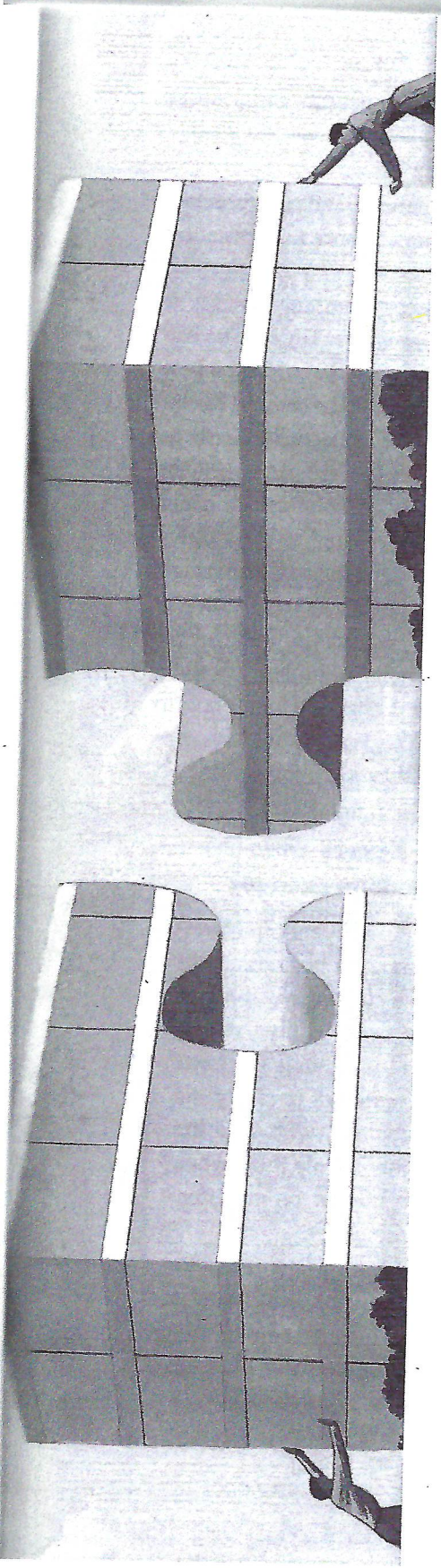
Linthicum (2004):
Next Generation Application Integrati



CHAPTER ONE

Approaching Application Integration

This chapter sets the stage for application integration concepts and introduces ways to view application integration solution patterns. The reader should focus on the concepts rather than the details. More technical details will come later in the book.

Application integration is a strategic approach to binding many information systems together, at both the service and information levels, supporting their ability to exchange information and leverage processes in real time. While this sounds like a pure technology play, the resulting information and process flow between internal and external systems provides enterprises with a clear strategic business advantage: the ability to do business in real time, in an event-driven atmosphere, and with reduced latency (see the tidbit on page 2). The business value of this is apparent.

Application integration can take many forms, including internal application integration—Enterprise Application Integration (EAI)—or external application integration—Business-to-Business Application Integration (B2B). While each form has its own set of eccentricities, once you dig into the technology, you'll find that both inter- and intracompany integration solutions share many common patterns. For example, there almost always has to be transformation technology present to account for the difference in application semantics, routing technology

to ensure that the information goes to the correct destinations, and rules processing to define integration behavior. However, there is much more to application integration.

Keep in mind that the application integration concept is nothing new. We've been dealing with mechanisms to connect applications together since we've had more than two business systems and a network to run between them. What is new is understanding the need for application integration solutions to support strategic business initiatives going forward, such as participating in electronic markets, supply chain enablement, Web visibility, Customer Relationship Management (CRM), and the real need to get all internal systems exchanging information and services. Indeed, as time marches on, we see the need to view application integration as a true paradigm, something that requires a great deal of business definition and architectural planning. Moreover, the application of new technology, such as integration brokers, to solve this problem brings more opportunity.

So, how do innovative enterprises leverage application integration? It's really a matter of understanding the need first, then the requirements, and finally how to solve the problem for their domain. Make no mistake: This is a difficult and complex process, but one you can handle when armed with the right information.

Moving to Real-Time Business Integration: An Example

Few examples illuminate the difference between the conventional (non-integrated) method of doing business and an integrated business more clearly than the purchase of a new car. Currently, a customer walks into an automobile dealership and orders a car. That order is then placed with the auto manufacturer. The manufacturer, in turn, orders the parts and creates the car, while the suppliers order raw materials to create the parts. Paper purchase orders are sent to the suppliers, who ship the materials and send paper invoices to request payment. Only then, when all the parts are received from the suppliers, can the car be manufactured and sent to the dealer—resulting in even more paper.

This process typically takes months, not weeks. It should only take days.

We need to think more comprehensively about how we capture and react to events. We need to recognize that all components of the integrated enterprise, or extended enterprise, affect the supply chain itself.

For example, when a customer walks into our car dealership and orders a car, or when the customer orders a car via the Internet, that action is, in and of itself, a business event that is captured. Our system must react to this event by performing several tasks instantaneously: logging the event, processing the rules bound to such an event, and moving information to other interested systems or humans.

The event must be logged so that it won't be forgotten should there be a failure as it is being processed. We need to process rules bound to the event, such as price limits and credit requirements. The internal (e.g., inventory) systems and external (supplier) systems must be informed of the event. Finally, the information created by this event, in this example, customer and car configuration information, must move forward to the appropriate systems. Typically, this should be a second, subprocess.

What is of note here is that all relevant systems are notified of the event and are supplied with all appropriate information, in real time, so that they can, in turn, instantly react to the event. In our car purchase example above, the sales event captured by our manufacturer's system generates an instant requirement for parts to create the car. In turn, this information triggers a cascading series of additional events within systems owned by the suppliers, events such as notifying a supplier of the raw materials required to build the parts. A single, primary event could thus trigger as many as several hundred other events, which, in turn, could trigger several thousand more events. It is exactly this chain reaction of events—events that serve a business need—that we hope to create.

Remember, this real-time application integration scenario is an instantaneous process. Within seconds of the initial order, the suppliers are processing requests for the raw materials, the factory floor is scheduling workers, and the logistic group is assigning resources in order to ship a car to a particular dealer. There may be hundreds of systems involved with the sale, creation, and movement of this car, all exchanging event information simultaneously. Of equal relevance is that all systems participating in the event will be notified instantly should there be any change along the supply chain. That is, if demand changes (e.g., if car sales are down) or if there is a parts shortage. Instantaneous notification is a two-way street, from orders to suppliers, from suppliers to orders.

Simply put, application integration is a complex problem. The simple reality is that most application integration projects exist just at the entry level. We have yet to see the real-time coupling of thousands of applications. This should not necessarily be discouraging. As with any complex problem, once it is broken down to its component parts, the solution becomes simply the aggregation of a number of solution sets. In this case, it's a combination of a variety of approaches and several types of technology. This seems to fly in the face of those who want to oversimplify the concept of application integration, thinking that a simple standard, such as XML, or a particular technology, such as application servers, holds the answers to all of their problems. Unfortunately, it's just not that simple.

The world of application integration is no different from the larger world of technology—it is advancing and changing rapidly. Ironically, as the technology changes, so does the problem it is designed to solve. The application integration problem is morphing from the very simple to the very complex, even as it moves from a departmental problem to an enterprise-wide problem, and, ultimately, to a trading community problem. Consequently, few companies have been able to get ahead of the "application integration curve." Without a complete solution, they remain short of discovering the full potential and benefits of application integration.

We are seeing that, as the problem grows, so do the potential benefits of the solution. The technology continues to respond to a perceived need. In this context, our pursuit of application integration is like chasing the tail of a growing beast. For now, that "beast" has remained ahead of us. A great deal of work remains ahead of us. But rest assured, a solution will be found and the once-unimaginable benefits of application integration will become an everyday reality.

As I've suggested above, as the problem domains become more complex, the application integration solution set evolves to address that growing complexity. No sooner is a "traditional" application integration problem solved (such as application-to-application and database-to-database integration), than the developed application integration expertise and technology is applied to more complex, but more rewarding, business issues.

Moving from Information-Oriented to Service-Oriented Application Integration

A clear trend is the movement away from information-oriented to service-based integration. Information-oriented integration provides an inexpensive mechanism to integrate applications because, in most instances, there is no need to change the applications.

While information-oriented integration provides a functional solution for many application integration problem domains, it is the integration of both application services and application methods that generally provides more value in the long run. That is the underlying theme of this book.

For example, a trading community looking to automate the processing of ordering raw materials may find that simply sharing information (order goes out, and confirmation comes in) is just fine to solve their integration problem. However, in another trading community, there may be a need to access remote services, such as the calculation of duty for intercountry trades. Again, you have to leverage the right approach for the business problem you are looking to solve.

