

---

# Enterprise Application Integration

**E**NTERPRISE computing has progressed enormously in just the last few years. Especially with the advent of the Web, not only is it possible for diverse organizations to automate and integrate their businesses and computer operations, it is imperative that they do so. Suddenly, as more and more corporations become Web-enabled and find themselves relying on a myriad of applications, the ability to evolve and integrate existing applications becomes significant.

Virtually all enterprise organizations at some time face the problem of integrating different applications and database systems. In addition, enterprise organizations must constantly evolve. This need to evolve occurs as enterprises strive for competitive advantages. In today's economy, it is rare for an organization to continue to be successful by merely maintaining the status quo. In a sense, enterprises are forced to evolve to stay at the forefront of their industries. Enterprises frequently find themselves having to merge with other enterprises, reorganizing their internal structure, and adopting new technologies and platforms as they strive for competitive advantages. More and more, they are adopting an eBusiness strategy. The failure of the "dot-com" business-to-consumer (B2C) economy has not affected the need for traditional enterprises to adopt an eBusiness strategy.

Enterprises still consider the eBusiness model to be an effective medium. The eBusiness model is particularly useful for managing purchasing and supply-chain issues, managing customer relationships and providing customer service, and providing Web-based applications and services. (An example of such a Web-based service is an online customer service application for bill payment and presentment.) Since it is imperative that enterprises adapt to business and technology-driven changes, they need an eBusiness model more than ever to adapt their existing business processes, applications, and enterprise systems to these changes.

Furthermore, it is not a simple matter for an enterprise to discard its existing applications, or even overhaul its established business processes, to effect a change in its business model. These kinds of changes are financially expensive to undertake and daunting in terms of human resources. Many enterprises cannot afford to make such changes or discard existing systems. Thus, it is critical for enterprises to be able to leverage their investments in their existing enterprise infrastructure and applications.

In these situations, enterprise application integration assumes a great importance. Enterprise application integration (EAI) enables an enterprise to integrate its existing applications and systems and to add new technologies and applications to the mix. EAI also helps an enterprise to model and automate its business processes.

Enterprise application integration has always focused on a company's IT department integrating new software modules or applications with its existing systems. How did a company handle these integration scenarios before the advent of EAI, J2EE, and the Connector technology? Companies handled such integration with a great deal of difficulty and often significant expense. Often, systems were merged by bringing in teams of expensive consultants, with little guarantee that they would deliver satisfactorily. Several years after undertaking these projects, it was not uncommon for companies to throw up their collective "corporate hands", write off the hundreds of thousands—if not millions—they had spent, and walk away from the project.

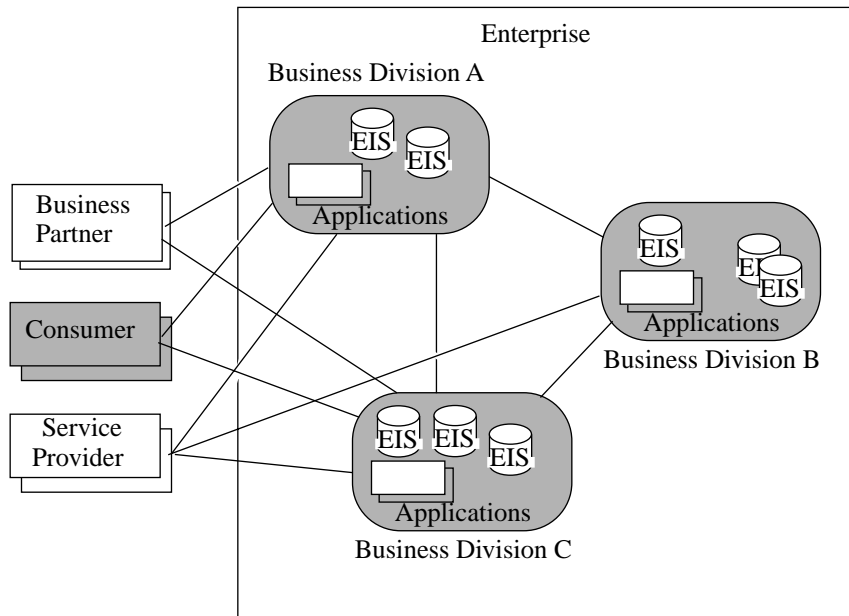
Enterprise organizations also must weigh the cost of replacing existing systems with new systems with the cost of merging existing systems with new systems. Discarding existing systems is never an easy choice: companies typically have invested huge sums of money to install, use, and customize these systems. Not only are their personnel comfortable with using these systems, even if the software is rife with drawbacks, but often the company's way of doing business has evolved to fit with these systems. It's difficult to just walk away from such an investment. Likewise, bringing in a replacement system has its costs: there's the purchase price of the new system, plus the training and customization costs. The investment in the new system can be as large, if not larger, than the investment in the existing system.

Companies also have the option to keep their existing systems and find the means to combine their functionality. In addition to retaining the existing systems, companies can integrate them with new applications to enhance functionality. The key with this option is the cost of integrating the separate applications and systems. EAI has grown out of this need to simplify the process of integrating applications and data.

## 1.1 What is Enterprise Application Integration

Enterprise application integration (EAI) entails integrating applications and enterprise data sources so that they can easily share business processes and data. Integrating the applications and data sources must be accomplished without requiring significant changes to these existing applications and the data.

Before EAI, integrating applications and data within a corporate environment has been an expensive and risky proposition. As we noted previously, companies were trying to combine applications that often ran on different hardware platforms and had no protocols for communicating with other software packages outside of their own narrowly defined realm. In a sense, companies had “islands” of business functions and data, and each island existed in its own, separate problem domain. (See Figure 1.1.)



**Figure 1.1** A Typical Enterprise Domain

How did an enterprise try to fix this situation? The company would bring in a team of consultants and embark on a long and expensive process of determining the feasibility of integrating their systems, designing the integration approach, and finally developing and implementing the procedures (both manual and computer-

ized) to achieve the integration. Sometimes, the analysis phase determined that it was not economical or possible to integrate the particular systems. Even when the integration did go forward, it might take years to accomplish. There was often no guarantee of success. Projects were often abandoned because of cost overruns or the belated recognition of significant difficulties. Even when they did complete, the resulting patchwork solution might be fraught with its own set of problems.

