Overview of Enterprise Application Integration

1

The integration of applications within and between businesses has become a hot topic. It is motivated by B2B (Business to Business), CRM (Customer Relationship Management) and by the Internet in general. The technologies that make this integration possible are called middleware. The aim of this chapter is to present a general picture of the problems related to integration, on the technologies used and the architectures that result. Technologies and architectures are then described in more detail in the following chapters.

1.1 Problems to Solve

The business application integration market is valued at more than US$40 million for the year 2000 and US$60 million for 2001. The cost of integration represents on an average 40% of the business information processing budget. These enormous sums are due to the needs for companies to evolve and to be competitive by continually adapting themselves to the needs and constraints of the market. This situation leads them to define objectives that will be achieved thanks to new working methods that use information tools. New software must be bought or developed and must be able to co-exist with old applications. All this software, of whatever origin, must be able to exchange information. On the other hand, in order to be increasingly close to the customer, a business must make its software accessible to the external world.

Thus every company's information processing manager sees themselves confronted, at the software level, by the following problems:

- provide a consolidated view of the company's data;
- integrate software from diverse sources;
- offer access to software to those inside and those outside the company;

- automatically connect the tasks related to a business function;
- reduce the reaction time of their information system;
- rapidly develop new applications.

Rapid application development is not the subject of this book. However, the architectures and technologies that are proposed here can make big contributions in accelerating the rate at which applications are developed.

1.1.1 A Consolidated View of Business Data

The information system must be able to offer a global picture of business data that is coherent and up-to-date (e.g. clients, orders, partners, accounts). This clarity is however far from being the rule. Many companies are organized by product or product line. Typically, at the information level, this is translated by an application and/or a database per product. Each database contains the description of the clients who have bought the product (Figure 1.1)

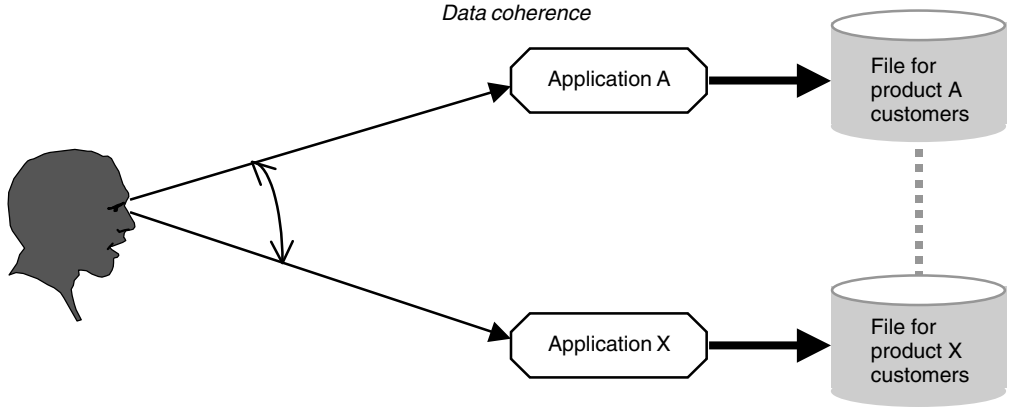

Figure 1.1 *Information system organised by product: the client's view is obscured*

Today, businesses want to place the customer at the centre of their activities. They want to have a global view of their customers, for example, to be able to list the products bought by a given customer. Old information architectures (Figure 1.1) therefore present two kinds of problem:

- The existing data models in each database are different and not necessarily consistent. Each gives a partial view of the customer. It is necessary to go through all the databases in order to discover all the products that a customer has bought. This is not always possible…

- Data incoherence leads to the fact that a customer *x* is designated in different ways in each database and is therefore difficult to recognize that customers *a* and *b* in fact represent the same entity.

The view of globally coherent data therefore imposes their extraction, their transformation to a common model and the definition of consistency rules so that duplicates can be identified.

1.1.2 Application Integration

The dream of every information systems manager is to be able to make their environment evolve easily. Now, the introduction of new software requires that it must exchange data with existing software. Communication channels must be constructed between each pair of applications before communication occurs. These connections can be complex and hard to manage if the two inter-communicating applications execute on different machines.

In the example in Figure 1.2, applications A and B can communicate. Assume that application A is old and stores its data in a local file F. Let us now assume that the new application, B, obtains its data by reading data in a file F1 whose format is different from that of F. Equally, assume that applications A and B run on two different machines. The creation of a communications link between these two applications implies the use of transfer software and the reformating of data from file F for storage in file F1.

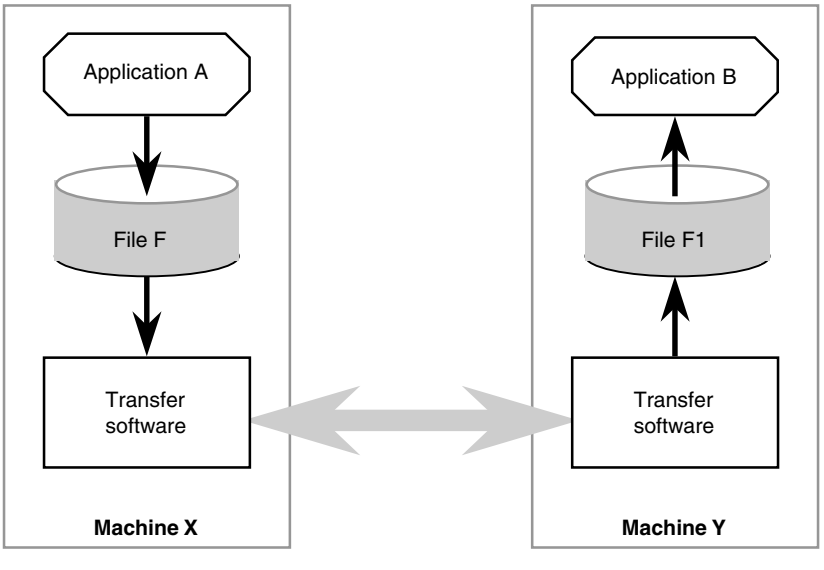

Figure 1.2 Simple communication link between two applications

Observe that the assumptions made in the preceding example are quite realistic. In order to make full use of the capacities of new generations of machine, a great deal of software is designed to run on them. Existing software, whose average age is 12 years, runs on machines of preceding generations whose operating systems are generally non-standard or proprietary.

Very often, to aquire some software that is useful for their company, an information manager is lead to buy a new machine. It then happens that their information environment becomes heterogeneous and therefore more complicated to manage.

Progressively, over a period of year, because applications have been added, the information systems of many companies resemble, if one examines the communications relations between software items, what has been called *spaghetti systems* (Figure 1.3).