Service-based application integration is not a new approach. We've been looking for mechanisms to bind applications together at the service level for years, including frameworks, transactions, and distributed objects—all in wide use today. However, the new notion of Web services, such as Microsoft's .NET strategy, is picking up steam as we attempt to identify a new mechanism that's better able to leverage the power of the Internet to provide access to remote application services through a well-defined interface and directory service: Universal Description, Discovery and Integration (UDDI).

The uses for this type of integration are endless, including the creation of composite applications, or applications that aggregate the processes and information of many applications. For example, using this paradigm, application developers simply need to create the interface and add the application services by binding the interface to as many Internet-connected application services as are required.

The downside, at least with service-based integration, is that this makes it necessary to change the source and target applications or, worse in a number of instances, to create a new application (a composite application). This has the effect of adding cost to the application integration project and is the reason many choose to stay at the information level.

Still, the upside of this approach is that it is consistent with the "baby step" approach most enterprises find comfortable when implementing solutions to integration problems. Service-based solutions tend to be created in a series of small, lower-risk steps. This type of implementation can be successful from the department to the enterprise to the trading community, but never the other way around—from the trading community to the department.

Application Integration Approaches

As we've come to appreciate, application integration is a combination of problems. Each organization and trading community has its own set of integration issues that must be addressed. Because of this, it is next to impossible to find a single technological solution set that can be applied universally. Therefore, each application integration solution will generally require different approaches. At this time, and in the foreseeable future, one-stop shopping is simply not an application integration reality.

Although approaches to application integration vary considerably, it is possible to create some general categories, which include

- Information-oriented
- Business process integration-oriented
- Service-oriented
- Portal-oriented

Information-Oriented

Technologists who promote the information-oriented approach to application integration argue that integration should occur between the databases (or proprietary APIs that produce information, such as BAPI)—that is, databases or information-producing APIs should be viewed as the primary points of integration (see Figure 1.1). Within Information-Oriented Application Integration (IOAI), there are many approaches. Information-oriented solutions can be grouped into three categories: data replication, data federation, and interface processing.

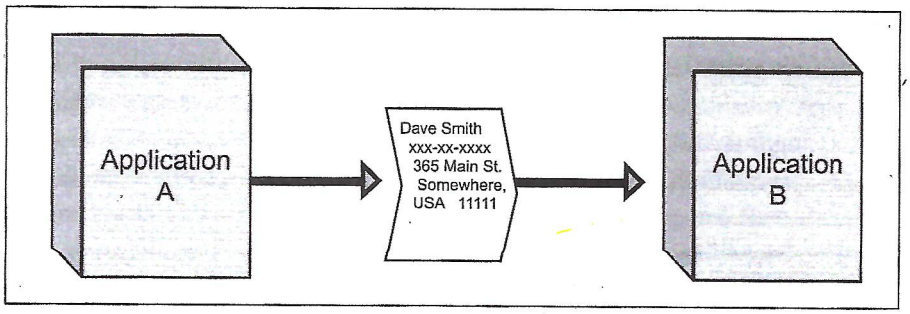


Figure 1.1 Information-Oriented Application Integration deals with the simple exchange of information between two or more systems.

Data Replication

Data replication is simply moving data between two or more databases. These databases can come from the same vendor, or from many vendors (see Figure 1.2). They can even be databases that employ different models. The fundamental requirement of database replication is that it accounts for the differences between database models and database schemas by providing the infrastructure to exchange data. Solutions that provide for such infrastructures are plentiful and inexpensive.

Many database-oriented middleware solutions currently on the market provide database replication services, as well. Replication services are accomplished by placing a layer of software between two or more databases. On one side, the data is extracted from the source database or databases, and on the other side, the data is placed in the target database or databases. Many of these solutions provide transformation services, as well—the ability to adjust the schemas and the content so they make sense to the target database.

The advantages of database replication are simplicity and low cost. Database replication is easy to implement, and the technology is cheap to purchase and install. Unfortunately, these advantages are quickly lost if methods need to be bound to the data, or if methods are shared along with the data. If these requirements exist, service-based solutions must be considered.

Data Federation

Database federation is the integration of multiple databases and database models into a single, unified view of the databases (see Figure 1.3). Put another way,

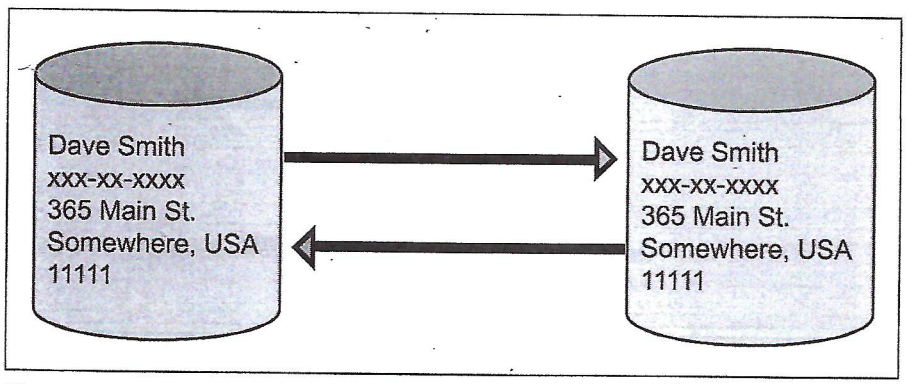


Figure 1.2 Database replication is the simple exchange of information between databases.

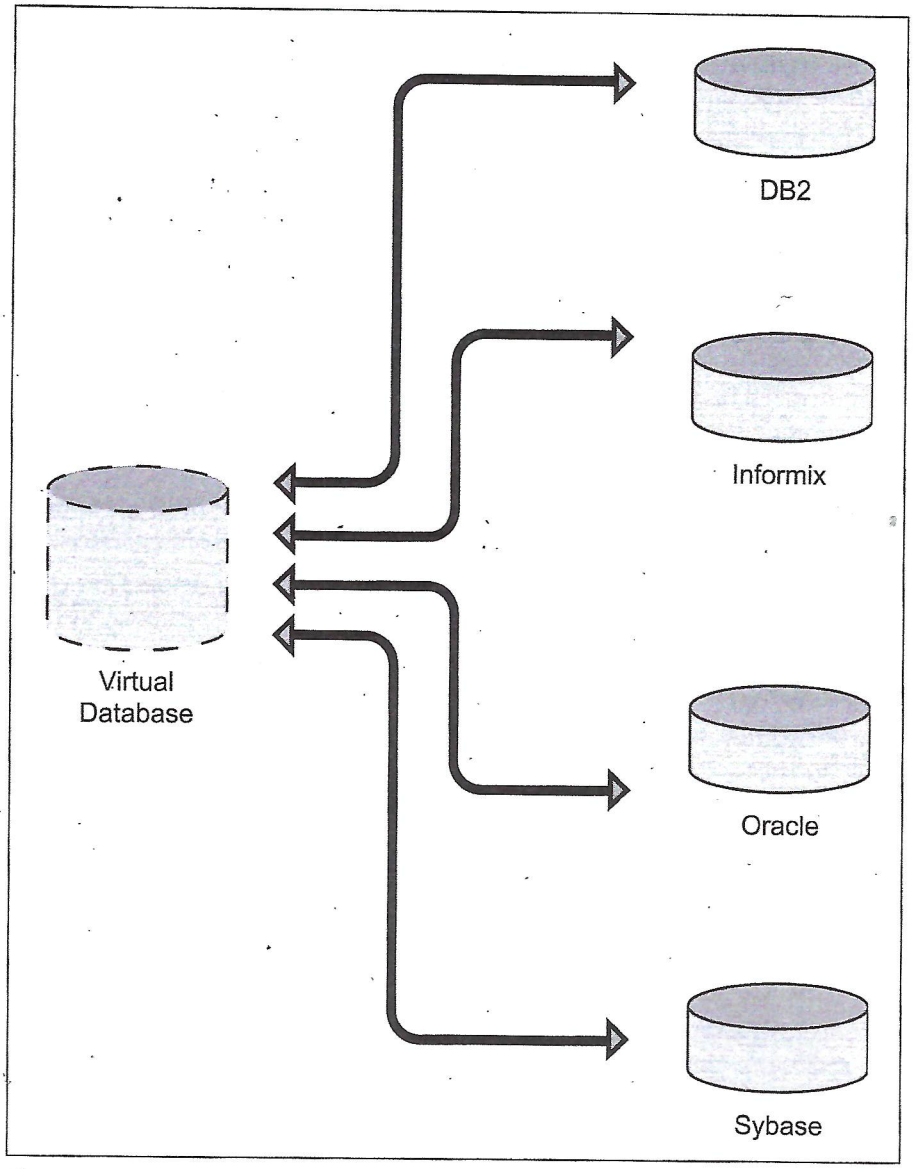


Figure 1.3 **Data federation allows many databases to appear as a single database.**

database federations are virtual enterprise databases that are comprised of many real, physical databases. While database federation has been around for some time, the solution set has been perfected only recently.