EAI represents a different approach to this problem. EAI defines semantics for application and data integration. That is, EAI defines a standard methodology or approach for applications and data sources to communicate. By supporting this standard, applications can easily communicate with other applications and data sources. The pieces in the integration puzzle—such as an underlying database management system (DBMS)—can change, but because of this common methodology, the replacement piece can be plugged in and the communication can continue uninterrupted.

There are many real-world examples of EAI, particularly in the banking and financial services and the telecommunications industry. Take AT&T for example. AT&T started as a phone service provider, then added cable television services and wireless service. Later, it became a broadband provider. The company has grown and evolved by merging with other companies and acquiring other businesses. As a result of this growth, and before its current plans to break into four different companies, AT&T needed to integrate its online customer services. It had to integrate its bill presentment for all services, payment for services, and its overall customer service. This entailed integrating access to the existing applications that provide these services.

By focusing on integrating business processes and data, EAI encompasses both the distribution of such processes and data and the concept of reusing modules. Most importantly, EAI approaches this integration as a process separate from the different applications. That is, someone can integrate various applications with each other, and with underlying data sources, without having to understand or know the details of the applications themselves.

EAI is best suited for environments that are heterogeneous rather than homogeneous. Heterogeneous environments are those whose applications and data do not all reside within the same environment, such as the AT&T example just discussed. A company may have reached this point because of acquisitions or mergers with other companies, where they have been compelled to absorb some other company's systems into their own environment. They may have been trying to increase their capacity—or avoiding replacing existing systems—by patching their own internally developed systems or other purchased systems onto their core systems. Or, they may be supporting large numbers of users on distributed systems with a multitude of platforms.

## 1.2 Web-driven Application Integration

With the advent of the Web, enterprise application integration has taken on a larger significance beyond that of merging application systems solely within an enterprise. Enterprise servers now handle and maintain huge amounts of data and business logic. Furthermore, because the Web enables easy information and service access, it has become a principal means of communication. An enterprise must be able to make its business data accessible to others, from internal employees to external partners, suppliers, and buyers. Employees require access to the enterprise data to keep abreast of company policies and developments, and to carry on the internal business of the company. For example, employees file their expense reports through a Web interface. Business partners may be communicating important technological information. Buyers and suppliers need access to enterprise data to facilitate the parts ordering and delivery process.

Providing services through the Web is rapidly becoming the emerging trend. Enterprises are recognizing that it is important for them to provide more of their services, such as customer support and product catalogs, through the Web. Enterprises have come to see that having such services available both in a traditional manner and over the Web enhances their business. The technology scenario is evolving at a breath-taking pace, and EAI is now increasingly being driven by Web-driven requirements and technologies.

Web-driven application integration, by making data and services more easily and widely accessible, places additional security requirements on an enterprise. All access to enterprise servers must happen in a secure manner. No company can risk the loss of data, or worse, having the integrity of their data compromised in any way. Likewise, such server access must also be transactional to maintain data integrity.

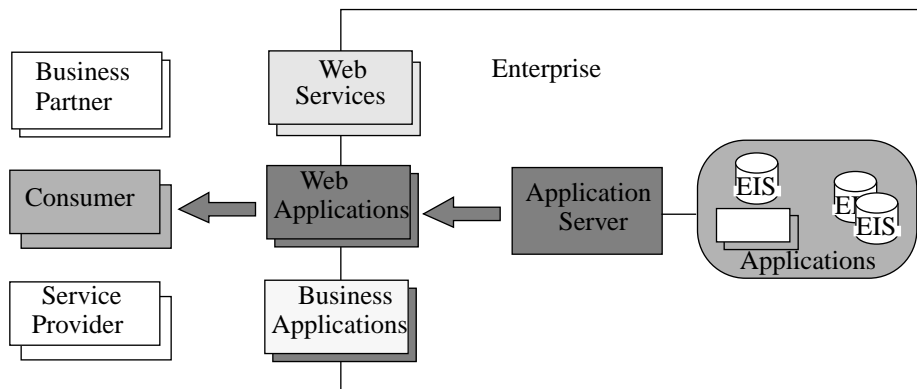
And, lastly, it's necessary for all this to happen in an environment that is scalable. Whether an enterprise starts large or small, the need for access to its systems is bound to multiply. An enterprise cannot risk using a system that does not have the capability to scale to many users over time. For example, an online stock trading application offered by the financial services industry must have the capability to handle transactions whose numbers can increase rapidly. It is best, too, if the enterprise can retain the flexibility to develop and add in new applications and extend its existing applications.

As more and more businesses establish a presence on the Web, Web-driven EAI becomes more and more essential. Enterprises need to integrate their existing applications and enterprise systems to drive their business-to-consumer and business-to-business interactions, plus their other Web services. In fact, success in eBusiness is driven by an enterprise's ability to integrate existing applications and extend the reach of these applications to Web-based access.

Up to now, applications were classified as either front-office or back-office applications. Front-office applications are considered to face the customer or end user. Front-office applications include applications for customer relationship management and marketing automation. Back-office applications provide the information infrastructure for running the back-end business processes of an enterprise. Applications provided by an enterprise resource planning (ERP) system are good examples of back-office applications. Traditional EAI focused on integrating the front and back office applications. However, traditional EAI is becoming Web-driven EAI. Rather than being targeted to the front end or the back end, most EAI applications are now integrated for the front and back ends and Web enabled.

Just as it is imperative for an enterprise information system (EIS) to move to a web-based architecture, there is also a need for enterprise applications to be deployed on widely adopted, standard application platforms. Enterprises now regard application servers as mature platforms for developing Web-based applications. As Figure 1.2 shows, application servers are particularly appropriate for the B2C and business-to-business (B2B) areas that place so much stress on application integration. The application server provides a natural point of integration between an enterprise's existing enterprise information systems and the Web-based applications. The application server also helps handle transactions and can be scaled as needed. The J2EE application platform is the technology of choice for enterprises and application vendors.

Figure 1.2 illustrates this Web direction to which enterprises are currently moving. The success of the Java programming language and the J2EE platform are also responsible for this Web-driven application integration, in large part because they make it easier to develop and implement Web-based applications.



**Figure 1.2** Web-driven Application Integration

To maximize this Web-driven application integration, enterprises are turning more and more to the Java programming language and the J2EE platform. Java is a platform-independent computer language that is designed for the Web, and it is a successful, widely adopted platform for enterprise application development.

