4

Integration Technologies

Chapter 3 provided an introduction to integration concepts, described integration roles within the enterprise, and defined integration methods providing access to function and data at various levels of the application design stack. In this chapter, we introduce the tools of the trade: those software systems, platforms, tools, and technologies used to connect systems in the enterprise in useful ways.

Integration Technology Landscape

Integration tools provide the foundation upon which higher-level functionality can be provided. They include the software components and technologies that connect systems in the enterprise in useful ways. Historically, integration tools began as basic hardware and software adapters providing physical connectivity to systems. These adapters have evolved with ever-increasing functionality to provide scalability, ease of deployment, and management of connections. In addition, new integration technology has evolved, layered on top of adapters, providing increasing levels of complex operations (Figure 4.1).

As we investigate integration tools, it is useful to recognize that there are two vantage points from which we can view these tools. First, if a

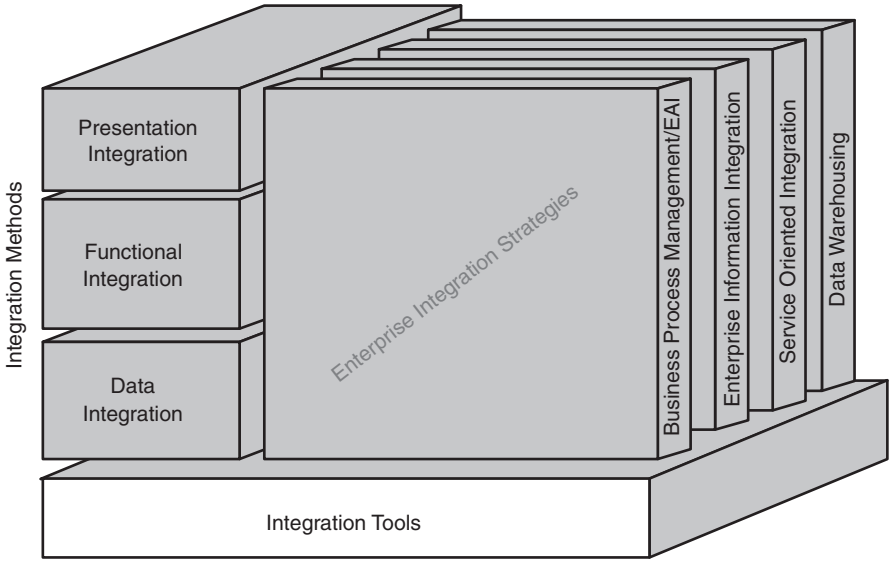

Figure 4.1. Integration tools form the foundation of higher level functions.

software component can be called from other software components, then it must have a mechanism, called an application interface, that allows this interaction. Second, if a software component needs to access other processing resources, whether they are databases or other applications, it must have an adapter that lets it access that resource. Figure 4.2 shows the positioning of application interfaces and adapters in a software technology stack.

It turns out that most integration technologies can act as either an application interface or an adapter. For example, a software component may want to call a service using a Web service interface. The Web service in this case is an adapter from the perspective of the software component. But the software component may provide services to other components using a Web service interface. In this case, the Web service technology is used as an application interface. One developer's adapter is another developer's application interface. As you investigate integration tools in the following sections, it will be helpful to keep this in mind. Consider where a particular technology fits relative to your experiences, then consider its use on the opposite side of the stack.

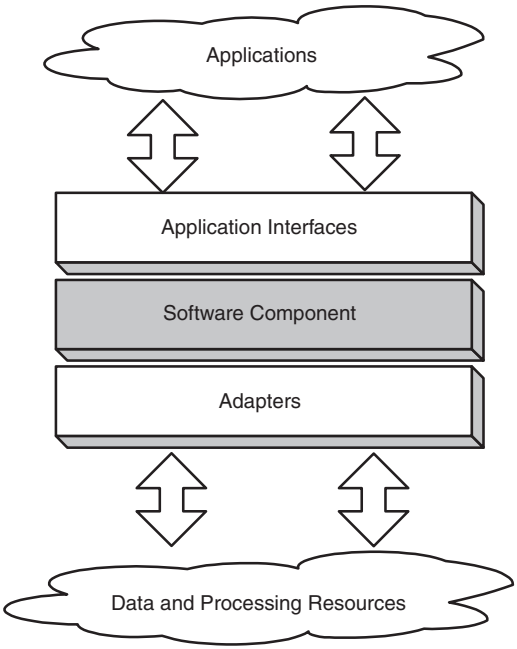

Figure 4.2. Application interfaces and adapters.

Middleware

The term "middleware" is used in many different contexts in the IT industry today. The Internet 2 project *(www.internet2.edu)* defines middleware as directory and security services, and other collaboration tools that are common tasks that should be implemented once, then used as services by other applications and users. In the context of customer service call center technology, middleware is software that relates data from computers to actual calls received at the service center, so that customer information is presented on screens simultaneously with the voice connection. In our context, that of integration, middleware refers to software that assists in the movement of data to satisfy a particular purpose. Figure 4.3 shows a software stack of integration tools and highlights the fact that middleware is the medium by which applications interact with data sources and other applications. In the following sections, we present a landscape of middleware tools and describe the technologies in common use today.

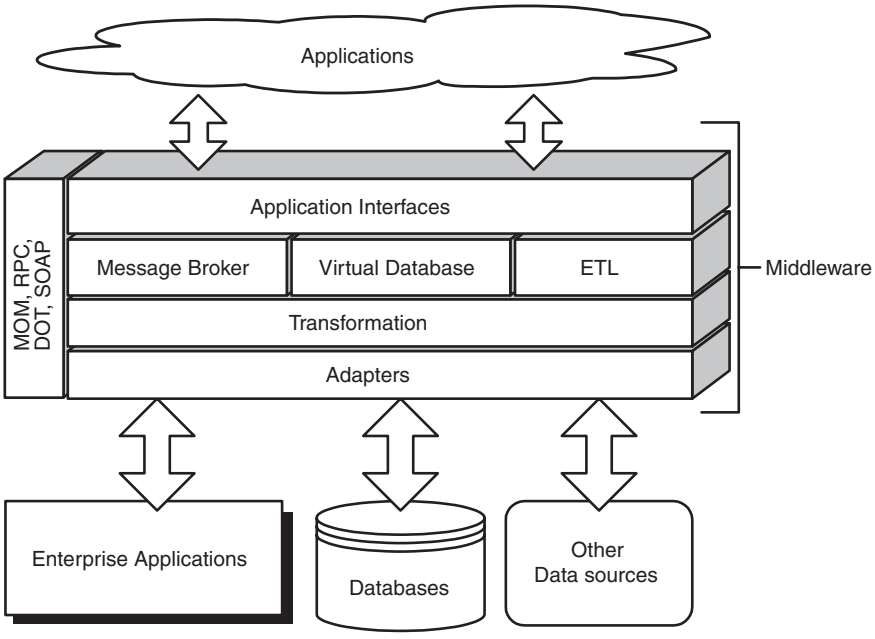

Figure 4.3. Middleware landscape.

Adapters

Each data source within the enterprise, whether it is a database or an application, is accessed by integration software through a software component called an adapter. Some systems do not have prewritten adapters available. If these systems are to participate in integration activities, an adapter will need to be written first.

Most modern packaged systems and databases have adapters available, written by the vendor of the package or database, or by a third-party software vendor. The following sections describe various types of data sources and adapters used to access the data.

Database Access Technology

Much of the research work in database design over the last thirty years has been focused on the separation of program code from the definition of data that the program works on. The prevalent theory is that pro-

grams should be written independent of any specific file structure used to contain the data. Instead of manipulating file structures directly, a program manipulates a model of the data. Exactly how the data is formatted and stored on an actual file system is not an issue the programmer is concerned with. Thus, the programmer's job is less complex. The idea that a program manipulates a model of the data rather than a physical structure is called data abstraction. The benefit of the data abstraction layer is that a program written against a data model can use a different storage technology simply by implementing the model on the new storage platform. In reality, new storage platforms are not swapped out often, but the practical benefit of data independence is that selection of a standard data model, such as the relational model, means there will be ample support on a broad range of platforms. The following sections describe various data models and methods of data access for each.