Database federation software places a layer of software (middleware) between the physical distributed databases and the applications that view the data. This layer connects to the back-end databases using available interfaces and maps the physical databases to a virtual database model that exists only in the software. The application uses this virtual database to access the required information. The database federation handles the collection and distribution of the data, as needed, to the physical databases.

The advantage of using this software is that it can bind many different data types into a unified model that supports information exchange.

Database federation allows access to any connected database in the enterprise through a single, well-defined interface. This is the most elegant solution to the data-oriented application integration problem. Unlike replication, this solution does not require changes to the source or target applications. Still, changes do have to be made at the application-oriented level to support federated database software. This is due to the fact that different interfaces are being used to access a different database model (the virtual database).

Interface Processing

Interface processing solutions use well-defined application interfaces to focus on the integration of both packaged and custom applications (see Figure 1.4). Currently, interest in integrating popular Enterprise Resource Planning (ERP) applications (e.g., SAP, PeopleSoft, and Oracle) has made this the most exciting application integration sector.

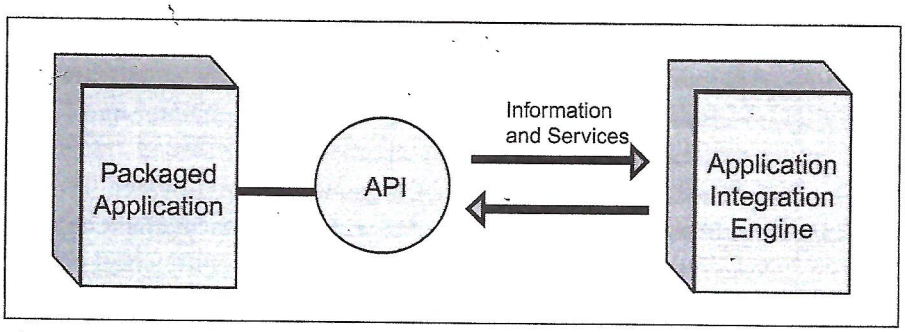


Figure 1.4 Interface processing externalizes information out of packaged applications through a well-defined API.

Integration brokers support application interface processing solutions by providing adapters to connect to as many custom or packaged applications as possible, externalizing information out of those applications through their open or, more often than not, proprietary interfaces. They also connect to technology solutions that include middleware and screen scrapers as points of integration.

The efficient integration of many different types of applications defines the primary advantage of using application integration-oriented products. In just days, it is possible to connect a SAP R/3 application to an Oracle application, with the application interface processing solution accounting for differences between schema, content, and application semantics by translating on the fly the information moving between the systems.

The downside to using application interface-oriented products is that there is little regard for business logic and methods within the source or target systems—logic and methods that may be relevant to a particular integration effort. In such a case, service-based solutions probably make the better choice. Ultimately, application interface processing technology will learn to share methods as well as information, perhaps by joining forces with service-based approaches.

Business Process Integration-Oriented

Simply put, business process integration-oriented products layer a set of easily defined and centrally managed processes on top of existing sets of processes contained within a set of enterprise applications (see Figure 1.5).

Packaged Application Interfaces: Information versus Services

While packaged application interfaces are primarily information oriented, there are a few that provide access to application services as well. These are hybrids. The best example of this is Business Application Programming Interfaces (BAPI) from SAP, but a few others also exist.

These interfaces allow you to invoke remote application services, such as processing a credit check or calculating shipping costs—processes that are more service than information oriented.

Packaged application interfaces, as you'll discover in Chapter 2, provide very different approaches to access both information and services. There are no standards for packaged application integration, even though the J2EE Connector Architect (JCA) is attempting to set some.

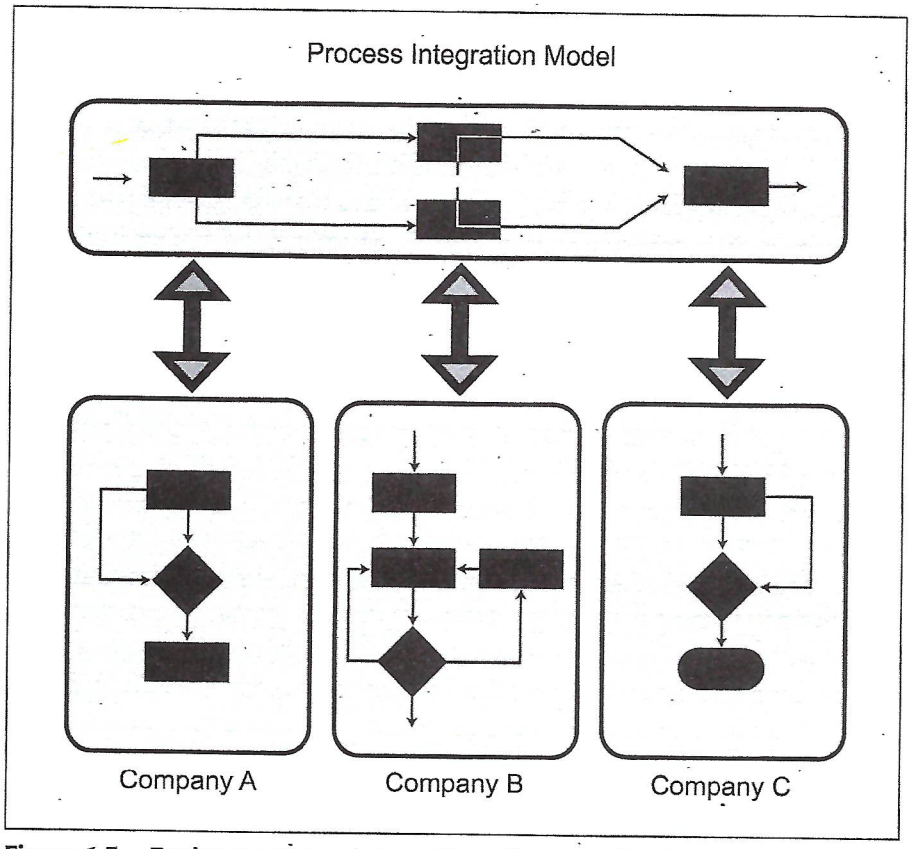


Figure 1.5 **Business process integration allows application integration architects to place a well-defined business process as the controlling entity, able to access both information and processes encapsulated in remote systems.**

Business process integration (BPI) is the science and mechanism of managing the movement of data, and the invocation of processes in the correct and proper order to support the management and execution of common processes that exist in and between applications. Business Process Integration-Oriented Application Integration (BPIOAI) provides another layer of easily defined and centrally managed processes that exist on top of an existing set of processes and data contained within a set of applications.

The goal is to bring together relevant processes found in an enterprise or trading community to obtain the maximum amount of value, while supporting the flow of information and control logic between these processes. These products

view the middleware, or the "plumbing," as a commodity and provide easy-to-use visual interfaces for binding these processes together.

In reality, business process integration is another layer of value resting upon existing application integration solutions, solutions that include integration servers, application servers, distributed objects, and other middleware layers. Business process integration offers a mechanism to bind disparate processes together and to create process-to-process solutions that automate tasks once performed manually.

However, by diminishing the importance of the plumbing, it is too easy to lose sight of the larger picture. In reality, no single application integration vendor has solved the plumbing issues. Ultimately, the solution to these issues will be delivered by a combination of business process integration and middleware vendors. That being the case, it is clear that the binding of middleware and process automation tools represents the future of application integration.

Business process integration is a strategy, as much as technology, which strengthens your organization's ability to interact with disparate applications by integrating entire business processes, both within and between enterprises. Indeed, business process integration delivers application integration by dealing with several organizations using various metadata, platforms, and processes. Thus, business process integration technology must be flexible, providing a translation layer between the source and target systems, and the business process integration engine.

There are many differences between more traditional application integration and business process integration.

- A single instance of business process integration typically spans many instances of traditional application integration.
- Application integration typically means the exchange of information between two or more systems without visibility into internal processes.
- Business process integration leads with a process model and moves information between applications in support of that model.
- Application integration is typically a tactical solution, motivated by the requirement for two or more applications to communicate.
- Business process integration is strategic, leveraging business rules to determine how systems should interact and better leverage the business value from each system through a common abstract business model.

BPIOAI views middleware, or the plumbing, as a commodity, with the ability to leverage both message-oriented and transactional middleware as points of

integration into any number of source or target systems. In fact, most integration servers and application servers are beginning to offer business process integration tools that support their middleware technology. Indeed, business process integration generally provides easy-to-use visual interfaces for binding these processes together and, along the way, creates visual BPIOAI.