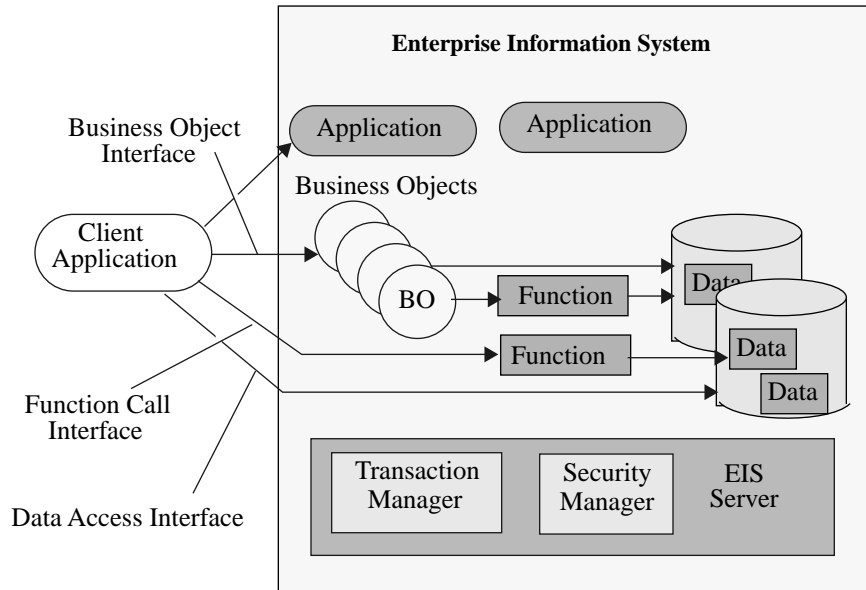
In addition to the Java platform, enterprises are using XML to exchange corporate data across application domains. XML is a platform-independent way of representing data formats, and it is invaluable for exchanging data among different entities. There is a synergy between XML and Java. XML is to data what the Java programming language is to application services. Because of XML's platform-independent features, it serves as a foundation for the current generation of Web technologies.

### 1.3 Enterprise Information Systems

Before delving into the details of EAI, it is useful to understand the definition of an enterprise information system. An enterprise requires certain business processes and underlying data to run its business. An enterprise information system encompasses these business processes and information technology (IT) infrastructure. Typically, the enterprise business processes include applications for handling payroll processing, inventory management, manufacturing production control, and financial accounting (accounts payable and accounts receivable).

We define an enterprise information system as an application or enterprise system that provides the information infrastructure for an enterprise. Typically, an EIS consists of one or more applications deployed on an enterprise system. An EIS provides a set of services to its users. Services exposed to clients may be at different level of abstractions—including the system level, data level, function level, and business object or process level.

Graphically, this might look as shown in Figure 1.3. In this EIS environment, the applications reside on the application server. The application server has a vendor-specific infrastructure, particularly regarding such services as transaction processing, security, load balancing, and so forth. Different vendors may supply the applications that sit on the server, or they may be developed by the IT department in house. Applications have been written in various languages, such as COBOL, C, and C++. There are application programming interfaces (APIs) for clients to access the different applications. An API is some routine that allows a client to do such operations as create a purchase order or update a customer record. The data access interface represents the means of access to the legacy datastores or relational databases. The business object interfaces are abstractions representing the business-specific logic for accessing functions and data.



**Figure 1.3** Enterprise Information System Environment

Many different applications and systems qualify as EISs. Typically, EISs include the following:

- Enterprise applications that have been developed by an enterprise specifically to meet its business needs. These are considered to be custom applications. Typically, legacy applications run on different computing environments. In addition, they are developed using different programming languages, such as C and COBOL.
- Applications that are part of an ERP suite of applications. ERP applications cover a wide range of functions, including inventory management, production control, human resources. Logistics applications are another set of ERP applications.
- Transaction programs running on a mainframe transaction processing system.
- Legacy databases that manage data critical to the business processes of an enterprise.

For a variety of reasons, EISs vary greatly even within the same enterprise. Usually, EISs vary because of the following:

- Enterprises purchase or implement different EISs over a period of years as their business needs grow.
- Enterprises deploy enterprise applications on different platforms or architectures.
- Enterprises customize an EIS to fit their own unique business needs.

Typically, an enterprise develops EISs over time, as a need for a particular EIS arises. For example, an enterprise may start out by purchasing a manufacturing system. Over the years, as its business grows, it incrementally adds different accounting packages, customer support, human resources, and so forth. It may be able to add some systems to the platform that hosts its manufacturing operations. However, other packages require different platform capabilities, or have only been developed for a particular platform or architecture. Not only does the enterprise add the new software systems, it also buys additional hardware that may be completely different than its original configuration. (The AT&T example mentioned earlier is another good illustration of this process.) It is easy to see that when an enterprise has been in business for a long time, it may very well have EISs in use that have been developed and installed on different computing platforms and architectures.

It is not uncommon for a large, established enterprise to have a few applications that run on a mainframe transaction processing system. These mainframe-based systems may have been purchased years ago. The same enterprise runs other applications that may be part of an integrated ERP suite of applications.

In addition, it is typical for an enterprise to customize its applications to its own enterprise-specific business processes. This level of customization can vary greatly. For example, an enterprise may purchase an off-the-shelf ERP application, and then customize the application so that it addresses its specific business processes. At the same time, it may develop other applications internally, using its own employees or consultants. These internally-developed applications are completely custom applications, designed to specifically meet the enterprise's business needs.

## 1.4 Challenges in EIS Integration

EISs differ significantly, in terms of their level of technological support, administrative and technological restrictions, ability to integrate with other systems, and exposure to low-level system details, as follows:

- **Level of technological support**—EISs vary greatly in their level of technological advancement. For example, there are vast differences in the support provided for transactions and security. Some EISs are rather primitive, and they may offer no support for transactional access. Or, if they do offer some support, it is limited in scope. Other EISs are more advanced in supporting a transaction and security infrastructure. They may allow transactional access to their resources. Or, they may support a two-phase commit protocol and distributed transactions, and thus be able to participate in transactions with other EISs.
- **Administrative and technological restrictions**—Many EISs impose specific technology and administrative restrictions on their users. These EISs are legacy systems or applications that have been in existence for a long time and their usage requirements may be more rigidly structured. For example, in some legacy systems it may be difficult to create new user accounts. Other legacy systems are difficult to extend to support development of new applications. An enterprise with such a legacy system must adapt to its shortcomings, but still must find a means to integrate the legacy system with other systems and new Web-based applications.
- **Ability to integrate with other systems**—EISs also differ in terms of their application programming models and client APIs, which makes it difficult to integrate these different EISs. These differences exist because most EISs were developed using architectures and technologies that best suited a certain class of enterprise applications and were prevalent at the time the application was initially developed. In addition, these EISs were developed when integration and interoperability with other types of systems and EISs may not have been the primary design goal.
- **Exposure to low-level system details**—Client APIs for these EISs may differ in the low level transaction and security management details they expose to application developers, and this makes it more complex to integrate EISs. The application developer must understand the programming details of the EIS's low level client API to properly integrate with the EIS. For example, suppose an EIS defines its client API using a C library. The C library defines methods that client applications use to manage transactions and perform transactional access to the EIS. Such a library may even expose the distributed communication

mechanisms between client applications and the EIS. The application developer now has the added task of understanding this C library—and the low-level mechanisms exposed through this API—to use the client API. This additional complexity increases the development effort in enterprise application integration.

Given the complex nature of application development and EIS integration, it is important that developers use standardized application development tools and integration frameworks.

Transactional access to EISs is also important in terms of EIS integration. Enterprises run their businesses using the information stored in their EISs—the success of an enterprise critically depends on this information. An enterprise cannot afford to have an application cause inconsistent data or compromise the integrity of data stored in an EIS. Various applications require ensured transactional access to the EISs.

Secure access to its EISs is also of critical important to an enterprise. An enterprise must be able to depend on the information in its EIS for its business activities. Any loss or inaccuracy of information, or any unauthorized access to the EIS, is extremely costly to an enterprise.

Scalability is another important requirement. Over time, enterprises typically increase their relationships to suppliers, buyers, and partners. Their applications, particularly those that access EISs, must be scalable and able to support a large number of clients. To accomplish this, use of connection pooling becomes an important requirement for EIS integration.

Additionally, enterprises must consider their existing application investment and a cost effective integration plan. Most enterprises and EISs have invested sizeable amounts in their existing application code and infrastructure. While they recognize the need to migrate to a J2EE platform, they must accomplish this migration incrementally rather than in one step. An incremental migration lets them get maximum use from their existing systems, but still gradually add new functionality as J2EE components and make more and more of their existing applications J2EE accessible. During this migration process, they can rely on application server and system software vendors to manage the system-level complexity of transactions and security, and thus let their application developers focus on solving business domain problems.

## **1.5 Enterprise Application Integration Approaches**

There are a number of different approaches to achieving enterprise application integration. We have identified five approaches that we feel are typically used to inte-

grate existing enterprise information systems with enterprise applications. These are:

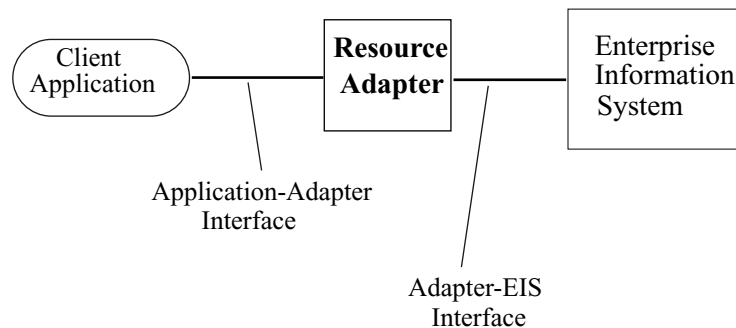
- Using a two-tier client-server approach
- Using synchronous adapters
- Using asynchronous adapters
- Using a message broker approach
- Using an application server-based approach

### 1.5.1 Two-tier Client-Server Approach

This approach is based on the two tier client-server model, and it is typical of applications that are not based on the Web. It was a widely used approach prior to the advent of Web-based applications, but is less used now.

With this approach, an EIS provides an adapter that defines an API for accessing the data and functions of the EIS. A typical client application accesses data and functions exposed by an EIS through this adapter interface. The client uses the programmatic API exposed by the adapter to connect to and access the EIS. The adapter implements the support for communication with the EIS and provides access to EIS data and functions.

Communication between an adapter and the EIS typically uses a protocol specific to the EIS. This protocol may provide support for security and transactions. It also typically supports content propagation from an application to the EIS. Most adapters expose an API to the client that abstracts out the details of the underlying protocol and the distribution mechanism between the EIS and the adapter. See Figure 1.4.



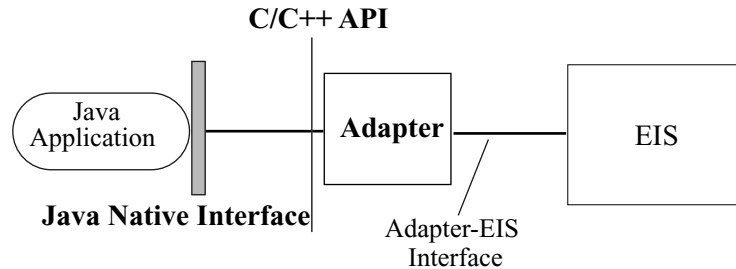
**Figure 1.4** EIS Resource Adapter Approach to EAI

Typically, a resource adapter is specific to a particular EIS. However, an EIS may provide more than one adapter that a client can use to access the EIS. Because the key to EIS adapters is their reusability, EISs, or independent software vendors (ISVs), try to develop adapters that employ a widely-used programming language and expose a client programming model that has the greatest degree of reusability.

An EIS may provide a simple form of an adapter, where the adapter maps an API that is specific to the EIS to a reusable, standard API. Often, such an adapter is developed as a library. When developed as a library, the application developer can use the same programming language to access the adapter as she uses to write the application, and no modifications are required to the EIS. For example, a Java application developer can use a Java-based adapter—an adapter written in the Java programming language—to access an EIS that is based on some non-Java language or platform.

An EIS adapter may be developed as a C library. ( See Figure 1.5.) A Java application uses a Java Native Interface™ (JNI™) interface to access this C library or C-based resource adapter. The JNI is the native programming interface for Java, and it is part of the Java Developers Kit (JDK™). The JNI allows Java code that runs within a Java Virtual Machine to operate with applications and libraries written in other languages, such as C and C++. Programmers typically use the JNI to write native methods when they cannot write the entire application in Java. This is the case when a Java application needs to access an existing library or application written in another programming language. (While the JNI was especially useful before the advent of the J2EE platform, many of its uses may now be replaced by the J2EE Connector architecture.)