Relational Data

The relational model is by far the most widespread data model in use. First proposed by Coad in 1970, enterprise-class relational database management systems (RDBMSs) are available from vendors such as IBM, Microsoft, Oracle, and Sybase. Inexpensive and even free RDBMSs are available from many smaller vendors and organizations. The popularity of the relational model is due to its simplicity and wide availability of tools. All data in the relational model is stored in tables, consisting of records called rows. Each row contains named fields called columns, each containing typed data, such as a number, a date, or a character string. The names "row" and "column" are derived from the visual representation of a table as shown in Figure 4.4.

Generally, each table contains one or more columns called the key, whose value uniquely identifies a row within the table. For example, in

Table: Product

Columns:	ProdNum	Product	Price
	1	Pencil	0.79
	2	Pen	1.29

Figure 4.4. Example table "Product."

Figure 4.4, the ProdNum field is the key for the product table. The key can be used to relate one table to another. Let's assume we are modeling a structure representing an order, one part of which is an order item. We might have a table called "Item" (see Figure 4.5). The two tables are related based on the ProdNum column. An operation called a join lets us combine this data into a new grouping. In this example, we join the Product and Item tables together based on the ProdNum field, creating a new logical set of rows, shown in Figure 4.6.

The relational model defines separate mechanisms to define table structures and to operate on the data contained in the tables. Data Definition Language (DDL) is used to define tables in the RDBMSs. In addition to creating table structures, DDL usually includes a way to define rules for how the data structures behave and interoperate. DDL allows a database developer to define a column as a key, ensure uniqueness of records within a table, and ensure that tables defined to have a specific relation will contain instances of records that exhibit the specified relation. The latter, called referential integrity, allows defining a rule requiring that an order item inserted to a table will only succeed if the order of which it is a part is already in the database. This is an important aspect of database design, because the ability to declare such rules in the data-

Table: Item

Columns:	OrderNum	ProdNum	Quantity
	1001	1	10
	1001	2	25

Figure 4.5. Example table "Item."

Relation: Join Product and Item on column ProdNum

Columns:	OrderNum	ProdNum	Quantity	Product	Price
	1001	1	10	Pencil	0.79
	1001	2	25	Pen	1.29

Figure 4.6. Sample join operation results.

base definition eliminates the need for application developers to program these types of rules. This reduces the programming task and ensures consistent treatment of the data across many applications.

Data Manipulation Language (DML) is used to perform operations on the data, such as the join operation illustrated above. The most common DML is Structured Query Language (SQL), which provides a standard command format to select, update, and insert data into the database. While SQL has been standardized, it is important to note that SQL syntax varies among the major RDBMSs vendors, so SQL written for one RDBMSs may or may not work with another vendor's RDBMSs. DDL exhibits an even greater variance across vendor products.

In addition to manipulating data in the database, most RDBMSs provide a way to create scripted functions called stored procedures, which can be executed by a client application. A stored procedure allows a database designer to encapsulate programmatic rules or other logic associated with the data.

RDBMSs access is available in two categories of interfaces: custom interfaces provided by a specific vendor, often referred to as a "native interface," and common interfaces designed to work across vendor products. A native interface is a programmatic interface, generally available in multiple languages such as C++ and Java, that lets a developer programmatically manipulate the database to read and write data. The actual manipulation usually is achieved using SQL, which is sent to the database using the supplied interface. A mechanism is provided to read rows of data back from the database. The most popular common interfaces are described in the following sections.

ODBC

Open Database Connectivity, originally created by Microsoft, but now widely adopted across a wide range of platforms, defines a standard interface into a RDBMSs. ODBC enjoys widespread use in applications that must access a RDBMSs. ODBC has also been implemented for nonrelational sources, such as spreadsheets and structure text files, by making these sources appear to conform to a relational model with rows and columns of data.

OLE DB

OLE DB is Microsoft's successor to ODBC technology. Built on Microsoft's widely used Component Object Model (COM) technology, it provides a common interface into both relational and nonrelational

data sources. The technology includes libraries to assist developers in creating drivers, called providers, that expose custom data sources as "tables" that can be accessed using the OLE DB interface. The result is a Microsoft-standard application interface into data sources, where applications can potentially access not only relational data sources, but nonrelational and custom data sources as well, given availability of the provider software. To date, various vendors have fulfilled this vision by providing OLE DB interfaces into many different data sources, including relational databases, flat text files, legacy data sources such as mainframe IMS (Information Management System), ISAM (Indexed Sequential Access Method), VSAM (Virtual Sequential Access Method), and many other data stores.

Interestingly, most OLE DB vendors for nonrelational sources provide the same capability using ODBC as well. While Microsoft pointed to the need to support nonrelational sources as part of the rationale for OLE DB, it turns out that vendors proved ODBC is an adequate vehicle for this same capability.

JDBC

Java Database Connectivity is the standard library interface into RDBMSs for programs written using Java. Similar to ODBC in concept, JDBC provides a standard mechanism for Java programs to communicate with relational data stores. As with ODBC and OLE DB, vendors have created JDBC interfaces into virtually any type of tabular data, including nonrelational data stores.

ADO

ActiveX Data Objects (ADO) is a Microsoft product that provides a set of COM objects that utilize OLE DB. The COM objects provide full database connectivity, including the ability to manage connections, execute queries, and perform database management functions (table creation, modification, deletion, etc.).

ADO.NET

With the release of Microsoft's .NET technology, ADO was completely revamped for modeling more complex data. ADO.NET supports access to databases, OLE DB data sources, and XML. While ADO and its underlying technology, OLE DB, included limited provisions for hierarchical data models, ADO.NET includes a new model with a firm foundation in hierarchical data representation. This new object model,

called the DataSet, can represent hierarchical business objects in their full richness, complexity, and completeness. Once a DataSet is fetched by an application, it can be presented in a graphical user interface, manipulated, and sent to other processing resources where additional processing may occur. Finally, the DataSet can be used to update back-end systems.

Object-Oriented Data

Addressing specific needs of the object-oriented development community, object-oriented database management systems (OODBMS) at one time were thought to be the up and coming challenger to RDBMSs as the reigning champion of the data storage world. This was in large part due to the overwhelming popularity and adoption of object-oriented programming languages such as C++, Java, and more recently, C# (pronounced "C-Sharp"). OODBMS have a conceptual advantage in that they store objects natively from the programmer's viewpoint. In object-oriented data (OOD), objects defined in the design model real-world objects, such as a customer, order, or invoice. A real-world object is generally too complex to be stored in a relational table. Instead, the relational model requires transformation between the object and the relational view, usually requiring multiple tables to store a single business object. This transformation activity is eased by a tools category called Object-Relational Mapping.

Although the idea behind OODBMS is sound, early implementations suffered from immaturity of the technology, such as poor performance and lack of experienced engineers. The result was that many projects failed under the spotlight, leading to a poor reputation in the early days of the technology. Today, OODBMS are still in use but are relegated to niche areas where organizations find the complexity of their object models have a better fit in the OODBMS, rather than having to convert between an object representation and a relational model.

Multidimensional Data

Multi-dimensional databases (MDDB) are databases designed specifically to support the needs of business managers and analysts seeking to derive meaningful business information from the huge amount of data in the enterprise. MDDB are the data storage layer supporting on-line analytical processing (OLAP), often called "data mining."

MDDB supports the definition of different views of the data in a model called a cube. Dimensions of the cube represent characteristics of the data that may be queried and summarized. In a typical enterprise, one might find an MDDB containing dimensions labeled "products", "sales region," and "sales period," Using an OLAP tool, a user could easily query for total sales for a particular product across the United States. The OLAP tool then makes it easy to generate reports to visually communicate a particular aspect of the business.

The MDDB optimizes the storage of the data to make business-oriented queries perform well. Large computing resources would be required to execute many of these queries if the data were stored in a format using a different technology, such as an RDBMSs.