While some may question the relevance of Business Process Integration-Oriented Application Integration, and even of application integration itself, I would argue that BPIOAI is the ultimate destination of application integration (acknowledging that we still have a long way to go to perfect the middleware). Despite current shortcomings, many application integration vendors are aggressively promoting BPIOAI as a vital component of their application integration technology package. In doing so, their strategy is clear—they are anxious to join the world of high-end, BPIOAI modeling tools. They hope that their application integration-enabled middleware, such as integration servers and application servers, will accomplish just that.

BPIOAI is best defined as applying appropriate rules, in an agreed-upon logical sequence, in order to pass information between participating systems, as well as visualize and share application-level processes, including the creation of a common abstract process that spans both internal and external systems. This definition holds true regardless of whether the business processes are automated or not. For example, processing an insurance claim and delivering a car to a customer are business events that can be automated with BPIOAI.

To this end, there are three main services that business process integration provides: the visualization of processes contained within all trading partner systems, interface abstraction, and the real-time measurement of business process performance.

By visualizing enterprise and cross-enterprise processes contained within trading partners, business managers are able to become involved in enterprise integration. The use of graphics and diagrams provides a powerful tool for communication and consensus building. Moreover, this approach provides a business-oriented view of the integration scenarios, with real-time integration with the enabling middleware or points of integration. This provides business analysts with the ability to make changes to the process model, implement it within the trading community, and typically not involve the respective IT departments.

Interface abstraction refers to the mapping of the business process integration model to physical system interfaces and the abstraction of both connectivity and system integration solutions from the business analyst. Business process

BPI by Example

There are three companies that participate in a trading community: Companies A, B, and C. Company A produces parts for skateboards, while Company B assembles and tests the skateboards, and finally, Company C sells the skateboards. Each has its own set of processes that are native to the respective company and its internal systems: a production system, an assembly system, and a sales system, respectively. Until now, automated integration has been nonexistent, and mail and fax serve communication needs between companies.

In order to integrate these applications, the trading community has decided to implement BPIOAI, defining a common process model that spans all companies and internal systems. This process model defines a sequence and logical order of events from the realization of consumer demand, the purchase of raw materials, the creation of the parts, the assembly of parts into a product, the testing of the product, and finally, the sale of the product to the ultimate consumer. This common model integrates with local systems by having visibility into their internal application processes, if possible, or perhaps through more primitive layers such as the database or application interface. What's important is that the common process model is able to produce events that are understood by the systems participating in the process, as well as react to events that the applications communicate back to the business process integration engine.

The use of a common process model that spans multiple companies for application integration provides many advantages, including:

- The ability to create a common, agreed-upon process between companies automating the integration of all information systems to react to business events such as increased consumer demand, material shortages, and quality problems in real time.
- The ability to monitor all aspects of the business and trading community to determine the current state of the process in real time.
- The ability to redefine the process at any given time in support of the business, and thus makes the process more efficient.
- The ability to hide the complexities of the local applications from the business users and to have the business user work with a common set of business semantics.

integration exists at the uppermost level in the application integration middleware stack. Those who use business process integration tools are able to view the world at a logical business level and are not limited by physical integration flows, interfaces, or adapters. What's more, the middleware mechanisms employed are

Walking Through a Process

Although each business process integration tool and project may take a slightly different approach, the internal process of interacting with the physical systems typically consists of the following set of events:

1. The source system that exists inside of a company posts an event to the business process integration engine; for example, a skateboard is sold.
2. The event is transformed, if required, so the event adheres to a standard set of business semantics and information processing mechanisms (synchronous versus asynchronous). This is going to be engine dependent, but there always has to be a common set of process semantics and information processing mechanisms defined at the engine level so the analyst can make sense of a business process that spans many types of applications, platforms, and databases.
3. The business process integration engine reacts to the event, once transformed, invoking other processes in other systems to support the execution of the common process model. For example, if a skateboard is sold, then send an order to the skateboard assembler, posting an event from the process engine to the assembler's target system (typically over the Internet).
4. Based on receiving that event, the local system reacts as per its internal processes and posts an event back to the process engines (say, when the skateboard is assembled).
5. The common process model sequences the master process, sending and receiving other events in support of the common process model. This is an ongoing activity, with information moving up to the process engine from the local systems, transformed if required, and down from the process engine to the local systems in support of the execution of the process model.

also abstracted and are not a concern of the business process analyst, as long as the common process model is interacting correctly with all source and target systems that exist within all companies.

Another way to view the process of creating a business process integration model is defining the hierarchy of processes within the trading community. This means that smaller subprocesses can be linked at the lower tier of integration or are native to the source or target systems. Building up from the lower-level processes to the higher-level processes, you may link the subprocesses into higher-level processes within the domain of the trading community.

The measurement of business process performance provides the business process integration with the ability to analyze a business in real time. By leveraging tight integration with the process model and the middleware, business analysts are able to gather business statistics in real time from the trading community; for example, the performance of a supplier in shipping goods to the plant, and the plant's ability to turn those raw materials into product.

Moreover, business process integration enables the technology user to track and direct each instance of a business process; for example, processing individual orders or medical insurance claims through a life cycle that may consume seconds, minutes, hours, days, or weeks. Finally, we need to measure and maintain contextual information for the duration of a process instance that spans many individual activities.

Indeed, the goal of BPIOAI, and of application integration in general, is to automate the data movement and process flow so that another layer of BPIOAI will exist over and above the processes encapsulated in existing systems. In other words, BPIOAI completes application integration, allowing the integration of systems, not only by sharing information readily, but also by managing the sharing of that information with easy-to-use tools.

In general, business process integration logic addresses only process flow and integration. It is not a traditional programming logic, such as processing a user interface, updating a database, or executing a transaction. Indeed, in most BPIOAI scenarios, the process logic is separated from the application logic. It functions solely to coordinate, or manage, the information flow between many source and target applications that exist within organizations.

Service-Oriented

Service-Oriented Application Integration (SOAI) allows applications to share common business logic or methods. This is accomplished either by defining

methods that can be shared, and therefore integrated, or by providing the infrastructure for such method sharing such as Web services (see Figure 1.6). Methods may be shared either by being hosted on a central server, by accessing them inter-application (e.g., distributed objects), or through standard Web services mechanisms, such as .NET.

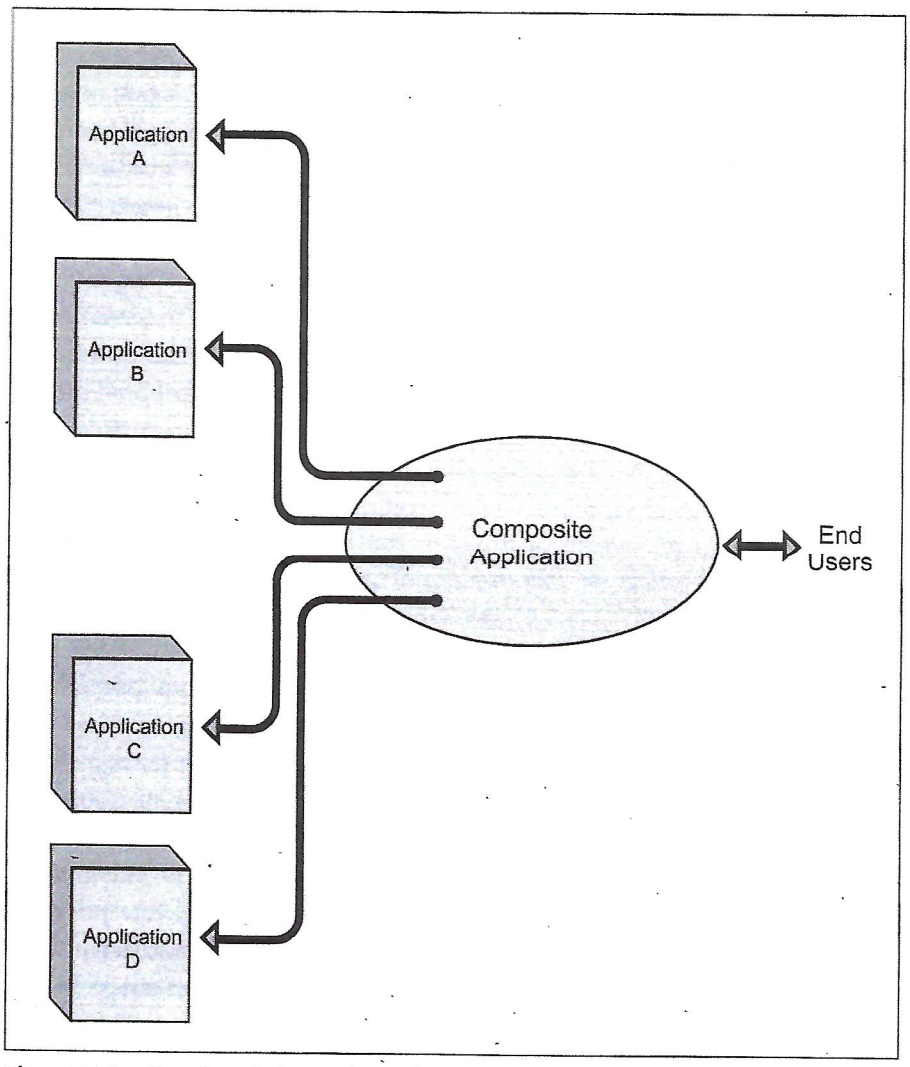


Figure 1.6 **Service-Oriented Application Integration provides mechanisms to create composite applications, leveraging services found in many remote systems.**

Attempts to share common processes have a long history, one that began more than ten years ago with the multitiered client/server—a set of shared services on a common server that provided the enterprise with the infrastructure for reuse and, now, for integration—and the distributed object movement. "Reusability" is a valuable objective. A common set of methods among enterprise applications invites reusability and, as a result, significantly reduces the need for redundant methods and/or applications.