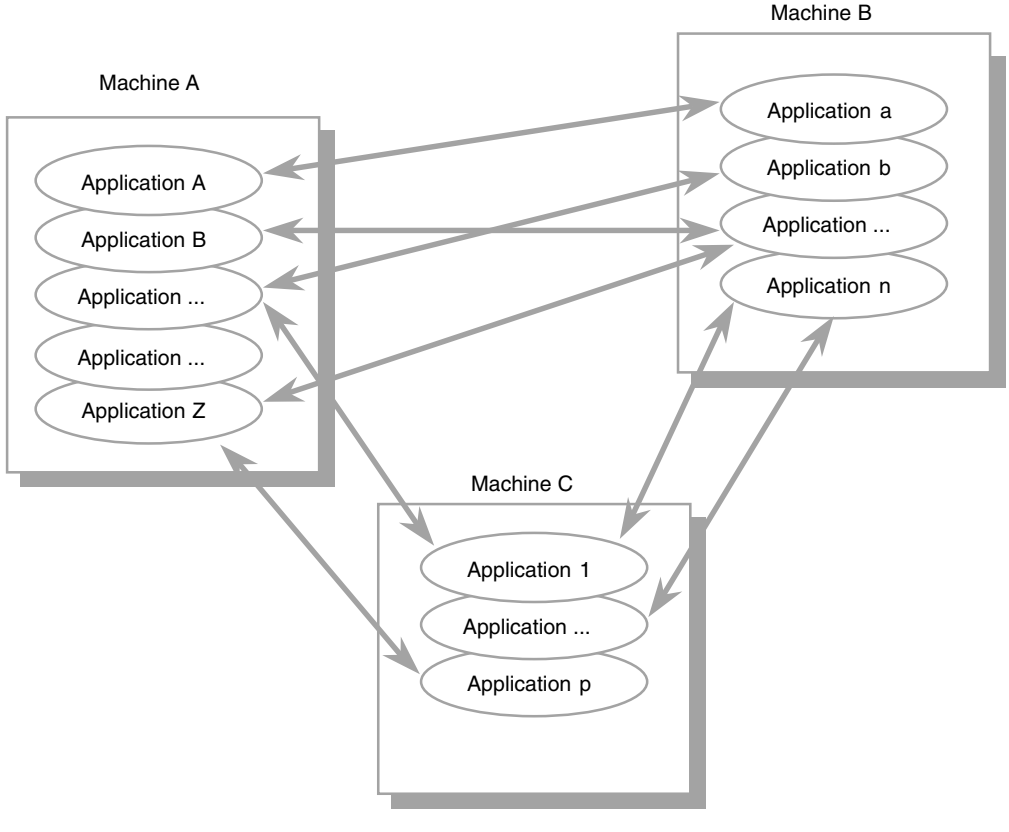

Figure 1.3 *Example of a spaghetti system*

Such a system is hard to manage and to develop. In order to avoid arriving at a system that is totally knotted with inter-application connections, a global solution must be devised. This solution must define a new architecture that allows easy integration of new applications and modification of existing ones.

1.1.3 Availability of Applications to the Entire World

Several factors lead to the need that access to certain company software must be possible outside the company, no matter where in the world it is. Let us consider some examples:

- Increasingly tight relationships between customers and suppliers leads the latter to give their customers free access to their order capture systems. The customers can then search the list of products and prices, and send their orders direct.
- In the area of banking, customers are apt to travel the world and to make requests of their banks from anywhere in the world. The possession of a private communications network by a company costs a considerable amount and therefore constitutes a brake on their geographic development.
- Companies having many branches or subsidiaries must maintain a certain number of applications and connect them to the central site. The problem of how to maintain the same version level of a piece of software in all branches is a problem that is, as yet, only incompletely solved.

Therefore, at present, companies experience the need to increase their geographical presence by offering easy access to their applications at the lowest cost.

1.1.4 Automatic Connection of Tasks Related to a Business Function

In order to adapt to market changes, companies must constantly redefine their operational processes (e.g. order processing activities). These processes are composed of a sequence of work steps that are connected according to a predefined sequence which depends on the expected goal and also on the data exchanged. During the automation of a business, each step in the automation process is represented by a piece of software. Very often the passage from one step to another consists of exchanging data performed in a manual fashion or by a virtual information connection that does not operate in real time (examples: file copy, block transfer).

The goal sought by EAI is to provide the technology for exchanging data and also the function of connection of tasks in order completely to automate the handling of a business process.

1.1.5 Decrease in Reaction Time for an Information System

Today, companies want to reduce the time between the instant when a piece of data is gathered by the information system and the instant when this data can be used by all applications. This lapse of time is called the latency time of a system. Such a need is clear in the banking world but also in many other areas.

Example: Not long ago, the French bank Z had a centralized database that contained the balance of each customer's account. When Mr Dupont wanted to withdraw money, he was able to go to his favourite branch, ask for the balance on his account and withdraw an amount, x. To do this, the branch connected to the central database and downloaded Mr Dupont's account. The withdrawal was then made locally and transmitted to the central database that evening after 6 p.m. During the night, the updates were performed and the new balance of the accounts produced. This means that if Mr Dupont was to go and ask for the balance of his account on the same day in another branch of the bank, he would have been astonished to see that the withdrawal of x had not been performed.

Typically, the latency time of this bank was one day. Today companies wish to reduce this time so that it corresponds to new requirements in their area of activity. This implies, at the information level, a high-performance system for inter-application exchange of information. This is the goal of middleware.

1.2 Application Integration: Semantic Level and System Level

In order to understand better what is done in a dialogue between two pieces of software, let us consider the example of the fictional PointCom company that wants to trade over the Internet (Figure 1.4). Let us assume that PointCom has the Ariba purchasing software (Ariba is the name of the company supplying software that is also called Ariba) which connects, over the Internet, to the market place software Oracle eXchange. The mechanisms of elements interposed in this exchange can be analyzed as follows:

- The dialogue or scenario process. Ariba has its own way of handling a purchase just as does Oracle eXchange and there is little chance that these

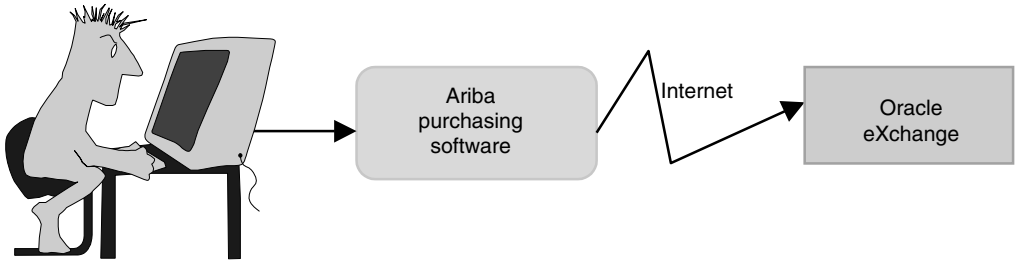

Figure 1.4 *Example of application integration. eXchange is a market place product from Oracle. Ariba is the name of a software company*

two ways of working are the same. Let us imagine two people, the seller and the buyer, who are discussing, for example, the purchase of a car. They each have a scenario in their head which can be:

- The seller: make them try the car, make an offer, have the offer accepted, complete the sale.
- The buyer: try the car, receive an offer, obtain other offers from other sellers, choose a seller, negotiate the price, buy.