XML Databases

An XML database stores XML structures natively, providing operations similar to those provided by an RDBMSs. Operations typically include queries, updates, administrative functions, security, and transaction management. One of the biggest advantages of an XML database is the ability to operate with structures that closely resemble business objects in the enterprise, while enabling XML advantages of human readability, interoperability, vendor independence and others. A business object is an information set and its defined operations, which can be encoded into a programming language. A typical IT system operates on business objects such as a customer, a product, an order, and an invoice.

Business objects are by their nature hierarchical. For example, a customer business object may include customer profile information (address, contact information, etc.), a service profile (a list of services the customer is signed up for), and order history. Such a structure cannot be stored in a simple relational table, because of the two-dimensional nature of a table. To store such a structure in an RDBMSs requires a transformation from the complex, hierarchical structure to rows and columns in a relational table. This transformation from XML to a set of relational tables is called shredding, where the structure is sliced and diced, decomposed, and fit into multiple tables with relations to represent the original structure. A similar process takes place when business objects in a programming language are being converted to and from the relational model. This process is called object-relational mapping.

The shredding process adversely affects performance, so the XML database has an advantage in this area. Performance of searching op-

erations varies widely between XML databases and RDBMSs, and both XML and relational vendors claim better performance for selected sets of search and retrieval cases.

Legacy Databases

Many legacy applications were written decades ago and use older data models for storing business information. Some of the data models still in use are described in the following sections.

Mainframe File Access

Many of the world's largest companies continue to use IBM mainframe-based applications to run their businesses. IBM's mainframe operating system, MVS (Multiple Virtual Storage, "virtual storage" referring to its implementation of virtual memory), now called OS/390, defined various file-based record access methods that many applications use today. These methods allow COBOL and other applications to manage records on the file system. Early applications used ISAM. ISAM extends a basic sequential access method where applications access records from a file in sequence but cannot select a record randomly from an arbitrary location in the file. The addition of indexes to the basic method allowed designers to set up multiple indexes on any one file or data set. Thus, an application can scan records in an appropriate order, for example, by customer name or by sales region.

IBM's VSAM extended ISAM by allowing retrieval of a record using a key (for example, a customer ID) instead of requiring a scan of records based on an index. While ISAM and VSAM are still in widespread use, IBM now encourages its customers to use DB2, an RDBMSs, as the preferred data model for applications.

Hierarchical Data Model

The hierarchical data model defines data structures as hierarchies. A record can contain any number of field values, and records with identical field definitions are grouped together as record types. Record types are similar to RDBMSs tables, and individual records are similar to RDBMSs rows. A hierarchical structure can be constructed by introducing a parent-child relationship between record types. A link is created, designating one record type as the parent and the other record type as a child. Links can be built up using many record types, constructing a logical tree that represents a hierarchical structure.

The hierarchical data model was first implemented by IBM for its IMS database product, an offshoot of IBM's work with NASA on the Apollo project. According to IBM, IMS has been successfully deployed in a high percentage of the world's top companies. IBM continues to sell and support IMS as a high-speed, transactional database, with interfaces now provided from both Java- and XML-based applications.

Network Data Model

The network data model was proposed in 1971 by the Data Base Task Group (DBTG) of the "Conference on Data Systems Language" (CODASYL), the same organization responsible for defining the COBOL programming language. As with the hierarchical model, a record in the network data model is a group of fields, with the record layout being defined by a record type, similar to a database table definition. Record types are related using a concept called a set type. A set type determines the relationships between record types; for example, a set type called order may include a customer record type and product record type, resulting in an order structure that contains customer information and a list of products being ordered. In contrast to the hierarchical data model, structures in the network data model are not required to be strictly hierarchical. Rather than requiring that a record type have at most one "parent," a record type can have multiple parents and multiple children. Whereas the hierarchical model supports one-to-many relationships through the parent-child relation, the network data model supports many-to-many relationships, providing potential for modeling more complex real-world information structures.

The most popular implementation of the network model is IDMS, owned by Computer Associates International (CA). According to CA, CA-IDMS is the database behind mission-critical applications installed at thousands of sites around the world. CA provides adapter technology using ODBC, allowing access from any ODBC client application.

Presentation Integration Tools

Presentation-level integration tools provide the ability to programmatically manipulate the screens of an application. This technology is usually called screen scraping, because data is scanned or "scraped" from a software version of a display. Most popular among these is a class of tools that interface to mainframe applications, acting as software-based terminals such as IBM 3270 or 5250 terminals. Since so much valuable

functionality is built into legacy applications, it is often convenient to reuse this functionality. Conceptually, this makes a lot of sense. Integrating at the presentation level has the benefit of preserving any business rules coded into the application. As mentioned in the previous chapter, the drawback of this method is that these interfaces are brittle. Any small change in the user interface makes the mapping to screen fields obsolete. Also, original designers did not intend these screens to be accessed programmatically for the purposes of integration, so the data available in any one screen may not be the complete set of data required. In fact, significant automated navigation may be required to collect the required data, creating a potential performance problem. Nevertheless, a designer may find a need to use this type of tool, particularly when no other mechanism exists to access required data.

Mainframe applications can be screen scraped using tools provided by many terminal emulation vendors. Terminal emulation software is a program that makes a computer or browser behave as a terminal for the purposes of running an application that requires a specific type of terminal. IBM has created a standard interface called High Level Language Application Program Interface (HLLAPI) and extended HLLAPI (EHLLAPI), which defines the programmatic interface to mainframes. These standards are supported by most of the terminal emulation providers.

Tools for integrating Web pages using a similar technology are also available. Web page screen scraping is generally easier, since free tools exist to treat Web pages programmatically. For example, Microsoft defines a Document Object Model (DOM) for HTML that lets a program navigate through the various elements in a Web page, picking data from fields, setting values for fields, submitting forms, and otherwise manipulating the Web page. Using this technology, a developer can fetch a Web page from a site, pull off needed information, and possibly submit the page back to the site, all the while fooling the site into believing it is talking to a normal browser with a human operator. Note that most sites have usage policies that prevent the automated capture and reuse of information published on the site.

Web page screen scraping suffers from the same drawbacks as terminal screen scraping. Since mainframe applications are generally mature, long-lived applications that have provided mission-critical functions for many years, changes are usually tightly controlled and methodically implemented over a significant time period. Typical release cycles usually enable only one or two releases per year. In contrast, Web pages tend to

change much more frequently, so the problem of changing pages is more acute and risky than with mainframe application screen scraping.

Application Interfaces

Packaged applications such as SAP, PeopleSoft, Siebel, Baan, and others generally provide an interface for applications to access the information managed by the application. In general, the application vendors have increasingly come to realize that their products will be more successful if enterprises are able to integrate the application with other systems, and have reacted by opening up these systems. Most provide a variety of interfaces to access both the data in the application and the functionality provided by the application. Often, a vendor will provide the interfaces through multiple technologies such as Java API, COM, XML, and Web services, providing flexibility by allowing the end user to use the technology of choice. While the interfaces are generally well published, the complexity of the applications themselves and the underlying data structures means that to make adequate use of these interfaces requires a great deal of understanding of the applications. Vendors generally provide documentation, training, and support that is invaluable for a developer seeking to integrate with these applications.

Message Oriented Middleware

Message Oriented Middleware (MOM) is software that moves data in discrete packets called messages. Messaging implementations have been around since the 1970s and were one of the original mechanisms for communications in distributed systems. While early implementations were custom solutions fitting a specific need, standardized solutions evolved with common behaviors. Most MOM software products provide a variety of features, including point-to-point messaging using message queues, guaranteed message delivery, both synchronous and asynchronous operation, and multiple destination delivery using a publish/subscribe strategy.

Message Queues

The Message Queues product allows an application to send a message to another application using a queue. A software component called a

queue manager ensures acceptance of the message and acknowledges receipt to the sending application. Figure 4.7 depicts the process. Application A sends a message to the queue manager, which places the message on a queue and acknowledges receipt to the sender. Note that this acknowledgement means simply that the message has been placed on the queue, not that the receiver has actually received the message. The receiving application, B, programmatically connects to the queue. When a message is available, application B is notified, and can read the next message in the queue. The advantage of using queued messaging is that processing in applications A and B is decoupled. Application A can send a message and continue processing, achieving greater overall performance. Application B can process messages at its own pace.