While most methods exist for single-organization use, we are learning that there are times when it makes sense to share between organizations. In a new twist on the longstanding practice of reusability, we are now hoping to expand this sharing beyond intraenterprise—to trading partners, as well; for example, sharing a common logic to process credit requests from customers or to calculate shipping costs using a set of Web services.

Unfortunately, absolute reuse has yet to be achieved on the enterprise level. It is an even more distant goal between trading partners. The reasons for this failure are primarily political. They range from internal politics to the inability to select a consistent technology set. In most cases, the actual limit on reuse results directly from a lack of enterprise architecture and central control.

Utilizing the tools and techniques of application integration gives us the opportunity to learn how to share common methods. More than that, these tools and techniques create the infrastructure that can make such sharing a reality. By taking advantage of this opportunity, we are integrating applications so that information can be shared, even as we provide the infrastructure for the reuse of business logic.

Sounds great, doesn't it? The downside might give you pause, however. This "great-sounding" application integration solution also confronts us with the most invasive level of application integration, thus the most costly. This is no small matter if you're considering Web services, distributed objects, or transactional frameworks.

While IOAI generally does not require changes to either the source or target applications, SOAI requires that most, if not all, enterprise applications be changed in order to take advantage of the paradigm. Clearly, this downside makes SOAI a tough sell. However, it is applicable in many problem domains. You just need to make sure you leverage SOAI only when you need it.

Changing applications is a very expensive proposition. In addition to changing application logic, there is the need to test, integrate, and redeploy the

application within the enterprise—a process that often causes costs to spiral upward. This seems to be the case, no matter if you're approaching SOAI with older technologies such as Common Object Request Broker Architecture (CORBA), or new technologies such as .NET, the latest service-based architecture to come down the road.

Before embracing the invasiveness and expense of SOAI, enterprises must clearly understand both its opportunities and its risks. Only then can its value be evaluated objectively. The opportunity to share application services that are common to many applications—and therefore making it possible to integrate those applications—represents a tremendous benefit. However, that benefit comes with the very real risk that the expense of implementing SOAI will outpace its value.

Portal-Oriented

Portal-Oriented Application Integration (POAI) allows us to view a multitude of systems—both internal enterprise systems and external trading community systems—through a single-user interface or application. POAI benefits us by avoiding the back-end integration problem altogether; it adapts the user interface of each system to a common user interface (aggregated user interface)—most often a Web browser (see Figure 1.7). As a result, it integrates all participating systems through the browser, although the applications are not directly integrated within or between the enterprises.

While the other types of application integration are focused on the real-time exchange of information (or adherence to a common process model) between systems and companies, POAI is concerned with externalizing information out of a multitude of enterprise systems to a single application and interface. That's clearly an approach that goes against the notions of the other types of application integration, which are more real-time- and event-driven-oriented, and the inclusion in this book of POAI was somewhat of a judgment call.

However, application integration, while typically referring to the automated movement of information or the binding of processes between two or more applications, without the assistance of an end user, can clearly also occur at the user interface. Indeed, most examples of B2B information exchange today are also examples of POAI, with digital exchanges leading the way. Therefore, it's different, but it still belongs within the discussion of application integration.

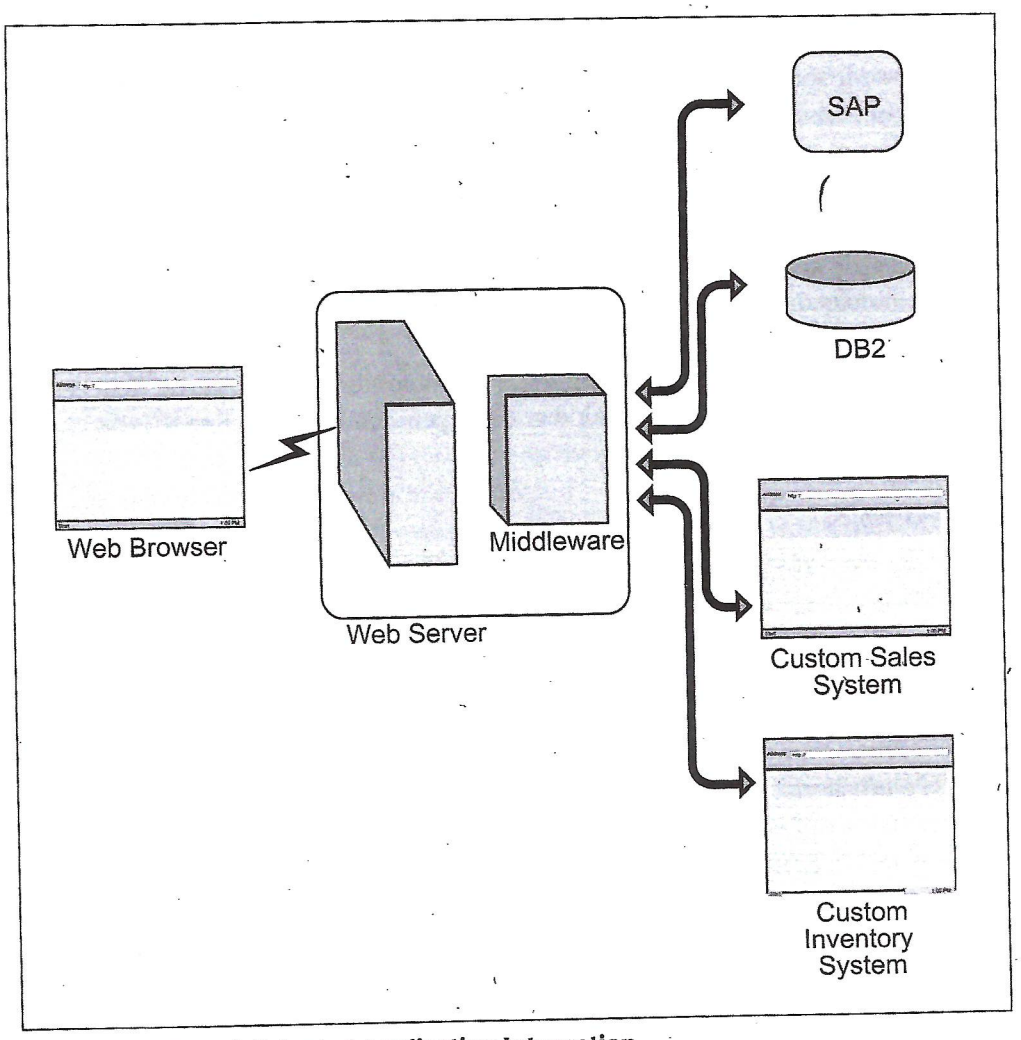


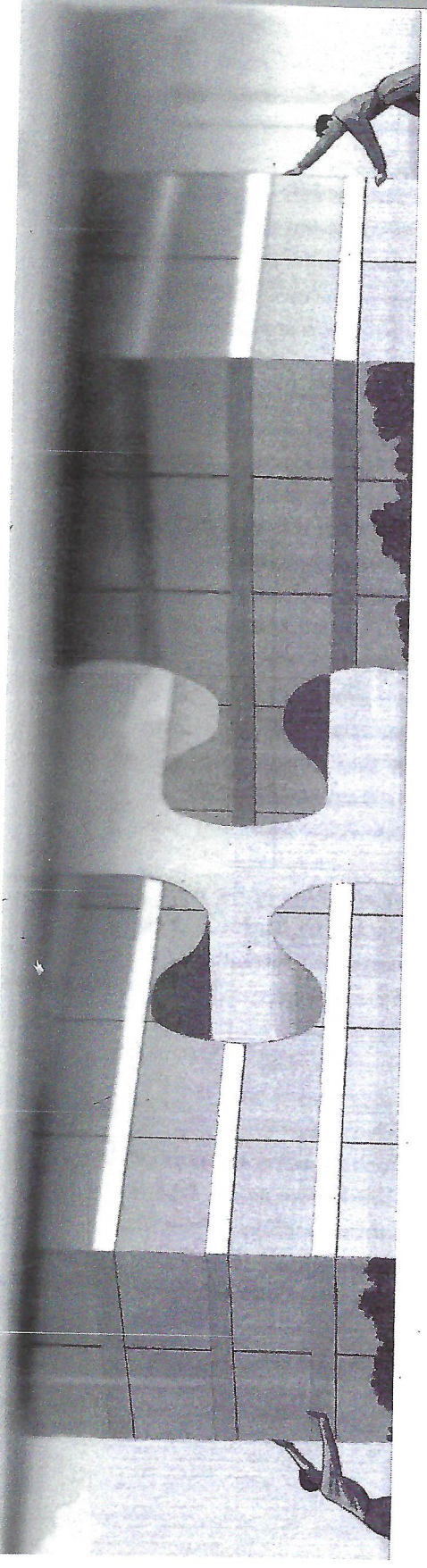
Figure 1.7 Portal-Oriented Application Integration.