Two people can adapt their respective scenarios in order to be able to engage in dialogue, but this is more difficult for two pieces of software. This means that an item of work must be performed at the level of dialogue processes when it is desired to integrate two software items. Such a scenario implies the description of the different steps and their connection, listing the hypotheses formed and handling exceptions:

- The data model: The exchanged data are, in fact, business documents (e.g. order forms) and not simple character strings. In the above example, there is no reason why the order form provided by Ariba should conform to what Oracle eXchange software expects. It is highly probable that these two documents will contain lots of identical data, but it will not necessarily be in the same format (example: the name of the customer is described in four fields in one case and in six fields in the other!) A conversion job from one model to another is therefore needed.
- The communication system: This, here, is a matter of the communication protocol used to exchange data. Here we come to the role of middleware. A large part of this work is about the different technologies implementing this exchange. Typically, in Internet contexts, the protocol used is HTTP (HyperText Transfer Protocol).

If the three preceding elements are considered, the communication system represents the exchange level of the system, the dialogue process and the

data model constitutes its semantic level. The first generation of middleware technologies implemented the system level and the second (the new generation) the semantic one.

1.2.1 Open Applications Group (OAG)

All of the above shows the complexity of interoperability between software. Only the existence of standards will allow the simplification of the construction of information solutions composed of heterogeneous software components. With this aim in mind, the Open Applications Group (OAG) consortium was formed in 1995. This group contributes to the definition of standards for the three elements used in an exchange. A model has been created which contains:

- Component definitions: Each application is modelled in the form of an object and offers a list of operations that can be executed on request.
- An application architecture: Communication between components is performed by exchanging a *Business Object Document* (BOD) according to the request/reply or publish/subscribe communications models.
- Scenario diagrams: The standard which describes inter-application dialogues is called OAGIS (Open Applications Group Integration Scenarii). Currently 51 scenarios are available. Each description consists of a summary, an exchange diagram, the list of assumptions made, the definition of the required components, connections of the exchanges and exception handling.
- The list of the middleware APIs (Application Programming Interfaces) required for communication. The standard is called OAMAS (Open Applications Middleware API Specification).
- A data dictionary describing the elements of the API.

For a more detailed description of all the existing standards, the reader should visit the OAG Internet site.

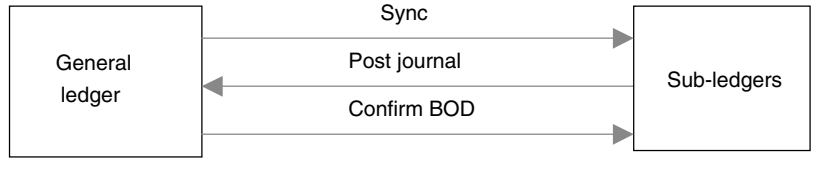

Figure 1.5 Example scenario diagram in OAGIS

1.2.2 The Client–Server Model

When several entities want to communicate between themselves, a communication protocol can be defined and accepted by all so as to avoid confusion. Different communication schemes exist. Television gives us many examples: debate moderated by a judge (the journalist) who gives the floor to each participant in turn (or uses a more complex algorithm); face-to-face interview between two people, one being the interviewer, the other the interviewee. The goal of such organization is that each question gets a reply and that dialogue occurs.

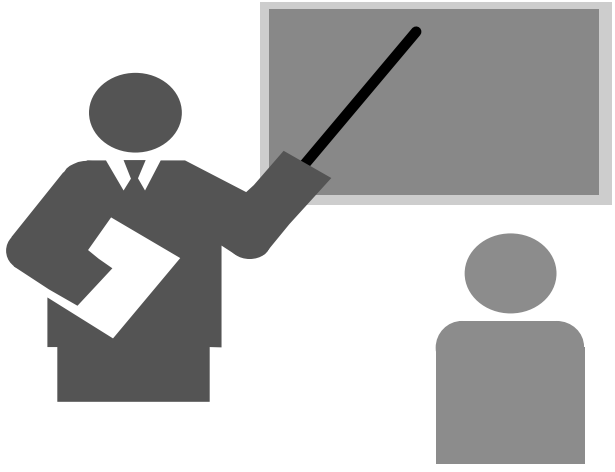

Figure 1.6 The master/pupil relation represents a particular communication structure

Over the years, different communications structures have been used to exchange data between programs. One of them scored with its flexibility. It is based on the interviewer/interviewee model and is characterized as follows:

- Communication implies only two entities. This model assumes that every communication involving elements of a group can be decomposed into a set of exchanges between two entities.
- One entity has the initiative in the dialogue (the interviewer) and the other waits for a request (interviewee).
- The interviewee entity is programmed to reply to a very precise set of requests. (Here, the analogy with what happens on TV is broken!) The list of permitted requests must be completely defined. This list is called the *interface* of the interviewed entity.

In this model of communication, the interviewed entity is described as offering services. To each request, there corresponds a well-defined service. For this reason, this entity is also called the *server*. The other entity, which requests services, is thus called the *client*. The whole forms the *client–server model*.

The operation of the client–server communications model is the following:

- The client sends a message containing a request to a server.
- The server executes the service associated with the request sent by the client.
- The server returns a message to the client that contains the result of the service it performed.

In this model, communication is always initiated by the client. The server is in a reactive mode. The model does not specify the way in which communications are implemented. The server's interface appears as a component that is essential for communication. It describes the list of permitted requests. Figure 1.7 shows the client–server model.

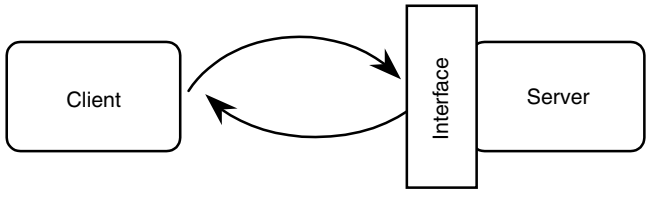

Figure 1.7 The client–server model

1.2.3 Three-tier Client–Server Model

At this point, it is interesting to see how the client–server model can be applied in a concrete setting. A first possibility consists of defining a client–server relationship between two applications. A second possibility is to decompose an application into a set of elements that are connected by this relation. Let us see how.

The functional analysis of an appilcation reveals three components:

1. The user interface component: This is most often graphic (windows) and allows the user to interact with the application. Frequently this part must be able to operate on different types of machine: PC, Macintosh, workstations or alphanumeric terminals. This component is called the *graphical user interface* or (in Microsoft's terminology) *user service*.

2. The component that engages in actual processing: This contains the application logic that represents the business rules. This component is called the *business services* or *process server*.
3. The component that accesses data: This part contains procedures that access data. It contains, therefore, the structure of the necessary database(s). This component is called the *data server*.