Most queue managers can provide a service called guaranteed delivery. The queue manager ensures that the message is safely stored prior to acknowledging receipt to the sender. Once an acknowledgement is received, the sender can be assured that the message will be delivered eventually, even if the receiving application is currently off-line.

In addition to the asynchronous mode represented above, some message queue implementations provide a synchronous messaging mode. In this mode, system A is blocked until system B actually reads the message from the queue.

As an example of queued messaging, a Web application may collect an order from a customer. After verifying that all information is present, the Web application can send the order as a message to a queue for order processing. The order-processing system can read one message at a time, process it, then notify the original user of the outcome using some other mechanism, such as an e-mail message.

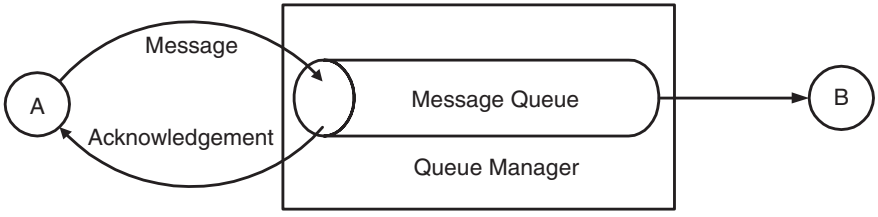

Figure 4.7. Message queues.

Publish/Subscribe

Publish/Subscribe communication (called pub/sub for short), further decouples senders from receivers. Rather than sending a message to a specific application, an application that generates information, called a publisher, sends a message to a pub/sub software component, simply identifying the subject of the message, often called a topic. The pub/sub software then forwards the message to any application that has subscribed to that topic via the pub/sub operation provided for this purpose. Figure 4.8 shows an application, A, sending a message to a topic, subscribed to by three applications, B, C, and D.

Pub/Sub technology is useful to distribute specific types of information to multiple applications. For example, a company may have one primary customer management system, which, when updated, distributes name and address updates to other applications within the enterprise. These other applications can register with the pub/sub engine for customer information updates. When a name or address changes, the customer management system sends a message to the pub/sub engine, which then distributes the updates to all applications subscribed to that topic.

Remote Procedure Calls

The growth of networking in the 1970s and early 1980s led to a rapid expansion in the use of distributed applications, where an application's functionality was spread across multiple hardware systems and the network became the medium that allowed the various components to com-

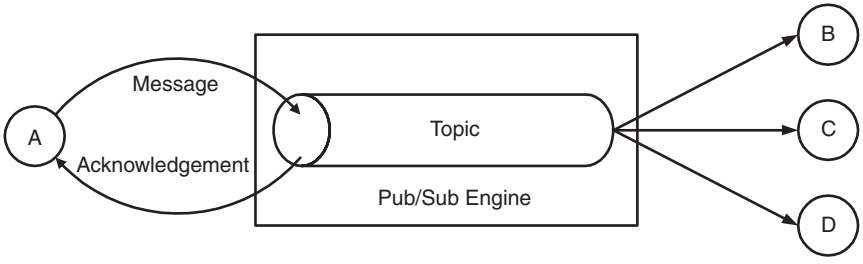

Figure 4.8. Publish/Subscribe.

municate. Procedural programming languages were prevalent at the time, and software design techniques dictated that functionality be abstracted in small units of work, usually referred to as procedures (similar in concept to today's methods in an object-oriented language such as Java or C++). Various techniques emerged to allow the calling of a procedure across the network, a class of technology called Remote Procedure Calls, or RPCs. These techniques were formalized into standards by several organizations. Among the most popular RPC standards were Open Network Computing (ONC) RPC from Sun Microsystems, and the Distributed Computing Environment (DCE) RPC from the Open Software Foundation.

The goal for an RPC implementation is to abstract the complexities of the network to make a call to a remote service appear as simple as a call to a local procedure. The "magic" that makes this possible is a pair of software components, called a stub and a skeleton, that provide the means to invoke the remote procedure and return results to the caller. The stub acts as a proxy for the remote procedure, exposing the procedure's functionality to the local application and hiding the complexities of managing the operation over a network connection. The sequence of events for a typical remote procedure is as follows (see Figure 4.9):

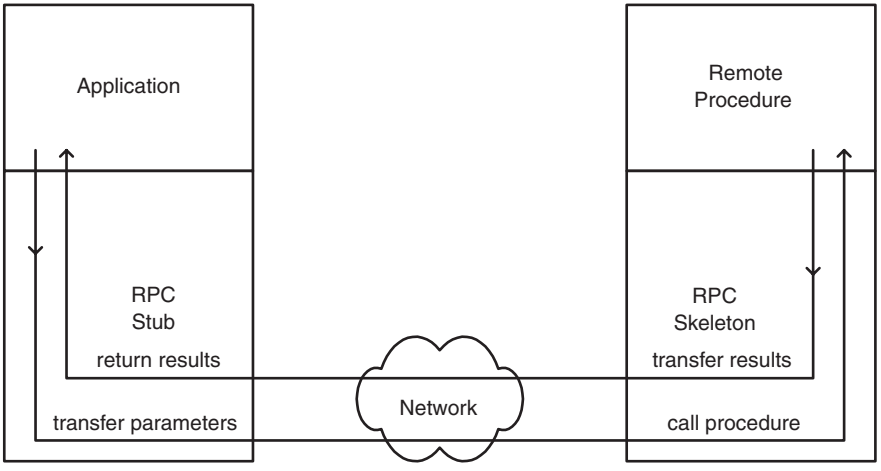

Figure 4.9. RPC execution.

1. An application makes a simple procedure call to the RPC stub. The interface for this procedure mimics that of the remote procedure.

2. The RPC stub locates the remote procedure and transfers the parameters over the network to the skeleton residing on the same computer as the desired remote procedure.

3. The RPC skeleton receives the parameters, then calls the desired remote procedure.

4. The RPC skeleton receives the results from the remote procedure, then transfers the data back to the RPC stub on the originating computer.

5. The RPC stub returns the results to the calling application.

RPC calls are typically synchronous; that is, the calling application is blocked until the results are available. Some RPC mechanisms allow asynchronous operation, and techniques are also available using multiple threads in the originating application to produce asynchronous behavior.

An important aspect of RPC is that the interface for a procedure is specified independently of the code that implements the procedure. The interface file is run through an RPC compiler to create actual code for the stub and skeleton, which can then be natively compiled and linked into the applications on both sides of the connection. The strategy of defining an interface independent of the implementation has evolved into a key software design practice for later technologies such as object-oriented development and distributed object technology.

Distributed Object Technology

Distributed object technology (DOT) is the object-oriented version of remote procedure calls. An application using distributed object technology defines a set of programmatic objects that are callable both on the local machine and remotely across the network. An object generally, but not always, represents a real-world object and defines various aspects of that object, such as the object's attributes and operations that may be performed on the object.

As an example, a retail Web site might manage a "shopping cart" for Web visitors who are potential purchasers. Users can view catalog items, add items to their shopping cart, and eventually "check out," specifying shipping and payment options. Such a Web site might use distributed object technology where the shopping cart is defined as a distributed object. Attributes of the shopping cart might include the customer name and address, and a list of items currently in the shopping cart. Operations provided by the shopping cart would include "add item," "delete item," "set shipping address," "set payment options," and "check out."

Figure 4.10 shows a generalized view of a distributed object implementation, modeled after Object Management Group's (OMG's) Common Object Request Broker Architecture (CORBA). The architecture is typical of all popular distributed object technologies including Enterprise JavaBeans and Microsoft's Distributed Component Object Model (DCOM). In this architecture, a client application is able to invoke methods on an object hosted by a remote computer running software called an object broker. The object broker provides an environment for managing all aspects of the objects, including administration, monitoring, communication between client and object, and object creation and destruction. Note that object methods can be invoked by remote clients or clients running on the same computer, which may be other objects.