Application Integration: Clearly the Future

Enter application integration—along with an opportunity to finally integrate all of these disparate systems with a minimum impact on the applications and the way an enterprise does business. Application integration provides a clear competitive advantage for most industries, an advantage that includes the ability to do business at light speed, along with the ability to satisfy customer demand in record time (by using automated processes instead of paper, faxes, and humans).

We are truly moving forward into a digital economy, where business runs within and between computers, where everything is automated, and where customers expect no less than instantaneous access to information.

What's important in meeting this goal is not just the application of technology to bind applications together, or externalize information to outside parties, but also the way in which you do it. Application integration is of little use if it's not quickly deployed, if it's not correct in operation, and if it's not able to adjust quickly as business needs change. This being the case, the way in which you approach your problem domain, the architecture you employ, and the technology you leverage has everything to do with the value of your application integration project going forward. Remember, application integration, if done right, is the strategic application of technology to provide an enterprise with the infrastructure required to handle most business events electronically, and in real time. In the end, that's what makes all the difference.



CHAPTER SIX

Middleware Basics

This is an introductory chapter, thus not for readers who already have a working knowledge of middleware. Having said that, it's probably not a bad idea for those of you who need a refresher to run through the basics again, including messaging, RPCs, distributed objects, and transaction servers. If you have no need to review these concepts, this can be a jumping-off point for the next several chapters, which cover middleware concepts and technology.

It is important to understand these concepts as we move forward in the book. Many of the more advanced and stylish notions, including Web services, are based on existing middleware models. For example, those who already understand distributed objects will have no problem understanding Web services.

I have devoted the first part of the book to application integration approaches and implementation. In the following chapters, we will concentrate on the technology that makes application integration possible: middleware and standards. This chapter provides an overview of middleware, setting the stage for the following chapters to describe several types of middleware technologies that may assist us in solving the application integration problem.

What Is Middleware?

There are many definitions of middleware. Ultimately, the definition that works best defines middleware in terms of its function. Middleware is a mechanism that allows one entity (application or database) to communicate with another entity, or entities. In other words:

> Middleware is any type of software that facilitates communication between two or more software systems.

Such a broad definition is necessary when you consider that middleware may be as simple as a raw communication pipe running between applications, such as Java's Remote Method Invocation (RMI), or as sophisticated as information-sharing and logic-execution mechanisms, such as Transaction Processing (TP) monitors.

Middleware's import role in the sharing of information means that its importance to the application integration solution is growing more evident. Once it was just a tool for moving information between systems existing within a single enterprise. Now we view middleware as a technology that allows us to move information between multiple enterprises. Middleware conceived and built exclusively for intraenterprise integration presents vendors with a significant challenge, given this new demand on the products.

It is a challenge vendors are anxious to meet. They are aware of the market benefit they will reap by supporting application integration. Of course, in their efforts to claim the marketplace, they are quicker to change their marketing message than they are to change their technology.

Middleware Models

There are two types of middleware models: logical and physical.

The logical middleware model depicts how information moves throughout the enterprise conceptually. In contrast, the physical middleware model depicts both the actual method of information movement and the technology employed.

In order to discuss the logical middleware model, we must first discuss point-to-point and many-to-many configurations, as well as synchronous versus asynchronous. Any examination of the physical middleware model requires a discussion of several messaging models.

Middleware can work in a point-to-point as well as many-to-many (including one-to-many) configuration. Each has its advantages and disadvantages.

Point-to-Point Middleware

Point-to-point middleware uses a simple pipe to allow one application to link to another application: Application A links to Application B. When Application A seeks to communicate with Application B, it simply "shouts down" the pipe using a procedure call, or message (see Figure 6.1).

When compared to other types of middleware, its inability to properly bind more than two applications limits point-to-point middleware. It also lacks any facility for middle-tier processing, such as the ability to house application logic, or the ability to change messages as they flow through the pipe.

There are many examples of point-to-point middleware, including Message-Oriented Middleware (MOM) products (such as MQSeries) and RPCs (such as DCE). While it is possible to link together more than two applications using traditional point-to-point middleware, doing so is generally not a good idea. The purpose of these products is to provide point-to-point solutions, primarily involving only a source and target application. Dealing with more than two applications invites too many complexities. Linking more than two applications with point-to-point middleware requires point-to-point links between all of the applications involved (see Figure 6.2).

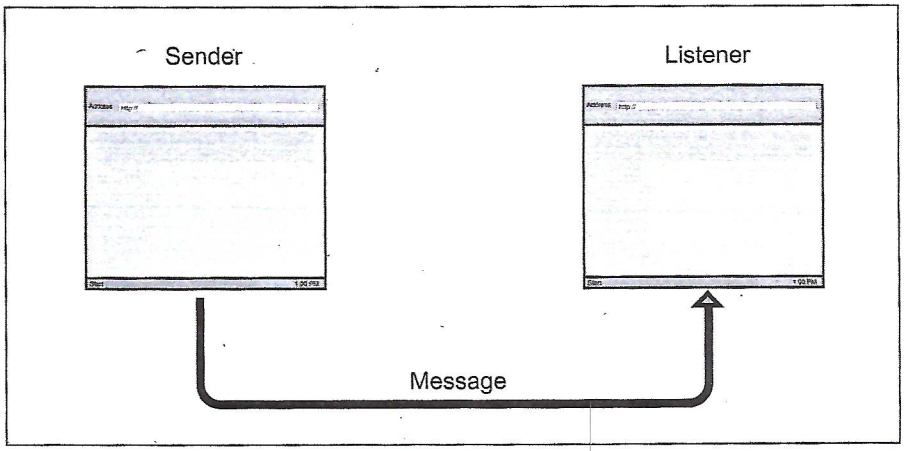


Figure 6.1 Point-to-point middleware.

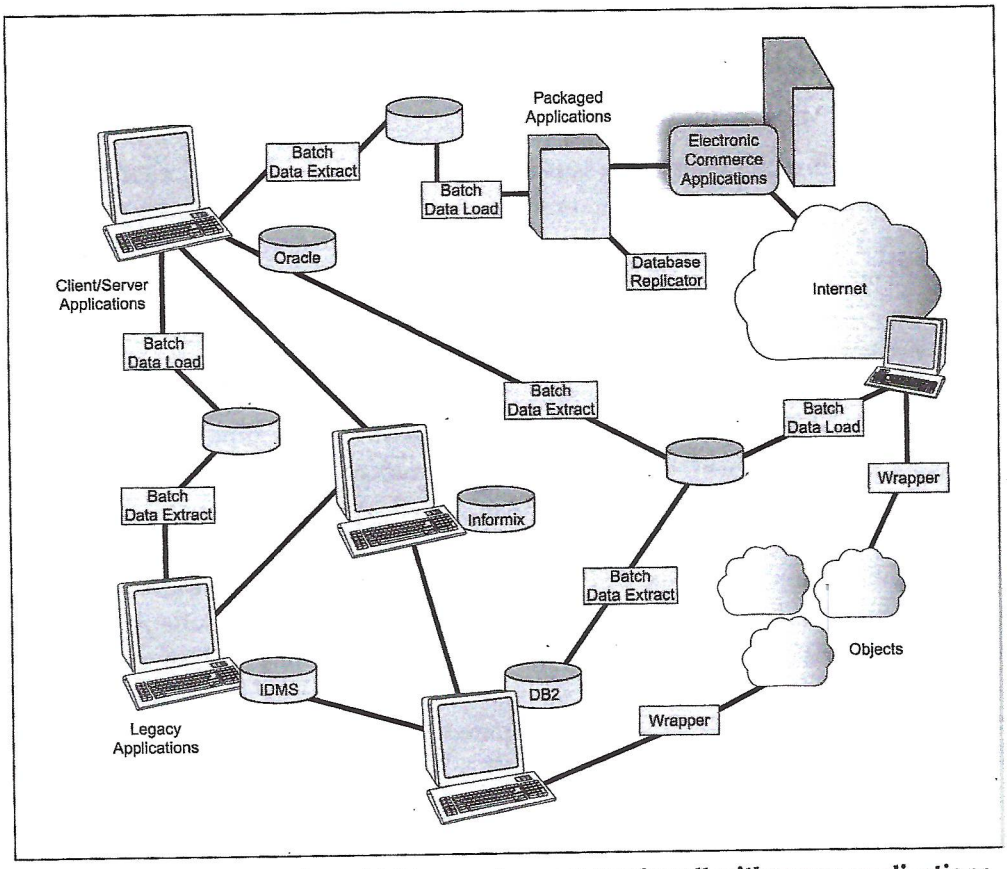


Figure 6.2 Point-to-point middleware does not work well with many applications.