The JNI interface to the resource adapter enables the Java application to communicate with the adapter's C library. While this approach does work, it is complex to use. The Java application has to understand how to invoke methods through the JNI interface. This approach also provides none of the J2EE support for transactions, security, and scalability. The developer is exposed to the complexity of managing these system-level services, and must do so through the complex JNI interface.



**Figure 1.5** Using the Java Native Interface

Another, more complex form of an EIS adapter might do its “adaptation” work across diverse component models, distributed computing platforms, and architectures. For example, an EIS may develop a distributed adapter that includes the capability to do remote communication with the EIS. This type of adapter exposes a client programming model based on a component model architecture.

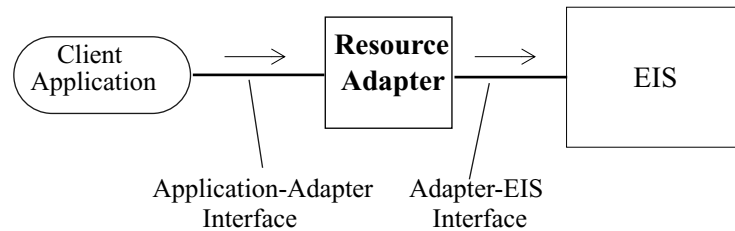
Adapters use different levels of abstraction and expose different APIs based on those abstractions, depending on the type of the EIS. For example, with certain types of EISs, an adapter may expose a remote function call API to the client application. If so, a client application uses this remote function call API to execute its interactions with the EIS.

An adapter for other types of EISs may expose a data-based programming model for the client application developer. When the adapter exposes this sort of programming model, a client application accesses EIS data using either a data representation and access model specific to the EIS or relational data model.

It is also possible for an adapter to build on the API (the remote function call or data access API) exposed by the EIS. That is, a more advanced adapter may use the lower level abstraction layer exposed by the EIS to build a higher level business process or business object abstraction for client application developers.

### 1.5.2 Using Synchronous Adapters

An adapter can expose either a synchronous or an asynchronous mode of communication between the client applications and the EIS. Figure 1.6 illustrates using adapters designed for synchronous communication. Adapters designed for this approach provide a synchronous request-reply communication model for use between an application and an EIS.



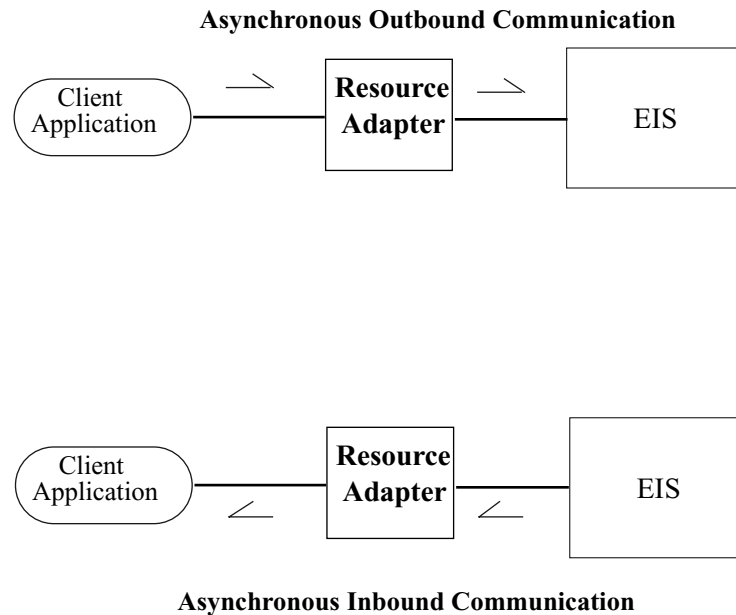
**Figure 1.6** Using a Synchronous Adapter

How might a synchronous adapter work? As an example, let's consider an adapter that defines an API that includes a remote function callable by an application. This remote function creates an accounts receivable item in the EIS. When an application wants to interact with the EIS to create an accounts receivable item, it invokes this remote function on the EIS. The application that initiated the call then waits until the function completes and returns its reply to the caller. The reply contains the results of the function's execution on the EIS. An interaction such as this is considered synchronous because the execution of the calling application waits synchronously during the time the function executes on the EIS.

One form of synchronous adapter allows bidirectional synchronous communication between an application and an EIS. This type of adapter enables an EIS to synchronously call an application.

### 1.5.3 Using Asynchronous Adapters

Asynchronous adapters provide another approach to application integration. Figure 1.7 provides a high level view of this form of communication.



**Figure 1.7** Using an Asynchronous Adapters

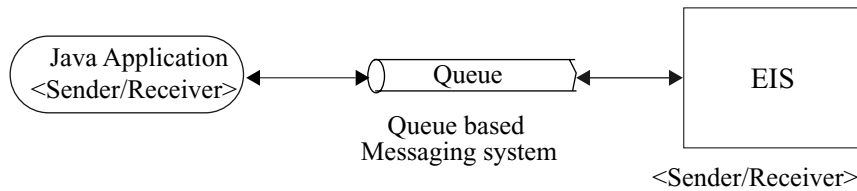
Let's use the same example of an adapter than exposes an API with a remote function that permits an application to interact with the EIS and create an accounts receivable item. This function is callable by an application.

With asynchronous communication, an application calls the remote function to create a new accounts receivable item in the EIS. The application makes the remote call, then immediately returns and continues its own processing. The remote function is sent to the EIS. The EIS handles the function and returns some reply information to the application as a separate asynchronous invocation. The resource adapter dispatches the asynchronous call from the EIS to the application.

The important point to remember is that the application does not suspend its own processing while the remote function executes on the EIS. Rather, the application continues its own work and receives notification at some later point of the results of its earlier remote function invocation. In addition, an EIS is able to asynchronously invoke or call an application.

**1.5.4 Queue-Based Approach**

Asynchronous message-based communication may also be used to integrate enterprise applications and EISs. There are two forms of asynchronous messaging: queue-based messaging and publish-subscribe messaging. A message broker may provide either one of these forms of messaging.