The application of the client–server model to these three components of an application leads to a model called the three-tier client–server model, each stage communicating only with its immediate neighbours (see Figure 1.8):

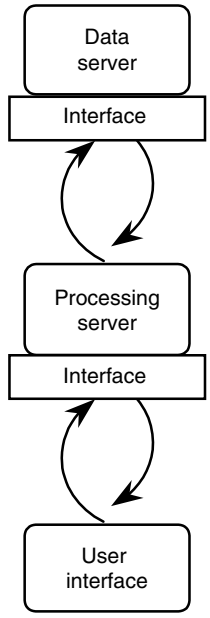

Figure 1.8 The three-tier application model

- The ground floor is formed from the graphical user interface. This is the entry point into the application.
- The first floor is formed of the business services.
- A client–server relation exists between the graphical user interface which plays the role of client and the business services which play the role of server.
- In order to be a server, the process server must have an interface that is completely defined and that describes the services that are offered.

- The data server constitutes the second floor.
- A client–server relation exists between the process server which plays the part of the client and the data server which plays the part of the server.
- In order to be a server, the data server component must have a completely defined interface that describes the data access services offered.

It should be noted that the process server part plays two roles: That of a server in relation to the graphical user interface and that of client in relation to the data server.

The decomposition of an application following the three parts described above has multiple advantages:

- Each part is physically independent of the others. Thus, the three stages can run on three different machines, communication between the stages being performed using middleware.
- Programming and maintenance of each of the stages can be performed independently of the other stages as long as the interface is not changed.

The functional separation leads to making the core code of the application (the process server) independent of the structure and the location of its data. It is also independent of the way in which the data are provided by the user.

At this point, it is interesting to examine the middleware technologies that allow the implementation of the client–server model as well as the integration of distributed applications.

1.3 Different Sorts of Middleware

The goal of integration is to reunite the elements which are distributed over several machines across a network in order to offer certain services. Integration and distribution are two related but opposing concepts. In programming, two sorts of element can be distributed and/or integrated: data and programs. The technologies allowing their integration are designated by the term middleware.

1.3.1 Integration of Data

There are two sorts of middleware for integrating data: On the one hand, those techniques that allow remote access to data (called gateways), and replication techniques on the other.

Gateways

The aim of gateways is to give the user the impression that all their data resides in a single relational database, although there is no such unique database. The real data are located in different databases and are retrieved using gateways in such a way that the retrieval is transparent to the user. The advantages are many:

- A single database image does away with the handling of user transactions.
- Data can be moved without affecting the user's application since it has a logical and not a physical view of the data.
- A relational database allows the use of the SQL language (Structured Query Language – the standard language for databases).

Many products exist on the market. ODBC from Microsoft, JDBC from Sun and Open Gateways from Oracle can be mentioned.

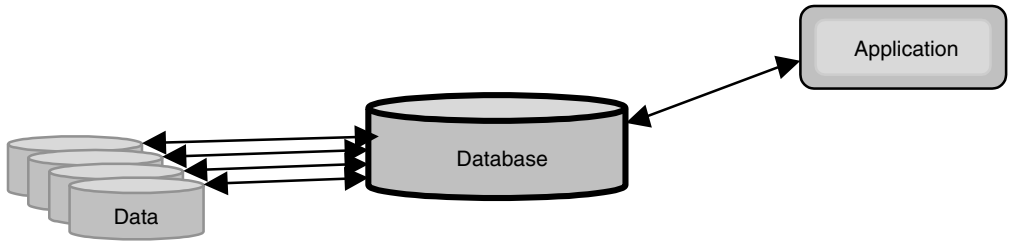

Figure 1.9 *Gateways for accessing data*

Replication

The replication mechanism consists of copying data from several databases. Thus a single piece of data exists in many copies, on different machines over the network. This technique is justified because it improves the performance of a system. The problem is to ensure that every modification on a data instance is immediately carried out on all other instances of this data. This is implemented by the replication technology. Typically, this function is an integral part of databases that allow replication.

The replication of data is a technology that is different from data distribution. Replication uses distribution, but a replicated database is different from a distributed database. In the former case, the same data exists in several instances while in the second, each data item is unique but the set of data is divided into groups that are distributed across several machines.

1.3.2 Integration of Processing: System Level

When two software components communicate, there must exist a link between them and each must have an appropriate interface to connect to the link. The addition of an application to an information environment holding n components, can lead to the construction of n communication connections and $2n$ application interfaces. One way to solve this problem is to introduce the concept of a unique communication bus or middleware, to which the applications connect using a clearly defined interface. The result of one such approach is shown in Figure 1.10. It can be seen that all the connections are grouped into a single system for the exchange of data.

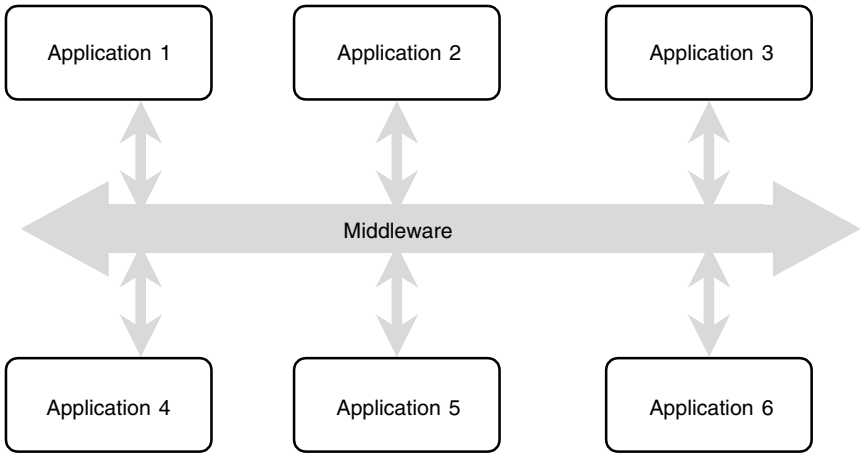

Figure 1.10 *Middleware or communications bus for distributed applications*

Such an architecture requires us to assume that the communications bus (or middleware) allowing the exchange of data between two or more applications offers a certain number of services.

The following can be mentioned:

- The availability of middleware on different machines: The information environment of every business is heterogeneous, that is to say that it is composed of different types and makes of machine. Applications running on these machines must be able to communicate.
- Reliable transfer: When the sending application hands a message to the middleware, it must be sure that the destination will receive it once and once only. This must remain true in the face of a network crash or the crash of one of its machines.

- Traffic adaptation: The rate of the communication bus must be able to sustain an augmentation in the traffic due to the addition of applications. The capacity of the middleware to adapt itself to the change (variation in the number of applications, number and type of machines) is essential since it forms the skeleton of an application system.
- The diversity of communication structures: An application might need to communicate with other applications or send the same message to *n* destinations. In the latter case, it is desirable that the sender can put one copy of the message on the bus and that the bus will take on the task of sending the message to each of its designated destinations.
- The use of names: The application sending a message denotes the destination not by its physical address but by a name. It is up to the transmission system to convert this name into a physical address.