The idea behind the point-to-point model should make it clear that it does not represent an effective application integration solution, given that most problem domains require linking many applications. Perhaps more important, in order to share information in this scenario, applications must be linked and information must be brokered through a shared, centralized server. In other words, sharing information requires a message broker, or transactional middleware.

However, as with all things, these disadvantages are somewhat offset by advantages. The great advantage of point-to-point middleware is its simplicity. Linking only one application to another frees the application integration architect and developer from dealing with the complexities of adapting to the differences between many source and target applications.

Many-to-Many Middleware

As its name implies, many-to-many middleware links many applications to many other applications. This capability makes it the best option for application integration. Being the "best option" makes it the obvious trend in middleware. In addition to this capability to link many-to-many, it is also the most powerful logical middleware model, in that it provides both flexibility and applicability to the application integration problem domain.

There are many examples of many-to-many middleware, including integration servers, transactional middleware (application servers and TP monitors), and even distributed objects. Basically, any type of middleware that can deal with more than two source or target applications at the same time is able to support this model (see Figure 6.3).

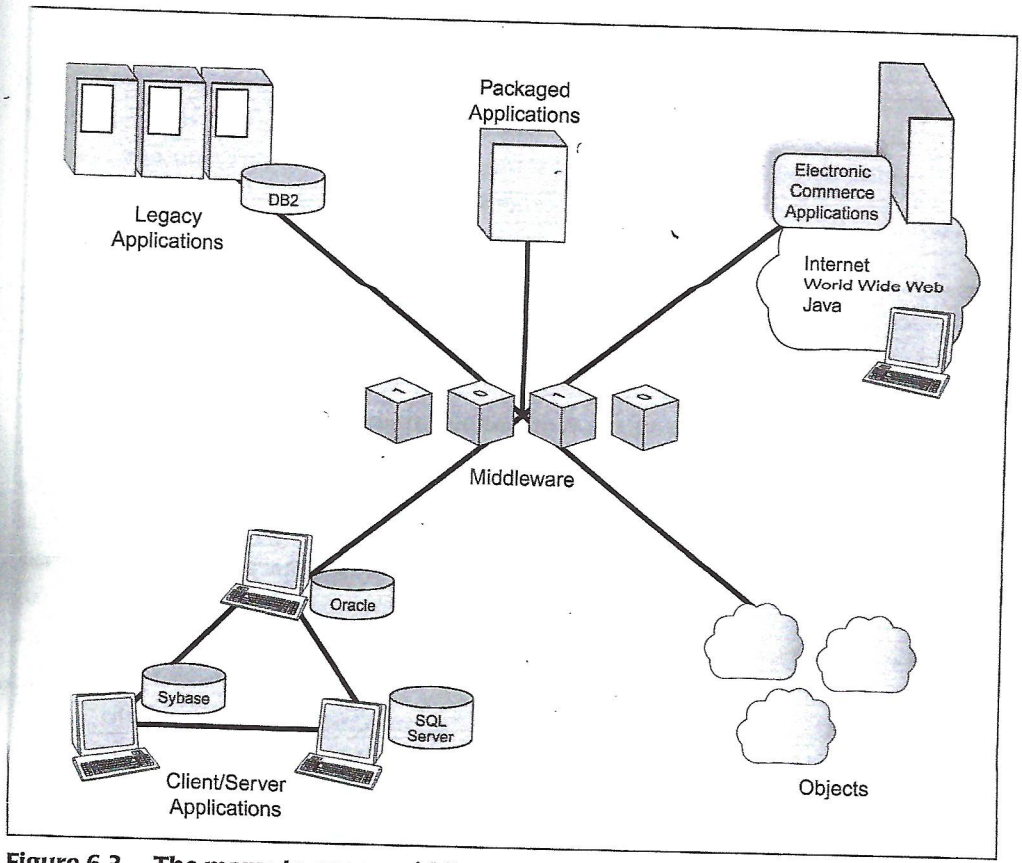


Figure 6.3 The many-to-many middleware model.

While the advantage of the point-to-point model is its simplicity, the disadvantage of the model is its complexity. Although the current generation of middleware products is becoming better at addressing the complexity of linking together so many systems, much work remains. After all, struggling with this complexity falls primarily on the shoulders of the developer.

Synchronous versus Asynchronous

As noted previously, middleware employs two types of communication mechanisms: asynchronous and synchronous.

Asynchronous middleware moves information between one or many applications in an asynchronous mode—that is, the middleware software can decouple itself from the source or target applications. The applications are not dependent on other connected applications for processing. The process that allows this to occur has the application(s) place a message in a queue and then go about its business, waiting for the responses at some later time from the other application(s).

The primary advantage of the asynchronous model is that the middleware will not block the application for processing. Moreover, because the middleware is decoupled from the application, the application can always continue processing, regardless of the state of the other applications.

In contrast, synchronous middleware is tightly coupled to applications. The applications are dependent on the middleware to process one or more function calls at a remote application. As a result, the calling application must halt processing to wait for the remote application to respond. We refer to this middleware as a "blocking" type of middleware.

The disadvantage of the synchronous model rests with the coupling of the application to the middleware and the remote application. Because the application is dependent on the middleware, problems with middleware—such as network or remote server problems—stop the application from processing. In addition, synchronous middleware eats up bandwidth because several calls must be made across the network in support of a synchronous function call. This disadvantage, and its implications, makes it clear that the asynchronous model is the better application integration solution.

Connection-Oriented and Connectionless

Connection-oriented communication means that two parties connect, exchange messages, and then disconnect. Typically this is a synchronous process, but it can also be asynchronous. Connectionless communication means that the calling

program does not enter into a connection with the target process. The receiving application simply acts on the request, responding if required.

Direct Communication

In direct communication, the middleware layer accepts the message from the calling program and passes it directly to the remote program. Either direct or queued communication is used with synchronous processing. Direct is usually synchronous in nature, and queued is usually asynchronous. Most RPC-enabled middleware uses the direct communication model.

Queued Communication

Queued communication generally requires a queue manager to place a message in a queue. The remote application then retrieves the message—either shortly after it has been sent, or at any time in the future (barring time-out restrictions). If the calling application requires a response (such as a verification message or data), the information flows back through the queuing mechanism (see Figure 6.4). Most MOM products use queued communication.

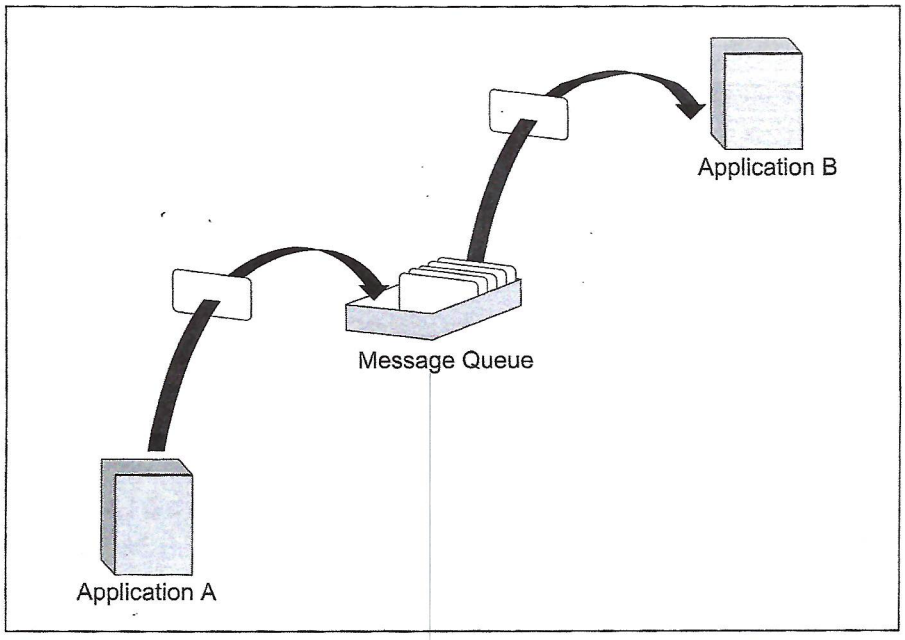


Figure 6.4 Queued communication.