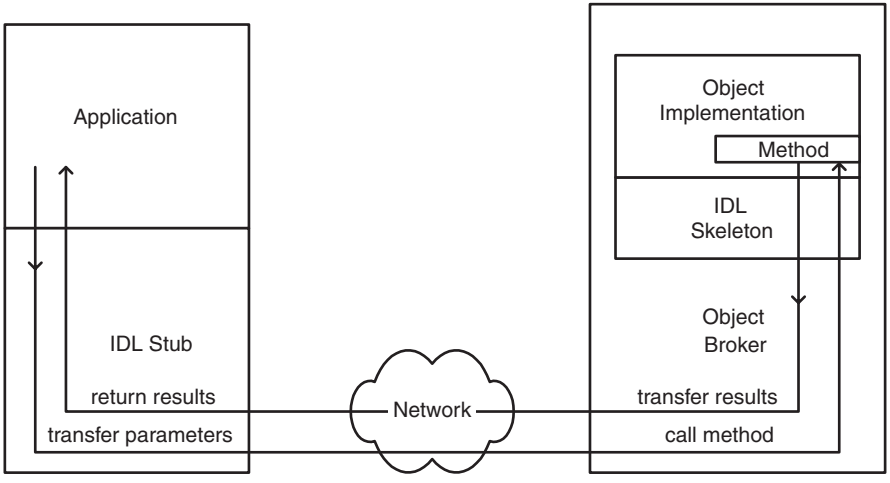

Figure 4.10. Distributed object method invocation.

Just as RPC technology uses an interface file to define how a remote procedure is called, distributed object technology uses an interface file called an Interface Description Language (IDL) file. IDL is a high-level language that precisely describes the methods and attributes for an object. Once the IDL is created, it is compiled into a programming language to create the IDL stub and IDL skeleton. A client application then accesses the local IDL stub as a proxy for the actual object, hiding the complexities of calling methods on an object over a network. The IDL skeleton becomes part of the implementation of the object, which is deployed on the object broker, ready for use by other applications. Three popular distributed object technologies are in common use today. These are described in the following sections.

Common Object Request Broker Architecture

Common Object Request Broker Architecture is an open, vendor-independent distributed object technology standardized by the Object Management Group, a consortium of hardware and software vendors. CORBA is designed to handle demanding computing requirements, supporting both high scalability and fault tolerance. CORBA implementations run on many platforms, and the technology is used in many implementations that require large numbers of simultaneous clients and high reliability.

The CORBA architecture follows the architecture described above. It has the advantage of being completely vendor neutral, allowing many programming languages to create or use CORBA objects. The key enabler for this technology is the IDL file, which can be compiled into stubs and skeletons for multiple languages. Client applications then compile the stub into their applications for access to the CORBA objects. The skeleton is used to create the object, which is then deployed to an object request broker, which manages the environment and interactions with clients.

Enterprise JavaBeans

Enterprise JavaBeans (EJB) is a technology developed by Sun Microsystems to address the needs of Java developers creating enterprise-class applications. EJB is a distributed object technology that defines an environment and programming strategy for software components. The environment provided for EJB is Sun's Java 2 Enterprise Edition, a

specification for Java-based, enterprise-class application servers. The J2EE specification is supported by many vendors and freeware organizations who make available servers conforming to the J2EE specification, including Sun, IBM, BEA Systems, Oracle, and JBoss.

EJB development on a J2EE platform is intended to lead to applications that are scalable, transactional, and secure. Many of the complex issues related to these capabilities are built into the J2EE platform, available automatically to the EJB developer as long as standard implementation processes are followed. Consistent with Java portability goals, EJB-based applications may be written once and then deployed on any server platform that supports the Enterprise JavaBeans specification.

Three categories of EJBs exist, to be used for various purposes in an application:

- Entity beans represent an object that contains data, such as a customer or order, plus operations that may be performed on the object. Entity beans are intended to be persistent, meaning they are at all times stored to a fixed medium to survive program bugs, restarts, or hardware failures. Automatic persistence is available for entity beans whose data originates in an RDBMSs.

- Session beans represent a series of interactions by a client application, such as a consumer loan application process consisting of multiple Web pages. The session bean can save information across interactions, but the bean itself is destroyed when the session terminates.

- Message-driven beans respond to messages received from message queues, using the Java Message Service (JMS), Java's standard interface to MOM systems. This provides an asynchronous processing capability, where the message-driven bean can be invoked by the receipt of a message sent from some other application.

Microsoft Distributed Component Object Model

Microsoft's Component Object Model (COM) is an interface technology following the distributed object technology architecture discussed above. It is the Microsoft-standard interface between almost any type of software component in Microsoft environments. COM uses its own

flavor of interface description language to precisely define interfaces to objects and adheres to the object-oriented principle of polymorphism, which means a particular interface definition can be implemented for a wide variety of objects. For example, cut, copy, and paste operations can be applied to text in a word processor, icons in a drawing program, or an activity in a flowgraph program.

For the distributed environment, Microsoft provides Distributed COM and COM+, which allow objects to be invoked across computers. COM+ has the additional capabilities of working closely with Microsoft's server technologies, environments specifically designed for hosting distributed objects that provide scalability, integrated security, and transaction management. COM is supported in several programming languages, and is embedded in many higher-level concepts in the Microsoft environment:

- *COM objects.* A component can perform virtually any type of processing and can expose its functionality using a COM interface.

- *Automation, sometimes called OLE Automation.* A desktop application can be controlled by another program by exposing its functionality using COM interfaces. For example, a program could launch Microsoft Excel, open a spreadsheet, read a value from a cell, then perform additional processing on this data.

- *ActiveX controls.* Generally visual in nature, but not always, ActiveX Controls are components that perform a specific function in an application, such as a visual calendar to assist the user in selecting a date. Microsoft arguably became the leader in component-based development in the mid- to late 1990s because of the widespread use of ActiveX controls (previously called OLE controls) for Visual Basic programmers. An entire industry segment sprang up to supply developers with components that were easily incorporated into Visual Basic and other Microsoft-based applications. Many ActiveX controls can also be used in Web applications, providing capabilities beyond standard Web technologies.

- *Application programming interfaces.* Many of Microsoft's programming toolkits are invoked using APIs based on COM.

- *Compound documents.* Microsoft technology allows a single document to contain objects originating from other applications. For example, a word processing document may contain a chart created from a spreadsheet program. When the user activates the chart, the originating application launches, temporarily incorporating its capabilities (menus, toolbars, drawing tools, etc.) into the current application.

Web Services

Web services are an example of an SOA (Service Oriented Architecture), which seeks to build applications out of loosely coupled, well-defined services representing blocks of business functionality. By loosely coupled, we mean a strategy of implementing a service where the specific technology requirements of client applications are considered only in an abstract sense. Rather than implementing an interface for a single, specific client, a designer instead defines a more general interface with the intent of servicing an entire class of applications. If successful, a loosely coupled design allows the swapping out of components fairly easily, as long as the interface is conformed to. In addition to loose coupling, SOAs generally provide a mechanism for remote discovery of interfaces so that an application can search for services and dynamically discover the specific interface requirements. Web services are composed of three main technologies: Simple Object Access Protocol (SOAP), Web Services Description Language (WSDL), and Universal Description, Discovery and Integration (UDDI). These technologies together provide the key aspects of an SOA: WSDL provides the definition of a service and its interface specification, UDDI provides a searchable directory so that applications can find an appropriate service, and SOAP provides the mechanism to invoke the service. Figure 4.11 shows the relationships between these three technologies.