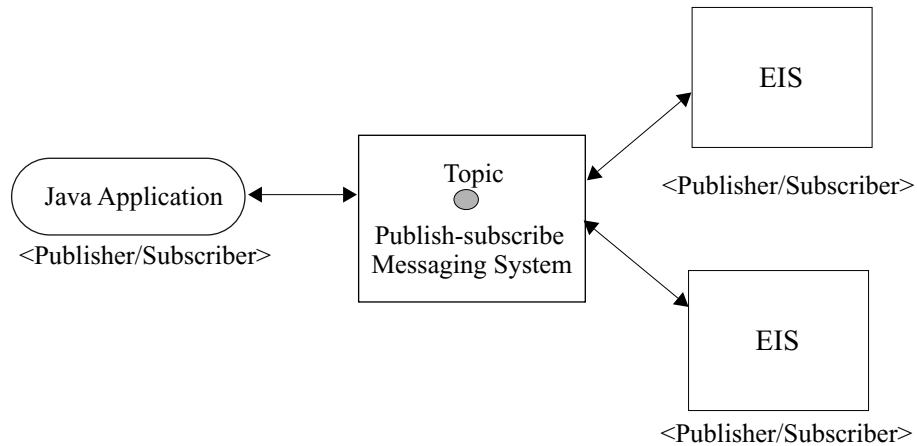


**Figure 1.8** Using a Message Queue for EIS Integration

Figure 1.8 illustrates queue-based communication. Queue-based communication, which is also called point-to-point messaging, involves one application sending a message to a message queue. With queue-based communication, a queue that is independent from both the sender and receiver applications acts as a message buffer between the communicating applications. The sender application sends a message to this queue, while the receiver application receives its messages from the same queue.

**1.5.5 Publish-Subscribe Approach**

The publish-subscribe approach works differently from the queue-based approach.



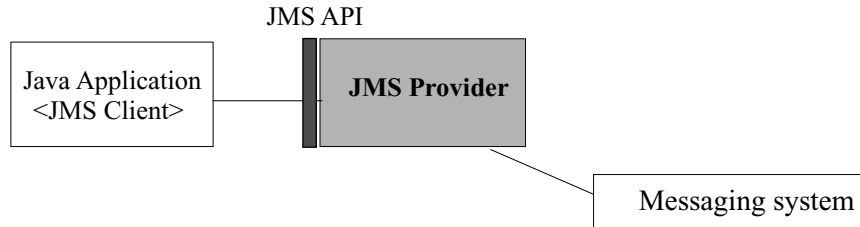
**Figure 1.9** Using a Publish-Subscribe System for EIS Integration

Figure 1.9 illustrates publish-subscribe messaging. This might be a stock quote service that publishes messages—updated stock prices—to subscribed portfolio applications. With publish-subscribe messaging, there are message publishers, who produce messages, and message subscribers, who register their interest in particular messages. There is also a separate publish-subscribe facility that acts as the integration point—publishers publish messages to this facility and the facility delivers messages to subscribers.

Here’s how publish-subscribe messaging works. A publisher application publishes messages on a specific topic, such as up-to-the-minute quotes on a specific stock symbol. Multiple applications can subscribe to this topic and receive the messages published by the publisher. The publish-subscribe facility takes the responsibility of delivering the published messages to the subscribing applications based on the subscribed topic.

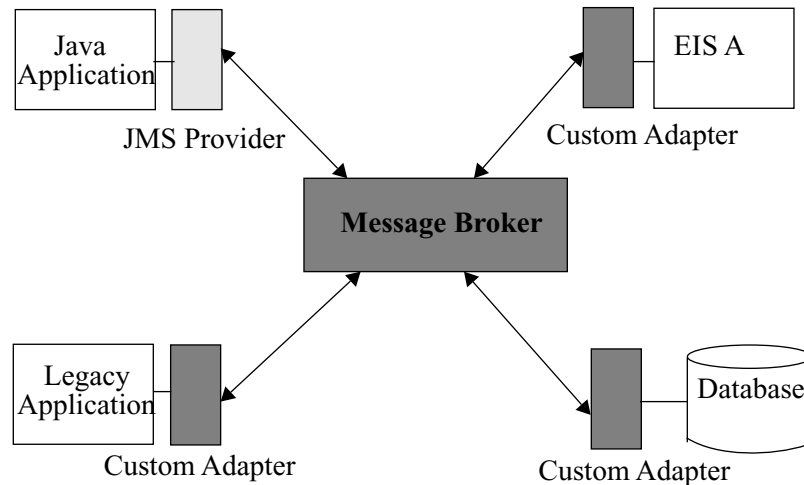
When an application needs to use either queue-based or publish-subscribe messaging, it must also hook into a messaging system that provides these mechanisms. The application typically uses an API exposed by the messaging system to access the messaging services. Typically, the messaging system uses a messaging adapter, also called a provider, to implement the messaging API. Java Message Service (JMS) provides an API for enterprise messaging systems. Applications, called JMS clients, use the JMS API to access the messaging service and either a

queue-based or publish-subscribe messaging system. (See Figure 1.10.) Refer to Chapter 6, Asynchronous Messaging, for more information on JMS.



**Figure 1.10** Using a JMS Provider

Figure 1.11 illustrates using a message broker for EIS integration. Notice that an adapter enables an application to access the message broker. In this scenario, an adapter maps the application-level interface for the message broker to the underlying asynchronous messaging mechanisms supported by the message broker, plus the adapter maps the message formats supported by the message broker. (The underlying messaging mechanisms supported by the message broker may be a queue-based or a publish-subscribe mechanism, for example.) Some adapters layer additional functionality between the application and the message broker. For example, they may add the capability to do message transformations—an adapter may transform application-specific messages to a format expected by the message broker. The message broker then converts the message to a format expected by the message receiver or subscriber.



**Figure 1.11** Using a Message Broker for EIS Integration

When applications and EISs use a message broker for integration and message delivery, the applications and the EISs can act as both message producers and consumers. For example, a financial accounting application can subscribe to messages that carry information on financial transactions. An order management application may send a message through the message broker that updates an account payable in the accounting application. Most message broker vendors provide vendor-specific adapters for popular EISs.

When an application and an EIS communicate using asynchronous messaging, they are considered to be loosely coupled. There are both advantages and disadvantages to a loosely-coupled integration. With a loosely-coupled integration between a target EIS and an application, the application can continue processing client requests without blocking on EIS performance or communication glitches. This improves scalability. However, application developers may find it difficult to program against an asynchronous messaging model. Also, these asynchronous messaging systems do not always support the propagation of security and transactional contexts.