The advantage of the queuing communication model over direct communication rests with the fact that the remote program does not need to be active for the calling program to send a message to it. What's more, queuing communication middleware typically does not block either the calling or the remote programs from proceeding with processing.

Publish/Subscribe

Publish/subscribe (pub/sub) frees an application from the need to understand anything about the target application. All it has to do is send the information it desires to share to a destination contained within the pub/sub engine, or broker. The broker then redistributes the information to any interested applications. For example, if a financial application wishes to make all accounts receivable information available to other applications, it would inform the pub/sub engine. The engine would then make it known that this information was available, and any application could subscribe to that topic to obtain accounts receivable information.

In this scenario, the publisher is the provider of the information. Publishers supply information about a topic, but they don't need to understand anything about the applications that are interested in the information (see Figure 6.5). The subscriber is the recipient, or consumer, of the information. The publisher specifies a topic when it publishes the information. The subscriber specifies a topic that they are interested in. In this scenario, the subscriber receives only accounts receivable information.

Request Response

The request response model is exactly what its name implies. A request is made to an application using request response middleware, and it responds to the request (see Figure 6.6). Examples of request and response middleware include

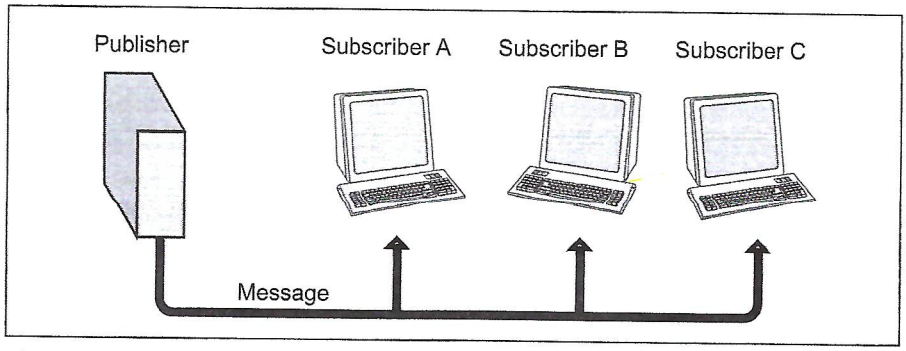


Figure 6.5 The publish and subscribe model.

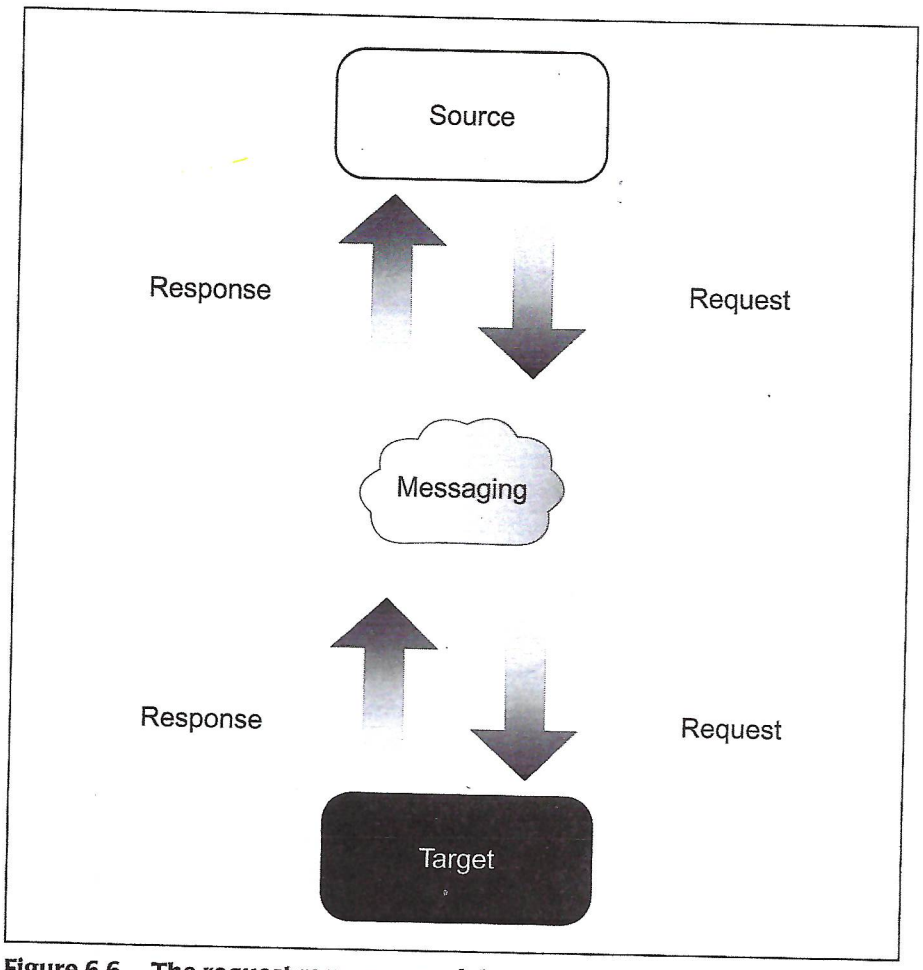


Figure 6.6 The request response model.

any middleware that can facilitate a response from a request between applications, such as integration servers or application servers.

Fire and Forget

The fire and forget model allows the middleware user to "fire off" a message, and then "forget" about it, without worrying about who receives it, or even if the message is ever received. This is another example of an asynchronous approach. The purpose of fire and forget is to allow a source or target application to broadcast specific types of messages to multiple recipients, bypassing auditing and response features. It also allows central servers to fire off messages.

Types of Middleware

The evolution of middleware is changing many of the identifying features that had once made categorizing middleware such a straightforward task. For example, many MOM products now perform publish and subscribe tasks, provide transactional features, and host application logic. The challenge of appropriately categorizing such a product should be plainly evident.

However, even as we acknowledge the difficulty in categorizing middleware, we note that several types of middleware continue to solve particular types of problems. For the purposes of our application integration discussion, we will describe RPCs, MOM, distributed objects, database-oriented middleware, transactional middleware (including TP monitors and application servers), and integration servers (see Figure 6.7).

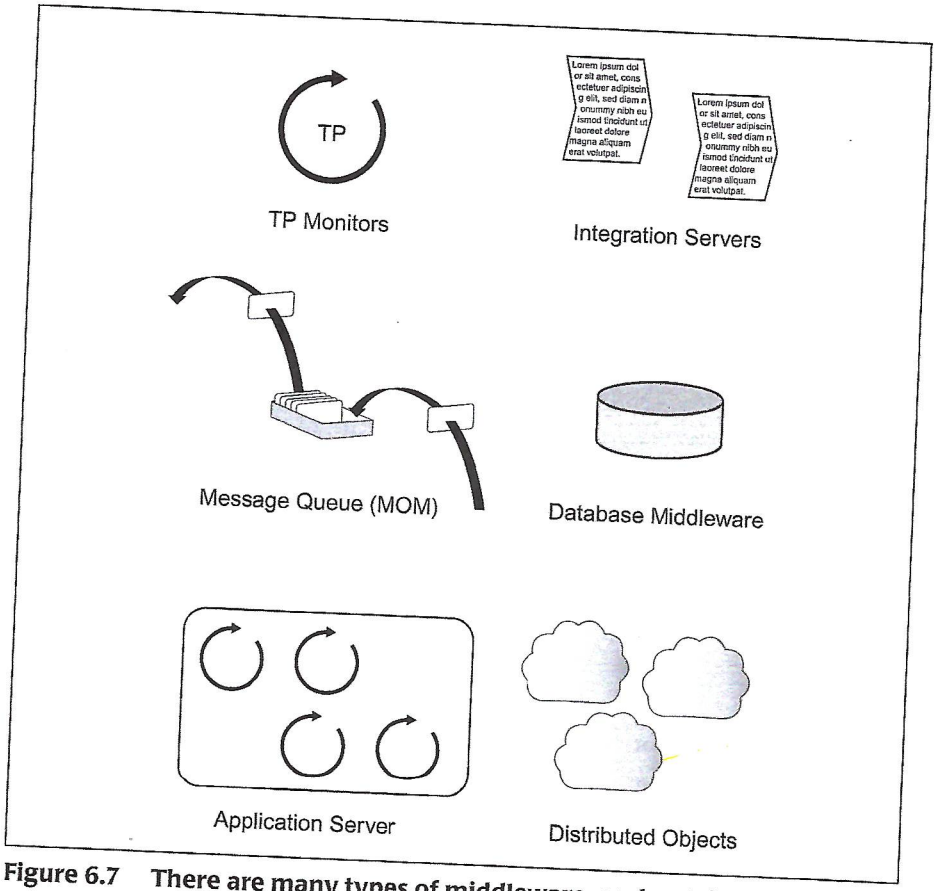


Figure 6.7 **There are many types of middleware, each solving its own set of problems.**