Transaction Processing Monitors

A transaction processing (TP) monitor is the precursor to today's application servers, having been in use for over 25 years. TP monitors provide robust processing capabilities to high-volume processing

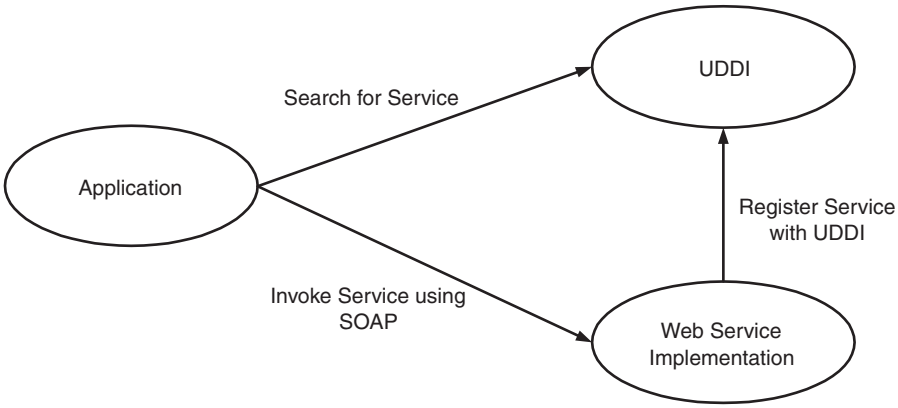

Figure 4.11. Web services architecture.

environments. Ideally suited to applications serving thousands of clients simultaneously, TP monitors funnel the many simultaneous requests into a small, manageable number of connections to back-end systems such as RDBMSs. The effect is to maintain a constant, manageably high load on the back-end systems while smoothing out the response times for the client systems. Though individual clients may experience slightly slower response times compared with a direct connection to the back end, the throttling mechanism of the TP monitor provides a much higher utilization of the back-end systems, leading to more efficient use of computing resources.

TP monitors generally provide management features such as automatic restarting of failed processes, dynamic load balancing, and consistency of distributed data using transactions. A transaction is a complex operation composed of multiple suboperations, which are made so that all succeed or all fail.

Application Servers

An application server is a software platform designed to simplify the creation of enterprise-class applications such as payroll systems, sales force automation, and high-volume Web sites. Many of these applications have similar detailed technical requirements, such as data access,

transaction management, and high scalability. Application server developers recognized the similarities of these applications and sought to develop these common, infrastructure-level features into a server, where they can be provided consistently to software components involved in an application. The benefit of using an application server is that it reduces the amount of custom infrastructure code that developers must write, freeing them to focus on coding business logic specific to their application.

Generally, an application server serves as the middle tier in a three-tier architecture. Software components running on the application server implement the primary logic of the application, independent of the graphical user interface. These components generally implement business rules and interact with enterprise systems such as databases and mainframes in the lower tier. Several categories of application servers are in use:

- *J2EE*. The most common type of application server conforms to the Java 2 Enterprise Edition specification published by Sun Microsystems, the inventor of the Java programming language. Vendors such as IBM, Sun, Oracle, and BEA Systems, as well as several freeware organizations, are supplying J2EE application servers to the market. These application servers provide a wide variety of important services to application developers, including distributed object technology using Enterprise JavaBeans, messaging, distributed transactions, Web application hosting using servlets and Java Server Pages (JSPs), and standards-based connectivity to enterprise systems using the Java Connector Architecture (JCA).

- *CORBA Object Request Broker (ORB)*. The standard environment for CORBA technology, the ORB hosts multiple instances of CORBA objects, provides common capabilities to CORBA objects, and can be configured for scalability, load balancing, fault tolerance, and resource management.

- *Web servers*. Though debated by some, Web servers can be described as application servers for purely Web-based applications. They provide common services for these Web-based applications, including a common visual presentation language environment (HTML), a common execution environment for components

(Common Gateway Interface, Internet Server API, and Netscape Server API), and a communication mechanism for client applications, based on Hypertext Transfer Protocol (HTTP).

- *Microsoft servers.* Microsoft does not market a product called an application server, but application server features, such as distributed transaction management, scalability features, and a common execution environment for components, have been available in Microsoft servers for years. The Windows .NET Server, not yet released at the time of this writing, appears to pull all the features of an application server together into a single product that can be viewed as an application server.

Transformation

Often, when data must be moved among systems, the structure and definition of the data fields vary greatly between systems. Transformation is the ability to convert the structure and element types to a different format to conform to a target definition. The structure of the data set, individual element data definitions, and relationships between the data elements are collectively called the schema for the data. Transformation is the process of converting from one schema to another. Various issues can arise when performing this conversion:

- *Missing fields.* A field in one schema may be missing from the other schema (e.g., address 3 in one schema has no counterpart in the other schema).

- *Datatype mismatch.* A field may have different data type (e.g., date type instead of string type).

- *Split fields.* A single field in one schema may have to be split into multiple fields for the second schema.

- *Structure mismatch.* Sections of one schema may map to multiple sections in another schema, (e.g., one schema may have a single customer section with name and address in one schema while another schema has name, plus shipping address and billing address).

- *Calculated fields*. Data in the target schema may need to be derived by applying a formula to fields in the source schema. For example, the target schema may include a subtotal field, which is calculated by summing the items from the source data set.

- *Data source differences*. One data set may be contained in a comma-delimited file, and the other data source may be in an RDBMSs.

Transformation tools help to manage the complexity of mapping from one schema to another. The result of a transformation tool is the ability to take data conforming to the first schema and convert it to data conforming to the second schema.

Message Brokers

In the course of integrating systems in an enterprise, companies generally begin the process by connecting a small number of critical systems, using a point-to-point strategy such as message queues. This approach works well for a limited number of systems due to its shear simplicity. However, consider the growing architecture as more systems are integrated. Figure 4.12 shows 6 systems with interconnections between them. In the diagram, we see that for a configuration of 6 systems, each system connects to 5 other systems. If we use message queues for communication, a reasonable assumption, each system will need 5 queues to accept inbound messages, and 5 queues to send outbound messages, leading to a total of 30 queues for full connectivity. In general, with N systems, we will have $(N^2 - N) / 2$ pairings, leading to $N^2 - N$ queues for such a configuration. So, integrating 20 systems will require 380 separate queues for full connectivity. Integrating 50 systems will require 2,450 queues for full connectivity. Integrating 100 systems will require 9,900 queues. As you can see, this strategy may be manageable in small configurations, but enterprises with dozens or even hundreds of systems will quickly get bogged down in a severe management and maintenance problem. In addition to the sheer number of connections to manage, adding a new system potentially requires the addition of new pairs. The overhead of maintaining full connectivity is too great for any reasonable organization to be expected to be successful with this type of architecture.

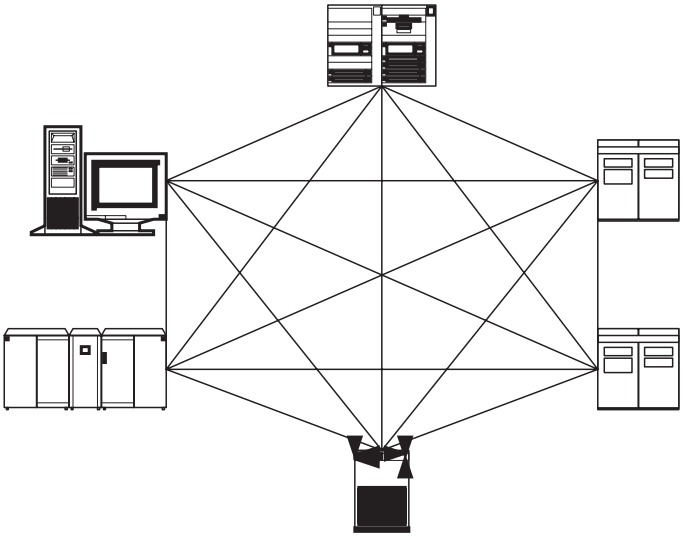

Figure 4.12. Point-to-point integration.

To solve this problem, a different architecture is required. Message brokers evolved as a solution to this problem, where the broker becomes the hub in a hub and spoke architecture. Figure 4.13 shows the hub and spoke architecture of the typical message broker. Connectivity is established between each system and the message broker, leading to a linear growth in the number of connections rather than the geometric growth we saw for the point-to-point architecture.

In exchange for a radically reduced architectural complexity, a message between systems now must be routed by the message broker. The message broker contains information on all the systems, so the message address to a logical system can be appropriately routed to a physical location.

Message brokers have evolved so that they not only provide connectivity management and routing, but also often include other advanced features, such as the following:

- *Transformation.* The message broker can transform a message to a specified format acceptable to a receiving system.