  This service allows the movement of application from one machine to other machines without incurring consequences for the applications that communicate with it.
- The concept of the transaction: The transaction concepts states that several entities (e.g. applications) belong to an application, each one completes its task or none of them do.

Let us assume, for example, that to organize a trip, a travel agent uses two applications. The first application performs flight reservations and the second performs hotel reservations. It is assumed that if it is not possible to reserve a flight to a given destination, it is useless to reserve a hotel there. In the same fashion, the traveller does not want to travel to a town if there is no hotel room for them.

Thus the "travel" transaction will only be completed if the requests for the flight and the hotel are satisfied. If one of them is not satisfied, no reservation is made and the transaction representing the trip is aborted.

Position of Middleware in the OSI (Open Systems Interconnection) Model

Middleware constitutes a communication structure that is independent of operating systems and the nature of transmission systems. In the OSI model which defines the different communication levels between information systems, middleware is located right at the top. It defines the communication protocols between applications (Figure 1.11).

This inter-application communication structure rests on communication structures at lower levels such as network protocols (TCP/IP, DECnet, SNA or OSI) and/or mechanisms offered by operating systems (e.g. interrupt processing).

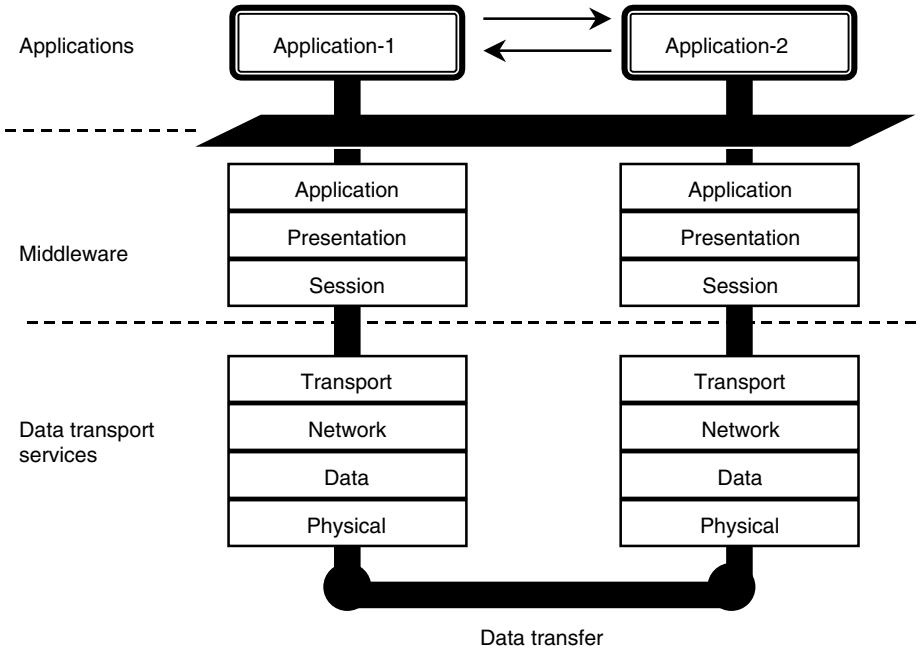

Figure 1.11 *Location of middleware in the OSI model*

Middleware Using Remote Procedure Call

When a new application is designed and when it is desired to distribute it over a number of platforms, there appears the problem of the unit of distribution. When it is desired to decompose an application into distributable elements, there occurs the task of defining the characteristics of these elements. Their "size" must represent the resulting processing time in relation to the transmission time for the request which asks for its execution. This "size" represents the granularity level of the distributed entity.

If we refer to the existing programming elements, the concept of procedure (or function) appears as a good candidate for distribution. This concept is known and understood by all and represents a very precise functional entity with a completely defined interface.

In traditional programming, every application is composed of a main program body and a set of procedures (Figure 1.12). The client–server model can perfectly be applied within this scheme. The main program which calls the procedures appears as the client for each of them and the server consists of the set of procedures. Each procedure offers a service and the server is formed from the set of these services. The interface to a service is exactly the

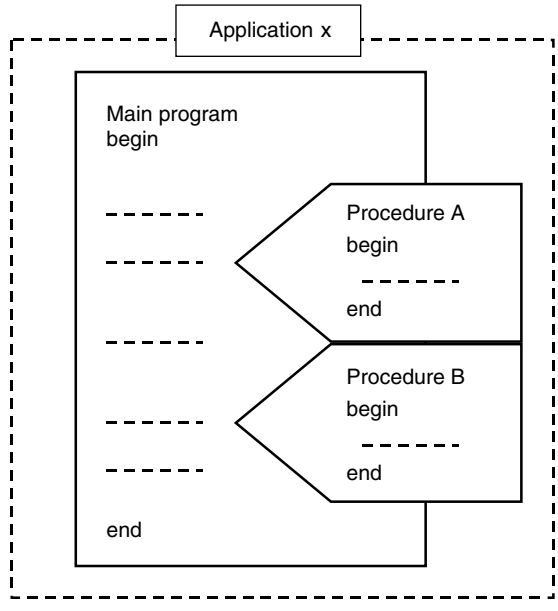

Figure 1.12 *An application is composed of a main program and a set of procedures*

interface to a procedure, that is its name and set of parameters. The server interface is formed of the set of interfaces of the procedures that it contains.

This approach shows the importance of the server interface. This is completely defined. Our goal is, however, to make this set of procedures (the sever) totally independent of the main program (the client). This independence clearly permits other clients to use the services of the server. These other clients need not be written in the same language as the server and must therefore communicate with it. This leads to the description of the server interface in a language that is not a language used to program clients and servers. This language is called IDL (Interface Definition Language).

In the distributed procedures model, the client calls the procedures that compose the server as if they were local to the client. In fact, they can be on any machine on the network (Figure 1.13). The middleware that supports this communication between client and server is called Remote Procedure Call middleware or RPC middleware.

At the client level, there is no difference between the call of a remote procedure and the call of a local one. This means that the code necessary for the preparation of the request message is external to the client. In the message-oriented middleware, this code forms an integral part of the client and must therefore be written by the application programmer. Here, this piece of