A message broker may provide additional services for enterprise application integration. These additional services are: message routing, transaction management, reliable message delivery, message priority and ordering, and message transformation. We discuss these topics further in Chapter 6.

### 1.5.6 Application Server-Based Integration

Figure 1.12 shows how an application server can be used for integration with existing enterprise applications and EISs.

An application server is a natural point for application integration, because it provides a platform for development, deployment, and management of web-based enterprise applications. Application servers are the platform of choice for application that are developed using a multitier architecture.

A typical multitier application consists of three tiers: a client tier, a middle tier, and an EIS tier. The middle tier typically implements the business logic for an application. As part of its implementation of application business logic functionality, the middle tier might access data and functions associated with applications running on the EIS tier. The middle tier also serves up both static and dynamic presentation content to the client tier.

The EIS tier contains the systems that run existing enterprise applications and databases. As described earlier, these EISs can be custom or off-the-shelf applications.

The client tier is composed of different types of client applications. A client can be a Web browser-based HTML client or a peer application.

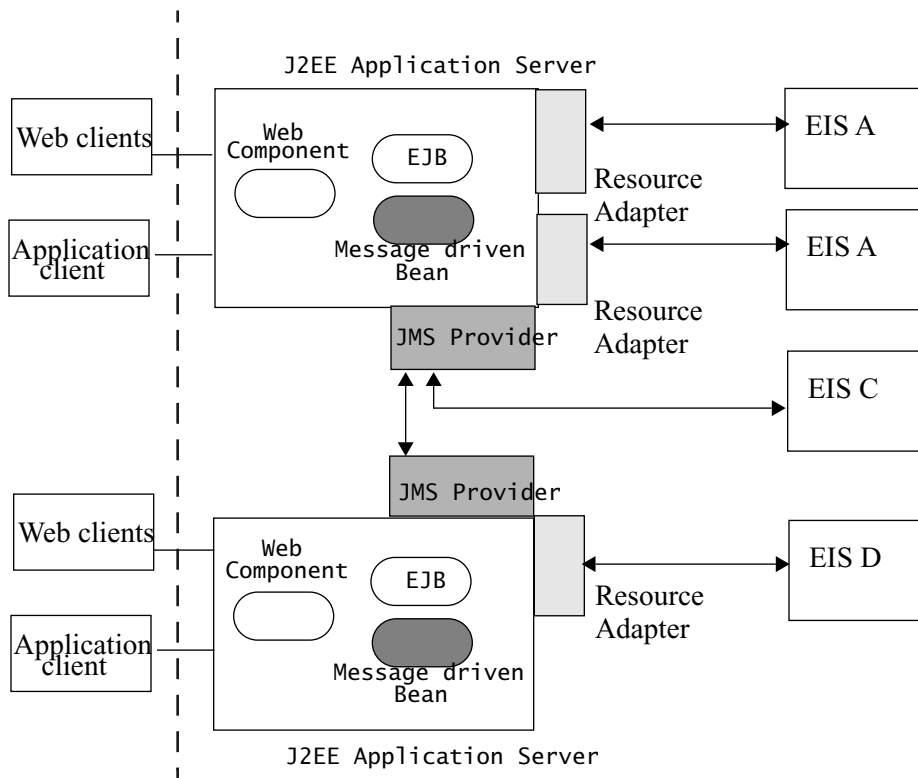
A typical application server supports a component-based model for developing applications. With this model, an application may be composed of different types of components, such as web components or business components. The application server provides deployment and runtime support for these application components. In effect, an application server provides an extremely useful platform for the development of Web-based, transactional, secure, distributed, and scalable applications. This increases the usefulness of an application server environment for enterprise application integration.

Typically, an application server provides a set of runtime services to its deployed components. These runtime services are hidden from the application components through a simplified application programming model. The services provided include:

- Support for transactions
- Security
- Load balancing and failover
- Database access
- Asynchronous messaging
- Distributed communications

- Web protocols
- XML support

It is possible to develop and deploy applications on an application server such that the applications can connect and aggregate access to multiple heterogeneous EISs and existing enterprise applications. When applications are developed with this ability to access multiple heterogeneous EISs, Web and business components that are deployed on the middle tier (or application server) use adapters to access the data and functions associated with the applications on these EISs.



**Figure 1.12** Application Server-Based Enterprise Application Integration

Application components deployed on the application servers use synchronous resource adapters to connect and access EISs. As explained earlier, this is tightly coupled integration between applications and EISs.

Application components can also use an adapter (or JMS provider) to a message broker to integrate with EISs based on asynchronous messaging. We explain this approach in greater detail in Chapter 6.

## 1.6 J2EE Connector Architecture and EAI

How does the J2EE Connector architecture fit in with the EAI scheme of things? To begin with, the Connector architecture is designed to simplify integrating J2EE components with EISs. The architecture makes it easier to connect J2EE components and applications to heterogeneous enterprise information systems (EISs). Examples of EISs include database systems, ERP systems, and main-frame transaction processing (TP).

How does the Connector architecture accomplish this? The Connector architecture defines a set of mechanisms, referred to as contracts, so that EISs can easily integrate with application servers and enterprise applications. These mechanisms are designed to be scalable, secure, and transactional in nature. These contracts exist between the J2EE application servers and the EISs.

The Connector architecture also defines a client interface API to enable J2EE application components to access a multitude of heterogeneous EISs. This client API is called the Common Client Interface (CCI).

