these can be dealt with easily and efficiently through replication or hardware, but they first must be found before a bad event occurs.

If an organization is able to effectively apply, change, and measure the EAI architecture, methodology, and technology it has achieved the status of an EAI-enabled organization.

Lessons Learned

Lessons learned are the scars that remain from painful experiences. We have several scars we would like to share with you. The most important of the lessons we have learned are these:

- Roll out EAI in planned phases, starting small and working your way forward in measured steps.
- Don't be impatient because return on investment occurs after several uses.
- Train and support the users of the EAI architecture from analysis and design through deployment.
- Don't over-architect but evolve your architecture based on what works for your enterprise.

Consider each of these lessons as you start on your EAI journey. The most important lesson we learned through experience is that an organization should never apply EAI enterprise-wide from the start. It should roll it out in a controlled, evolutionary manner that stresses the importance of learning, planning, and successful implementations.

EAI technology should not be selected or brought in by itself as infrastructure. Infrastructure is never successful without the context of an application. A roadmap should be established in conjunction with the first application so that the EAI architecture is grown in a controlled manner. The first application of EAI does not demonstrate the full value of the architecture. Only subsequent iterations with reuse allow the full value to be apparent. It is important to remember that "quick and dirty" integration starts out costing less than EAI integration, but its costs quickly grow at an exponential rate compared with the linear growth rate of EAI-based integration, so some time and patience is required.

The major stumbling block to achieving EAI nirvana is the lack of knowledge and the biases that exist across the organization. To address this problem it is important to lay out the EAI architecture, document its use, and train, train, train! The EAI architecture developed by an organization should be made as simple as possible, and no simpler. Achieving this balance is the most difficult task the architect will face.

Getting the organization lined up behind your EAI architecture requires a successful demonstration of an implementation. Don't retrofit an existing application that has been integrated; apply EAI to a new system or enhancement. Then make sure you have the appropriate support infrastructure to maintain it properly.

Finally, plan to "learn as you go." Each use will bring out holes that need to be filled or improvements that need to be considered. Learn from each application, and take appropriate actions. The most effective way to learn is to instrument and measure your EAI environment.

Tying It All Together

The role of information systems in the enterprise continues to expand as businesses try to remake themselves in the age of the Internet. There can be many business reasons to employ EAI, but these three business trends are clear signs that an organization needs to become EAI-enabled:

- Increasing connectivity with suppliers and customers
- Demand to create new single-view components (e.g., customer, employee, and supplier)
- Demand to reengineer back-end processes for efficiency (e.g., straight-through processing)

Increased connectivity is the most significant driver for EAI. As connectivity is increased the need to tie what was once back-end processes into new front-end applications increases. The next driver is the need to bring together information and business processes to create a single view of the customer or other significant entity. Getting a total picture of a customer will allow the enterprise to provide better service. Finally, reengineering of processes often leads to new flows and integrations between applications. All of these trends increase integration significantly.

There are many reasons why an IT organization should employ EAI, but three trends within IT organizations are clear signs of the need to become EAI-enabled:

- Continued growth in purchased applications
- Competitive imperative to reuse components including monolithic mainframe and midrange applications
- Growing architectural sprawl

Purchased applications were once thought to be replacements for existing applications. Unfortunately, removing these legacy applications has proven much harder than expected. This has led to the need to reuse the existing software base to support new applications as well as the purchased applications. As the needs for integration continue to grow, it is becoming harder and harder to accomplish integration because of the diversity within the enterprise. As the architecture continues to expand, the complexity increases. EAI helps to reduce the complexity of these IT trends.

An enterprise that is experiencing one or more of these trends will need to plan their EAI strategy. An enterprise is EAI-enabled when they have a strategy, architecture, methodology, and technology. The organization will need to change as well. This will require the development of skills in the following areas:

- Developing explicit architectures
- Applying middleware software
- Legacy wrapping
- Component design

IT organizations will need to develop new core competencies. Architecture expertise is one of the most significant competencies that need to be developed. The future of the architecture is based on middleware and its application to wrapping legacy applications. Becoming familiar with different types of middleware and techniques for wrapping legacy applications will be a requirement for EAI. Finally, the core of EAI is to turn different applications and data into components that can be plugged into applications. The architecture of the future will be based on a component model. Therefore, organizations need to remake themselves around these areas.

Final Thoughts

EAI is more than a technology trend. It is a way to think about architecting your information system in order to leverage the existing IT investments more effectively when developing new applications. It can also be a significant boost to improving operational efficiency. Even though integration has been around since the beginning of information systems, it is ever changing. Integration has become more important than development in the creation of new applications. This book has provided all the information you need to make intelligent decisions to put your enterprise on the road to being EAI-enabled.

APPENDIX A

EAI-Related Vendors

It is risky to identify vendors and products in a book, especially in a fast-moving area such as EAI. We decided to include this appendix to give the reader resources for expanding his or her research and as a starting point for identifying EAI products to solve user problems. We are not endorsing any of the vendors or products in the list. Furthermore, the volatility of the EAI market means the list is incomplete. Some vendors and products will disappear, and others are not included in the list. Finally, we cannot provide any assurances about the quality or utility of the products offered by these vendors.

In order to make the listing more user friendly, we have organized the list of vendors into three categories:

Application server and object request broker vendors. These products are oriented to the creation and integration of components.

Message oriented middleware, message broker, and processware. These products are oriented to the development of multistep integration.

Tools for application and data integration. These products are oriented to data integration or provide tools that can help in developing a full EAI solution.