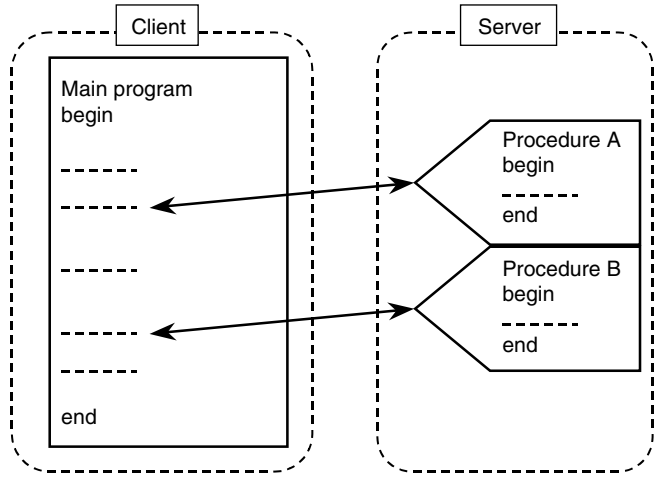

Figure 1.13 *The client–server model applied to distributed procedures*

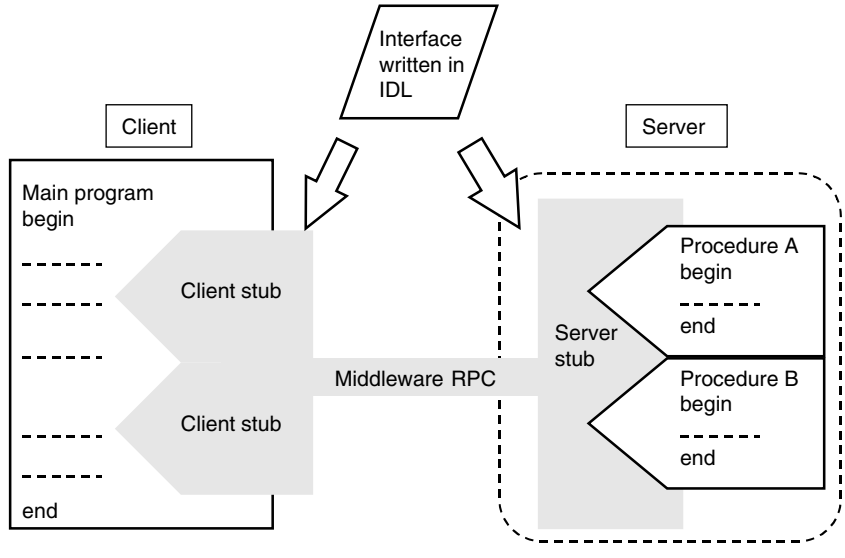

Figure 1.14 *In RPC middleware, communications code is automatically generated*

code is automatically generated from the IDL language which describes the server's interface used by the client (Figure 1.14). The piece of code associated with the client is called the *client stub*, and that associated with the server is called the *server stub*.

Principal Characteristics

RPC middleware can be characterized as follows:

- The client and server code is independent of the communication system. The client does not know if the procedure is local or remote.
- The client code does not have to prepare messages nor to locate the server. This work is done by the RPC middleware.
- The dialogue system is totally external to the client and server. It is described in a specific language called IDL from which is automatically generated the code necessary for communication.
- The communication structure is constructed when the code is compiled. It is therefore completely defined before execution time.
- Communications are synchronous. After having made its procedure call, the client program waits for the result. Only when the result arrives does it resume processing.
- RPC technology is completely standard. Standardization includes the IDL language as well as all the services necessary for communication.
- Many vendors today offer products that comply with the standards. Therefore, it is possible to combine different products.

RPC technology exists today as products that are very stable. However, its success is not as assured as message-oriented middleware. Its basic principle, procedure call, seems to be at too low a level. So, during the designing of complex information systems, aspects of distribution must be considered very early in the specification phase. Now, the concept of a procedure only appears very late in the design phase. Using RPC middleware as a distribution system implies modification to the currently used methods for specification and design by introducing the concept of server at the highest possible level.

If the characteristics of RPC are compared with those desired of a communications bus (see Section 1.3.2), it can be seen that this technology offers everything except the following:

- Reliability of transfer: If for any reason, the server or the network does not work, the message will never be delivered and will be lost. Error handling or crashes are entirely left in the hands of the client's code.
- The transaction concept: Current standard RPC technology does not offer transactions. This is a very serious limitation and for this reason, some

software vendors are offering non-standard products based on RPC which implement the concept of the transaction.

- Message passing: The communications structure in RPC is one-to-one and not one-to-many. This means that a client can only talk to a single server at a time during a request.

RPC technology is available through products from several manufacturers thanks to the OSF/DCE (Open Software Foundation/Distributed Computing Environment) standard. Amongst the other middleware technologies, it is located as the lowest communication structure.

Object-oriented Middleware: CORBA and COM

RPC middleware uses, as unit of distribution, the concept of the procedure. This concept comes from programming and, unfortunately, is not present at the analysis level, that is at the level where modelling takes place. This limitation has led to research on identifying an entity that would be at a sufficiently high level to be used in modelling and be sufficiently close to programming concepts to be easily translated into a programming language. Such an entity exists. It offers a language that is common to the user and to the computer scientist. It is called an *object*.

In brief (but see Appendix A for more details), an object has a name, has attributes that define its state and operations that describe its behaviour. Objects belong to the real world. Thus, for example, a bank account can be considered an object. Its number is its name, its possible attributes can be its balance and the currency in which it is expressed, and its operations can be opening, deposit, withdrawal, obtaining the balance or account closure. The use of object technology allows complex information applications to be modelled by combining objects using a collection of static and dynamic relationships. In this approach, an application looks like a set of co-operating objects (Figure 1.15). From this fact, the object becomes the identified unit of distribution.

Object technology has very interesting properties. In particular:

- The concept of encapsulation: This concept allows the separation of the external aspects of an object, which constitute its interface (attribute names and operation names), from the internal aspects (way in which attributes and operations are implemented). The external aspect is defined during the modelling phase and the internal is specified during the programming phase. This concept allows one to talk of objects without bothering about how they are implemented.

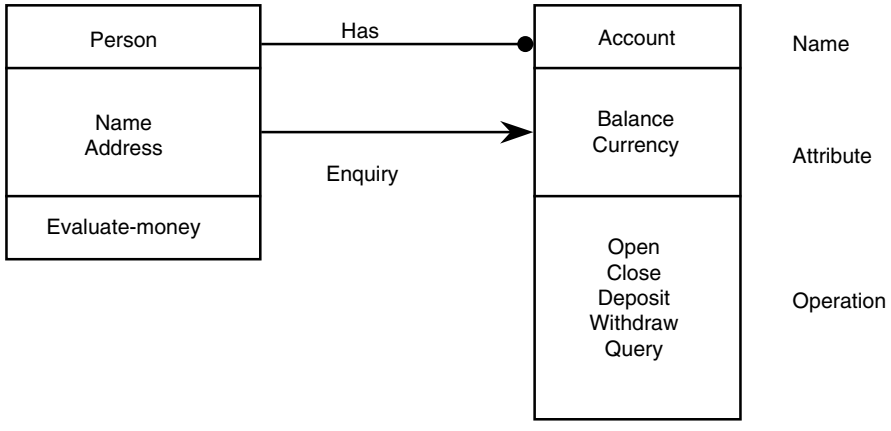

Figure 1.15 *Example of an application composed of two objects, person and account*

- An object system appears as a list of interfaces behind which is the code associated with the attributes and operations of objects. It is also possible to modify the code independently of the interfaces. This is a fundamental point in object technology.
- The concept of inheritance. An object can inherit characteristics (attributes or operations) from another object. This mechanism makes models more concise and eases software reuse.

If it is assumed that objects are distributed across the network, inter-object communication corresponds to the request for the execution of an operation on an object (the server) by another object (the client). This request is implemented using a specific communications bus called *object-oriented middleware* (or ORB, the *Object Request Broker*).