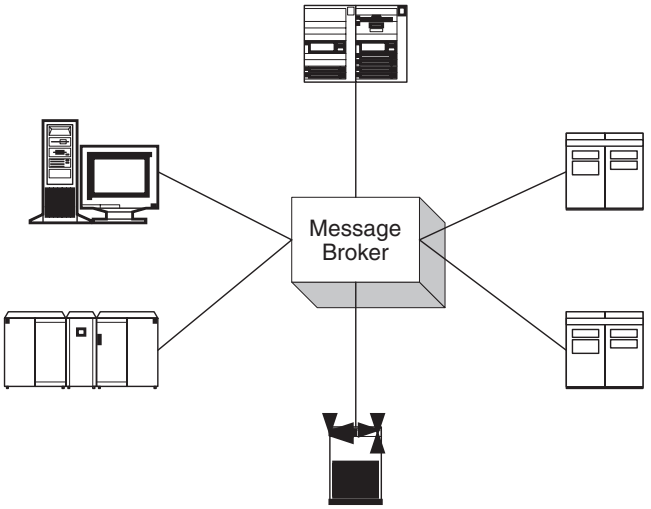

Figure 4.13. Message broker architecture.

- *Rules processing*. The message broker can intelligently transform or route messages based on content, message type, or scripted rules.

- *Message warehousing*. Messages can be stored for auditing and historical purposes in a format suitable for reporting and analysis.

- *Management services*. These include features to configure and manage the participating systems, and a repository to store and manage information about each participant such as message formats, security, etc.

Virtual Database

A virtual database (VDB) is a software layer that makes many databases look like a single, logical database. VDB technology is designed to reduce the complexity of data access. While analysts estimate that data access consumes up to 70% of the programming resources for a project

in a distributed environment, the VDB technology reduces this number significantly. A VDB uses information about data sources, formats, and structures, called metadata, to define logical views of the data. Views are then accessed by programmers, reporting systems, analytical tools, or other applications as though they were native data sources (see Figure 4.14)

A VDB implementation requires a design step to create the logical views. Ideally, a company will design these views with reusability in mind so that many applications can take advantage of the data. Generally, an architect or other data-savvy individual familiar with the enterprise data sources builds the views.

Many view VDB technology as being similar to data warehouse technology, which moves data from many sources into a centralized data store, where it can then be queried. Since this latter strategy introduces latency in the data (data is generally pulled periodically, such as nightly

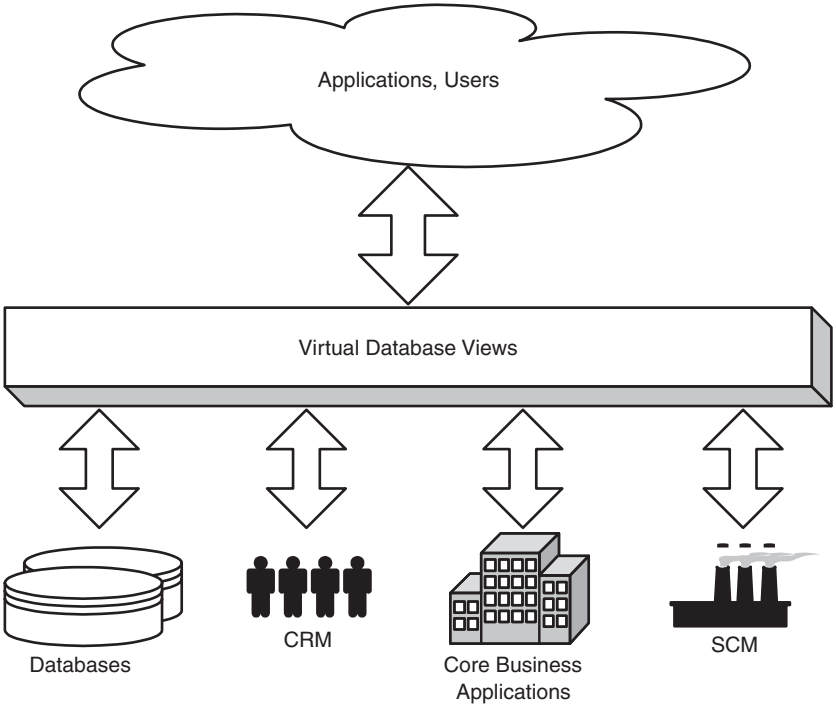

Figure 4.14. Virtual database.

or weekly), it is most useful when real-time or near-real-time data is not required, whereas VDB technology supports real-time or near-real-time access. Data warehousing technology is also useful only for reading and analyzing data, whereas VDB technology supports both reading and writing data to the enterprise. While evaluating VDB vendors, however, note that some VDB vendors support only reading data.

A VDB provides a mechanism to read and possibly write data to and from enterprise systems. This mechanism is called a query language, and falls into two broad categories:

- A fine-grain query language provides highly detailed access to data sources, allowing specification of field-level information, and logical operators to select precisely what information should be retrieved. Examples include SQL, and the XML-based query languages XPath and XQuery.

- A coarse-grain query language defines a higher-level abstraction for the data, generally defining a business object interface to read and write data. The coarse-grain query is modeled after SOAs such as Web services, where a block of functionality is invoked as a service with a small number of parameters. A business object (such as a customer, for example) is exposed through a service interface that may require a customer ID or customer search string.

The Virtual XML Database

The growth in use of XML has spawned a new breed of VDB, which presents its views in XML format. Called a Virtual XML Database (VxDB), this type of VDB exposes XML views to client applications, where each view conforms to a certain XML format. VxDBs typically provide an XML-based query language such as XPath or XQuery, or a coarse-grain query model that exposes an XML view as a named object with a number of parameters. For example, a customer XML view can be exposed that includes XML elements and attributes defining a customer profile and provisioned services. An application could request the customer view and provide the customer ID for the desired customer. The VxDB gathers the data from the appropriate enterprise systems, converts the data to XML, and then returns the XML document to the requesting application.

Extraction, Transformation, and Loading

Most businesses need to analyze their business information in order to monitor business activities and seek out new opportunities for growth. While analyzing data can and often does occur within a single system, say an order-processing system, an accurate view of the enterprise cannot be achieved without viewing information across the many systems used by the enterprise. The typical solution is to create a data warehouse to hold a copy of all the important information across the enterprise. The data warehouse is a database specifically designed to assist in the decision-making process. Various decision-support and business-analysis software packages access the data warehouse to present information to business users and decision makers.

The question is, "How does data get from the many systems within the enterprise into the data warehouse?" Extraction, transformation, and loading (ETL) software is specifically designed to solve this problem. The data warehouse has a specific database schema that defines all the business structures important in the analysis of the business. But it is unlikely that any of the business systems have the same schema. Instead, data is extracted from a business system, transformed to conform to the data warehouse schema, then loaded into the data warehouse. ETL tools, though specifically designed to support a data warehouse, share many attributes with more general integration tools. ETL tools can often be used to implement data integration between two systems, and a variety of integration tools such as adapters, messaging and transformation tools, and virtual databases can be employed in combination to move data into the data warehouse. The right tool for a company will depend on whether the company would like the flexibility of general integration technology useful for broader integration tasks or a highly focused tool specifically designed for a single purpose.

Data Quality Issues

One of the major roadblocks to integrating systems is the problem of dirty data. Most data in enterprise systems was manually entered at some point in the past, and changes to data occur over years and decades as customers come and go, change residences, change products

and services, etc. Problems can affect virtually any data that users manually enter, which ultimately includes much of the data in the systems. The goal of data quality management is to improve the accuracy of the data before, during, and after an integration project.

A wide range of data problems are accumulated in systems over the years, reflecting many potential problems with data. Dirty data is generally an automation inhibitor, meaning various automation projects, including integration, are adversely affected by dirty data. The reason many companies don't invest in data quality management tools is that most systems, in isolation, can continue operating without degraded performance. A common problem is that the same physical entity, like a customer, is represented with more than one similar name and address. For example, "Bob Williams, 2501 N. 6th Street" is similar to "Robert N. Williams, 2501 North 6th St." A human operator should recognize these as the same customer, so the existing systems may be sufficient for performing customer service. However, automated processes will generally not recognize these as the same, which leads to the very real problem of duplicate accounts for the same person, multiple mailings being sent to the same person, and the inability to correlate a customer's account across multiple systems. A list of potential problems includes the following:

- Spelling and keyboard entry anomalies
- Inconsistent use of abbreviations and acronyms
- Inconsistent use of middle name and initials
- Variation in date and currency formats
- Missing data (e.g., area code or state not typed in)
- Spillover (e.g., a portion of name field typed into one of the address lines due to space limitations)
- Embedded coded values (e.g., a product code entered in the description text field)
- Incorrect data in fields (e.g., a portion of the name appearing in an address field)

- Business and personal names intermixed
- Text descriptions in data fields (e.g., special dialing instructions in a phone field)
- "Creative" data entry, where operators inject new attributes into existing text fields.