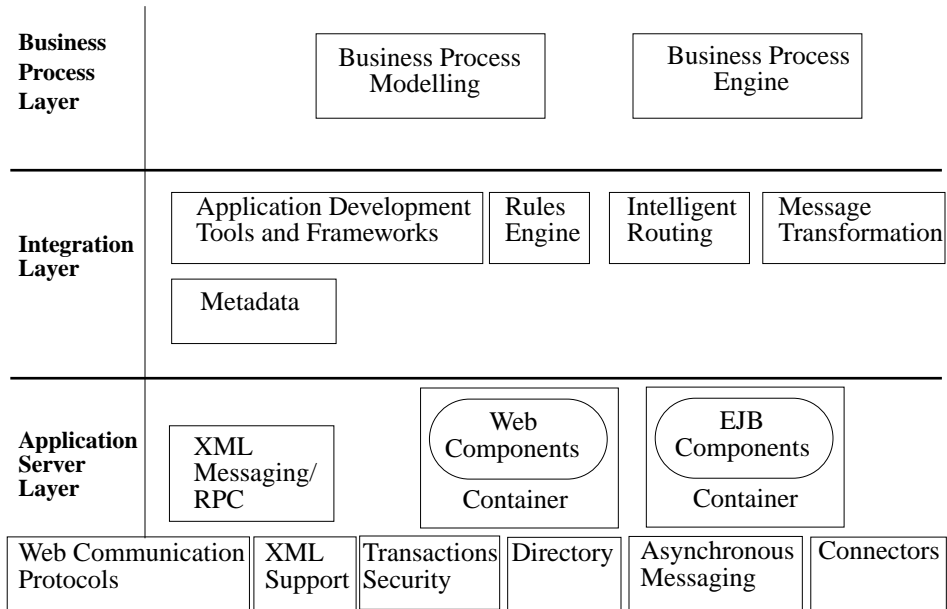
An EIS vendor that wants to participate in the Connector architecture must provide its half of the bargain—that is, the EIS vendor must support the Connector contracts. The EIS vendor can provide a standard resource adapter for its EIS, and this resource adapter can plug into any J2EE-compliant application server. (A resource adapter is a system-level software library that a Java application on the J2EE platform uses to connect to an EIS.) The resource adapter is the connection conduit between the enterprise application on the application server and the EIS.

Because the Connector architecture defines the resource adapter requirements, the EIS vendor is assured that his resource adapter will work with any J2EE-compliant application server. This means that the EIS vendor must only provide one standard resource adapter for all J2EE application servers, and not a separate adapter for each application server.

Likewise, the application server vendor, by following the terms of the Connector contracts defined for an application server, only has to extend its product once to support the Connector architecture. By supporting the Connector architecture, the application server also supports multiple EIS resource adapters, regardless of the EIS vendor.

Application integration in a Web-based, ebusiness environment encompasses three layers: a business process layer, an integration layer, and an application

layer. Each layer, in turn, holds technologies that serve as the application integration building blocks. See Figure 1.13.



**Figure 1.13** Application Integration Layers

The application layer technologies, which are based on the J2EE platform and utilize the Connector architecture, enable an application integration project to not only link with existing enterprise systems but also with the Web and other wireless applications.

A J2EE-based application server is at the bottom layer of this application integration platform. A J2EE application server provides value to the application integration platform through such services as:

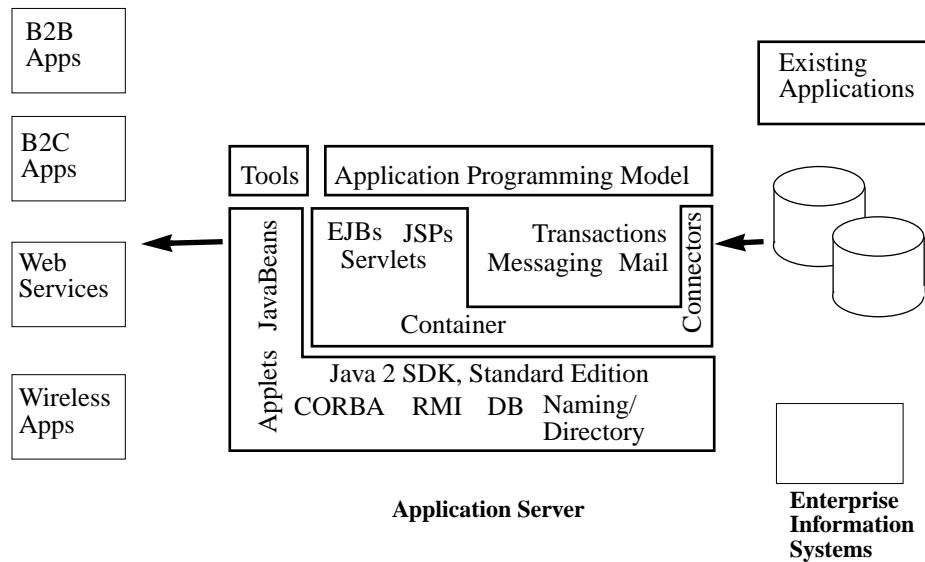
- The J2EE component-container model. This model includes the EJB container and such components as enterprise beans and message-driven beans. It also includes the JSP and servlet components defined in J2EE platform.
- Java Message Service. JMS provides support for asynchronous messaging.
- A set of APIs that support transactions, security, and naming and directory services.
- A set of APIs that add support for XML messaging and Remote Procedure

Calls (RPC). (It is anticipated that this support will be in future versions of the J2EE platform specification.)

The application integration platform adds an integration layer on top of the J2EE-based application server. This integration layer provides support for application development tools and frameworks. These development tools and integration frameworks are based on the J2EE application programming model, and they rely on metadata for generating and providing services. The integration layer also adds support for such functionality as a rules engine, intelligent message routing, and message transformation, all on top of the base functionality provided by the J2EE application server.

Lastly, a business process layer serves as the top-most layer for the platform, and represents an enterprise's unique way of doing business. Enterprises rely on software packages from different vendors to develop and manage their business processes. This business process layer exposes business process level abstraction by providing support for business process modelling and for the business process engine.

Figure 1.14 illustrates a typical application integration platform, with the J2EE platform and the J2EE Connector architecture together acting as building blocks for Web-driven application integration.



**Figure 1.14** Application Integration on the J2EE Platform

## **1.7 Conclusion**

There is a definite trend among enterprises towards integrating their existing enterprise applications and information systems with Web-based applications and services. More and more, enterprises must establish a Web presence and make their business services available to Web clients. However, at the same time, an enterprise cannot afford to discard its existing systems and applications, but must leverage these existing assets to be successful.

This chapter highlighted some of the tasks and challenges that face enterprise's compelled to integrate their information systems and applications and then expose these applications and systems to the Web. It also showed how the J2EE platform and the J2EE Connector architecture serve as building blocks for Web-driven application integration. The J2EE platform and the Connector architecture, by providing standardized integration contracts, have enabled application servers to serve a key role in the Web-driven application integration process.

This Web-driven application integration is a process that closes the gap between existing applications and Web-based applications and services. Ultimately, Web service and wireless clients, in a B2C or B2B context, will be able to initiate business processes that act on critical information maintained in EISs.

In the next chapter, we provide an overview of the Connector architecture and describe its role within the J2EE platform.