This model provides functions similar to RPC technology. Thus the client object does not know the location of the server object and the client does not have to construct the request message.

Communication between client and server objects can be defined in a static or dynamic fashion:

- Static communication is performed in the same way as in RPC. This form of communication is described in a standardized object-oriented language called OMG IDL (Object Management Group Interface Definition Language). Client and server stubs are generated by the OMG IDL code and they allow, respectively, the client and the server object to be connected to the middleware object.

- Dynamic communication is established by the client at runtime. The client can interrogate the middleware object to find the interfaces of the objects available on the network. The chosen interface server has no way of knowing if the request that it received was generated in a static or dynamic fashion.

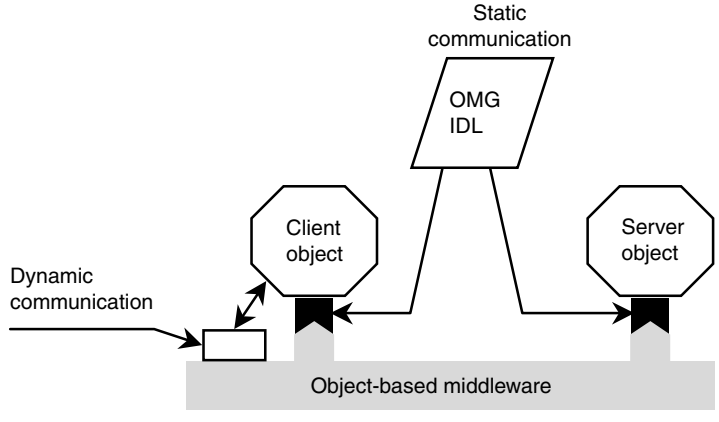

Figure 1.16 *Object-based middleware allows communications established in a static fashion*

Object-oriented middleware uses the concept of interface which has the following properties:

- An interface represents the services offered by the server object. These services are thus directly associated with the object's operations.
- It permits the generation of new interfaces using inheritance. The infrastructure of an object-oriented information system is formed from the set of interfaces connected to the communications bus. Their update is facilitated by the inheritance mechanism that allows the introduction of new objects while protecting the old ones.
- An interface can be associated with one or more services. Let us recall that an interface describes a set of services but does not specify how or by whom these services are performed. This decoupling of service description and the service proper allows the following:
  - The possibility that there will be several implementations of a single service. Thus there is no constraint on the server itself. In particular, it can be written in something other than an object-oriented language.
  - The connection of one server to several interfaces. The services offered by a server can appear in different interfaces. Rather than duplicating

the server's code, this last option can be activated by different interfaces.

– The conversion of an existing application into a server. To do this, it is enough to connect it to an object interface for which this application can offer one of the services. This approach allows the transformation "on the quiet" of an application environment into an environment composed of objects.

A New View of an Information System

The set of object interfaces in an information system represents the services offered by this system. This leads to a new view of an information system. It is no longer described as a set of applications but as offering a set of services (Figure 1.17). One of the major benefits of object-oriented middleware is that it contains a database that holds the list of all available interfaces. It is therefore possible at any time to consult this list in order to find out what the capabilities of the system are. Every client object can consult it and dynamically determine the new services.

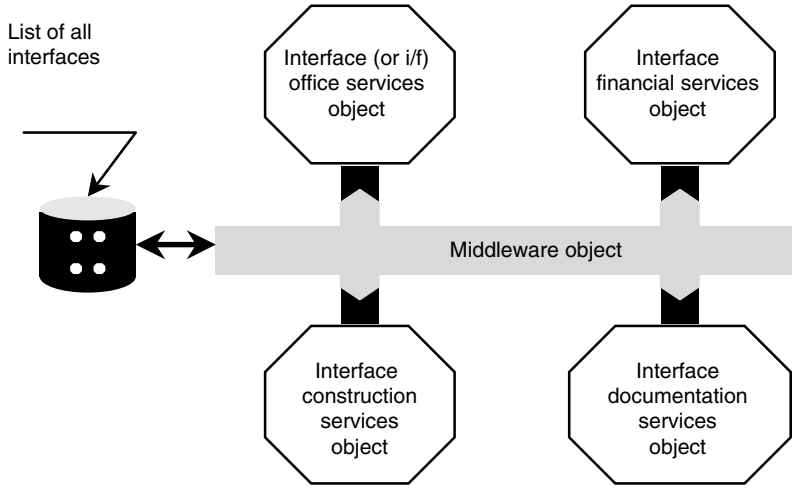

Figure 1.17 *A new approach: an information system is described by the list of offered services*

In such an environment, the addition of software simply means introducing one or more interfaces that describe the services offered by this new software and attaching the software to these interfaces. If the software must access existing services, communication among these services must be described in the OMG IDL language.

Standards and Middleware Objects

In distributed object-oriented middleware, there are two models. The first model has been in existence since 1990, and was established as an international standard by OMG. It is called CORBA (Common Object Request Broker Architecture). The second model was proposed by Microsoft and is called DCOM (Distributed Component Object Model).

Several products implement the CORBA standard. BEAObjectBroker from BEA Systems, DSOM (Distributed System Object Model) from IBM, ORBIX from Iona Technologies and ORBPlus from Hewlett-Packard. Some of these products have become very mature thanks to their availability since the start of the 1990s.

The Microsoft product is more recent (1996). In addition to running on Windows platforms, it also runs on UNIX machines. So, the DCOM model from Microsoft tries to position itself as an alternative to CORBA.

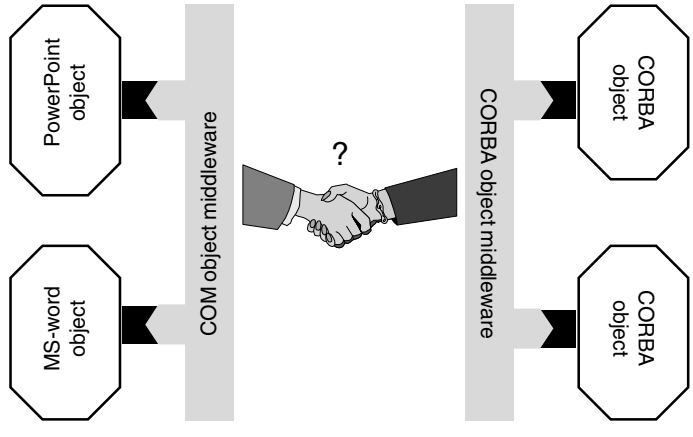

Figure 1.18 *Object-based middleware is divided into two worlds. Will they understand each other?*

From the above, it would appear that the CORBA model runs on all major platforms, the Windows PC included, but the latter is dominated by the Microsoft object model. So, all the software in this group is based on the COM (Component Object Model) which is now distributed. To allow the CORBA and COM models to communicate, their specifications have been compiled by OMG.