Data Quality Tools

Vendors have created a host of software products to help clean up dirty data. The most common tools are described in the following sections.

Data Analysis

Data analysis is the process of inspecting existing data to determine its characteristics and possible quality problems. Data analysis tools allow accessing data sets, then inspecting each field to perform a statistical analysis, creating a report of characteristics, including minimum and maximum values and potential abnormal conditions. The purpose of the analysis is to understand the source data in depth to determine a strategy for data cleanup.

Address Correction

Given an address, address correction software will reformat it according to local postal requirements, flagging invalid addresses for later manual processing. This software is available in both batch and interactive modes. Interactive modes assist users in entering addresses during interactive data entry. This software often speeds the data entry process by allowing shortcuts, such as entering a zip code first to automatically populate city and state data. In the end, the software assures that the entered address is a real physical address with consistent formatting.

Matching

Matching software attempts to match one record with one or more other records in a set. This software includes various algorithms to perform a "fuzzy search" on the target record set, where similar records are found and scored with a match confidence level. These algorithms include fea-

tures such as synonym matching, so that "Bob" and "Robert" are treated synonymously, and keyboard heuristics, so that simple typos can be accounted for in the match. To account for typos, many algorithms include the concept of "keyboard distance," so that keys close together may be treated as almost synonymous. The "I" and "O" keys are next to one another on a standard QWERTY keyboard, so an algorithm may treat "Robert" and "Ribert" as possible synonyms.

Matching algorithms are often executed interactively to aid in account searching. For example, a service agent on the telephone may enter a customer's name to locate her account information. A matching algorithm is executed to present a list of candidate records, with key information such as name and address, ranked according to the match confidence level. The agent can then select the appropriate account.

Deduplication

Deduplication (deduping) is the process of scanning a set of records and identifying those records that represent the same physical entity, such as a customer. This type of tool is invaluable to companies that send communications to customers via postal mail. Here, the elimination of duplicate records ensures that only one mailing will be sent to an individual, rather than two or more. This results in a concrete savings in collateral and postage, and also fosters better customer relationships, as receiving duplicate mailings irritates most customers and reveals inefficiencies in a company's operations. For companies that send frequent communications, the cost savings are quite significant.

Concordance Database

In the typical large enterprise, business information often resides across a number of isolated enterprise systems, each with its own unique identifiers for records such as customers. Many integration projects will require records to be correlated based on whether they represent the same physical entity, such as a customer. For example, in a company where paid services are managed in separate systems, each system will have its own definition of a customer. In this situation, creating a "single customer view" requires correlating customer records. One approach would be to execute a "fuzzy match" each time a customer view is being called upon. But this is a computation-intensive process that should be avoided if possible. The typical solution is to create a separate database, called a concordance database, that correlates records in different systems.

Creation and management of the concordance database utilizes other data cleansing tools such as address correction and deduping. The process reads key information and important matching fields, such as name and address, from each of the participating systems and applies matching and/or deduping algorithms to associate the related data into single entity records. Data cleansing is often applied for fields such as names and addresses so that the concordance database includes the cleanest possible data. The result is that the concordance database contains a single record for each business entity contained in all source systems, plus a key into each source system that participated in the process. Once the concordance database is established, applications can use it to perform a preliminary search for records, allowing additional retrieval from the individual systems once the proper entity is identified. Ongoing management of the concordance database requires synchronization with the individual systems. When information changes in one system, such as an address change, that change should be propagated to the other systems. This synchronization, though not a trivial task, can be carried out by other integration tools, such as a message broker sending update requests to the target systems.

Communication Models

An important concept in designing how systems interoperate is the model of communication that the systems support. Interactions between systems can be synchronous or asynchronous. Systems can support one or both of these communication models. Generally, supporting both models provides greater flexibility and ease of integration, although with some programming effort it is usually possible to make one model behave like the other model.

Synchronous Communication

Synchronous communication means that a component invokes another component and waits until the second component completes its action before proceeding. The most straightforward example of synchronous communications is a local procedure or method call. In the local procedure call example, the first component makes a direct procedure call to

a second component located on the same computer in the same process. The synchronous model extends to components in different processes or different machines by using a variety of technologies, such as RPC, distributed object technology, or Web services. The most popular type of synchronous communication, called request-response, is shown in Figure 4.15. Here, component A sends a request to component B and waits until a response is received.

Synchronous communication is used when the requestor depends on the response before proceeding. For example, an order-processing component may need to calculate sales tax as part of determining the total for an order. The component would send a request to the sales tax calculator as a synchronous call, waiting until the sales tax is received as a response, after which processing can proceed. Other situations where synchronous communication is ideal include most cases where processing is in response to user interaction. In this situation, processing is usually invoked in the context of a user command, such as clicking on the Submit button on a Web page. All steps in the processing should be carried out synchronously and a result provided to the user.

The disadvantage of synchronous communication is that components become dependent on one another. Any adverse condition in the called component, such as server problems or network problems, can potentially block the caller indefinitely.

Asynchronous Communication

In cases like the sales tax calculator example above, the calling component can perform no other processing until a result is returned. In many situations, continued processing does not depend on a result returned

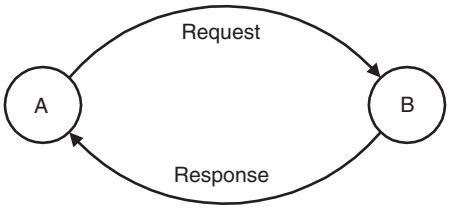

Figure 4.15. Request-response.

from an invocation, and a second communication model can be used—asynchronous communication.

In the asynchronous communication model, a component invokes a second component, then continues processing without waiting for a result. In some cases, the component sends the request and continues on its way, never caring whether the operation completes. In other cases, the component expects to be notified at a later time when the operation completes, a process called asynchronous notification.

Asynchronous communication is generally accomplished between components using messaging technology. Three types of asynchronous messaging are in common use:

- Message queues allow an application to send a message to another application using a queue. The sending application continues processing as soon as the message is placed in the queue. The receiving application reads messages from the queue at its own pace.

- In publish/subscribe messaging an application that generates information, called a publisher, sends a message to the pub/sub software and identifies the subject of the message, often called a topic. Applications interested in a topic register with the pub/sub software, which forwards the message to all subscribed applications.

- Broadcasting allows an application to send a message to any connected application, without regard for which application, if any, ever receives it. A common use of broadcast messaging is to broadcast status from a server providing a specific set of services. The server will broadcast its current status, and any online application can receive the message and take appropriate action. For example, the status may indicate degraded performance, in which case an application may seek an alternate source for required services. Figure 4.16 shows a typical example, where application A sends a broadcast message, which is routed to all connected systems, B, C, and D.

Summary

In this chapter, we looked at a wide variety of technologies composing the foundation of integration. These technologies share a common pur-

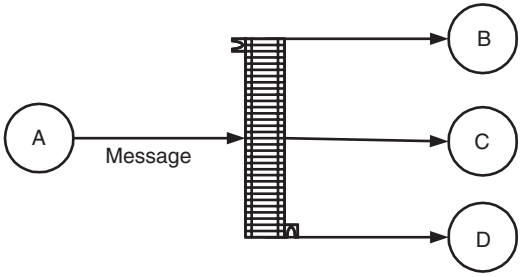

Figure 4.16. Broadcasting a message.

pose, that of enabling communication between the systems, applications, and databases within an organization. While these technologies are the tools that make integration work, they can be misused without a governing strategy. In the next chapter, we look at the last aspect of the Integration Cube, enterprise integration strategies, which provide the governing approach to achieving the Connected Enterprise.