For each category, we list the vendor name, the product, and the Web site where you can find additional information.

Table A.1 Application Server and Object Request Broker Vendors

VENDOR	PRODUCT(S)	WEB SITE
Allaire	ColdFusion	www.allaire.com
Apple	WebObjects	www.apple.com
BEA Systems	ObjectBroker, WebLogic	www.beasys.com
Bluestone	Total-e-Server	www.bluestone.com
Computer Associates	Jasmine	www.cai.com
Compuware	Uniface	www.compuware.com
Expersoft	CORBAplus	www.expersoft.com
Fujitsu	Interstage	www.interstage.com
Gemstone	Gemstone/J	www.gemstone.com
Hitachi Software	TPBroker	www.hitachisoftware.com
IBM	WebSphere	www.ibm.com
Inprise	Visibroker, Application Server	www.inprise.com
Iona	Orbix, iPortal	www.iona.com
iPlanet	iPlanet Application Server	www.iplanet.com
Lotus	Domino	www.lotus.com
Microsoft ObjectSpace	Windows - COM / DCOM Voyager	www.microsoft.com www.objectspace.com
Oracle	Application Server 8i	www.oracle.com
PeerLogic	Live Content Broker	www.peerlogic.com
Persistence	PowerTier	www.persistence.com
Progress	Apptivity	www.progress.com
SAGA	EntireX	www.sagasoftware.com
Silverstream	Application Server	www.silverstream.com
Sun	Netscape Application Server, NetDynamics	www.sun.com
Sybase	Enterprise Application Server	www.sybase.com

Table A.2 Message Oriented Middleware, Message Broker, and Processware

VENDOR	PRODUCT(S)	WEB SITE
Active (Acquired by webMethods)	ActiveWorks	www.activesoftware.com
BEA Systems	MessageQ, eLink	www.beasys.com
Candle	Roma	www.candle.com
Constellar	Constellar Hub	www.constellar.com
Crossworlds	Crossworlds	www.crossworlds.com
Extricity	Alliance Manager	www.extricity.com
Hewlett-Packard	Changengine	www.hp.com
HIE	Cloverleaf	www.hie.com
IBM	MQSeries	ww.ibm.com
Iona	Integration Server	www.iona.com
Level 8	Geneva	www.level8.com
Mercator	Enterprise Broker	www.mercator.com
Microsoft	MSMQ	www.microsoft.com
Neon	MQSeries Integrator	www.neonsoft.com
OnDisplay	CenterStage eIntegrate	www.ondisplay.com
PeerLogic	Live Content Pipes	www.peerlogic.com
SAGA	Sagavista	www.saga.com
Sun / Forte	Conductor	www.sun.com
Talarian	SmartSockets	www.talarian.com
Tempest Software	Tempest Messenger Service	www.tempest.com
TIBCO	ActiveEnterprise	www.tibco.com
Viewlocity	Amtrix	www.viewlocity.com
Vitria	Businessware	www.vitria.com

Table A.3 Tools for Application and Data Integration

VENDOR	PRODUCT(S)	WEB SITE
Acta Technology	ActaWorks	www.acta.com
APILink	APILink	www.apilink.com
Ardent	Data Stage	www.ardent.com
Back Web	BackWeb	www.backweb.com
BMC Software	MAXM	www.bmc.com
Clientsoft	Client Builder	www.clientsoft.com
Cross Access	eXadas	www.crossaccess.com
D2K	ILN	www.d2k.com
Data Channel	Data Channel Series	www.datachannel.com
Data Junction	Data Junction	www.datajunction.com
ETI	ETI Extract	www.eti.com
iE	iE Integrator	www.ie.com
Information Builders	EDA Middleware	www.informationbuilders.com
International Software Group	ISG Navigator	www.isgsoft.com
Open Connect	IWare	www.openconnect.com
Netmanage	Rumba	www.netmanage.com
Poet Software	POET	www.poet.com
Praxis International	Omni Enterprise	www.praxisint.com
Scribe	Scribe Integrate	www.scribesoft.com
SmartDB	SmartDB	www.smartdb.com
SofTouch	CrossPlex	www.softouch.com
Softblox	Symbiant	www.softblox.com
STC	Xchange	www.stc.com
Synchrologic	iMobile	www.synchrologic.com
Thought	Cocobase	www.thoughtinc.com
webMethods	webMethods B2B	www.webmethods.com

APPENDIX B

EAI-Related Internet Resources

EAI Journal

www.eaijournal.com

Thomas Communications, Inc. publishes the *EAI Journal*. This magazine is published in paper form. It provides news and articles related to EAI, e-business, and integration. Thomas Communications provides an online subscription service known as WebFlash.

EAI Forum

www.eaiforum.com

The EAI Forum is hosted by Concept Five Technologies and offers a wide array of online articles highlighting EAI technologies and companies involved in e-business. E-clips is an online resource that provides links to recent articles from a variety of publications. In addition, the EAI Forum hosts Crossfire, an online debate and discussion live on the Internet.

ITQuadrant

www.messageq.com

www.eaiquadrant.com

ITQuadrant provides two EAI-related online resources. The first is MessageQ.Com. This online source provides news articles and information such as directories, previews, events calendar, and other features. It focuses on middleware. The EAIQuadrant is similar but focuses on EAI-related topics.

ITtoolbox

eai.ittoolbox.com

This Web site is part of the ITtoolbox series. The site provides information similar to the ITQuadrant site. It provides additional information on solutions and techniques.

Index