In the CORBA world, the situation developed rapidly thanks to the great influence of the Java language. The problems of inter-operability between products was solved by standard 2.0 which specifies the communication

protocol between CORBA agents. The new 3.0 standard published in 1998 concentrates on the idea of software components introduced by Java as the Java Bean concept. This new concept of a component has as its goal making easier the development of new applications. This simplification is welcome in the CORBA world, which is well known for the complexity of its implementation.

Java RMI Middleware

Object-oriented middleware implementation remains complex despite the new tools that have appeared on the market. On the other hand, object reuse based on libraries is not very common; this is for the reason that it is not very practicable. Thus, in order to correct these deficiencies a new higher-level concept was introduced called the component.

The concept of a component appeared in the Java language where it is called a Java Bean. A Java Bean can be seen as a container formed from one or more basic objects and it offers a well-defined interface. These components form the basic blocks that can be assembled to construct Java applications that are called applets or servlets.

In the Java language, the basic unit is the component. A Java application can be formed from one or more components and can be distributed across a network. The middleware allowing components to be integrated is called RMI (Remote Method Invocation). The resulting architecture is very similar to that obtained using objects and their middleware. In both cases, the concept of the stub is used to connect to the middleware. In the same way, RMI

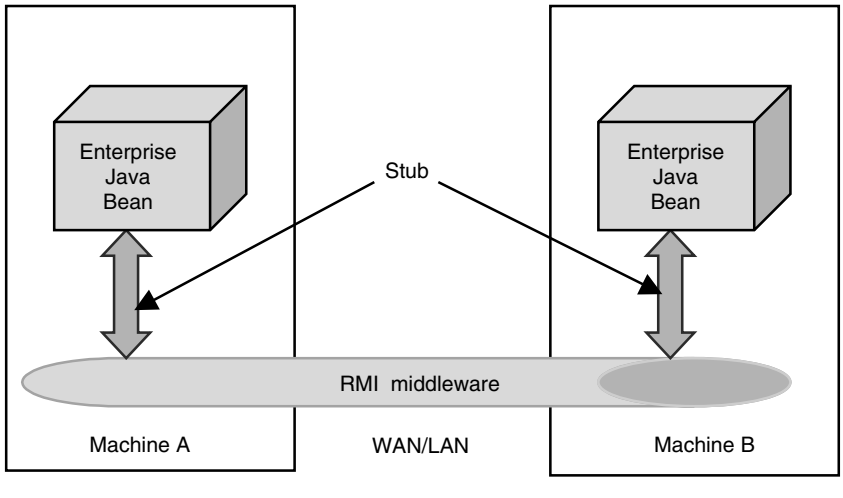

Figure 1.19 *Java RMI Middleware*

middleware requires the execution of a method on a remote object. During this, it transfers activation parameters from the method to the network.

The big difference between object-oriented middlewares lies in the fact that RMI can transfer a component as a parameter. Thus, with Java, it is not the only data that can be transferred across the network but also code (or program). Objects, in CORBA and DCOM technologies, are fixed. They remain on the machine where they were installed. With Java RMI, components can be exchanged for other components, and this allows code mobility.

For more detail on Java technology, the reader is invited to read Chapter 6.

Middleware for Business-internal Applications: MOM

MOM, or message-oriented middleware, forms the technology of choice for implementing the communications bus concept for several applications to allow them to exchange messages. For several years, object technologies were also used to connect applications. However, the fact that they operate in synchronous mode means that applications are always in a run state. This represents a constraint that is difficult to satisfy in production environments. Unavailability of a destination application implies the loss of a message which is not acceptable in a business system.

MOM technology, asynchronous by nature, supports the decoupling of intercommunicating applications. Messages are stored while they are awaiting delivery. The temporary unavailability of an application does not imply message loss. The high level of reliability and the ease of handling in a production environment have convinced designers to use this technology for the integration of business applications.

MOM technologies have developed a great deal of late; they have developed to such a point that one can talk about the appearance of new generation of product.

The old generation used distributed structure that can be represented as a communication bus. This bus solved the problem at the level of message exchange. Products based on this concept have been available on the market for several years. They have a high level of maturity and represent a highly reliable technology. They are very often used in information systems for industrial fabrication. For example, BMW's production lines use the BEAmessageQ product from BEA Systems to handle production. IBM also offers a message-oriented middleware product, called MQSeries, that is used in inter-bank exchanges.

The new generation bases itself on a centralized architecture offering a set of services at a higher level. It allows the semantic exchange level to be reached.

The terms *hub and spoke* are often used to describe it. A good example of this new approach is the Advanced Queuing product from Oracle which is used on the *Amazon.com* Internet server to transfer orders from the Internet server to the *back-office* software. In order to show the characteristics of these two generations of MOM better, we will consider them separately, starting with the old technology. The new generation will be described in Section 1.3.3.

Principal Characteristics

This message-oriented middleware technology is decentralized (each application has one message queue for input and one for output) and is characterized by the fact that it is not standardized. The best way to state its principal characteristics is to examine one of the most advanced. If we refer to BEAmessageQ, the following functions are available:

- Asynchronous or synchronous transmission of messages. BEAmessageQ allows applications to communicate in an independent or inter-dependent fashion. The asynchronous message exchange mechanism allows the sending application (the client) to put its message in its output message queue and continue with its processing. The middleware transmits the message to the input queue of the receiving application (the server) and, when the latter is available, it reads the message and processes it. Thus client and server function as expected.

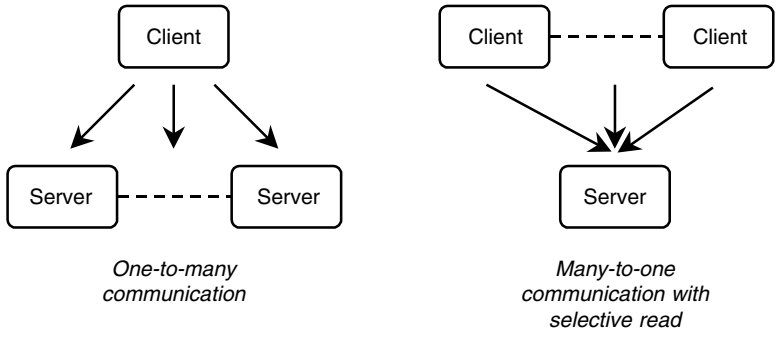

Figure 1.20 Examples of communication structures

In synchronous communication mode, when the client has put its message in its output message queue, it must wait for a reply. No parallel processing is possible between the client and server:

- Guaranteed message delivery: BEAmessageQ allows clients to label messages so as to guarantee their delivery. Each message thus labelled is copied to disk so that it is not lost. The message remains on disk while the

original has not been delievered to its destination. In the case of a system, server or network failure, the middleware automatically re-sends the message until the server receives it. It is only then that the disk copy is deleted. This system does not predict when the message will be processed by the server but it guarantees that it will be processed.

- Availability on many platforms: The BEAmessageQ product is available on more than 20 platforms. These include the Windows NT, UNIX (AIX, HP-UX, Sun), OVMS, IBM MVS, etc., operating systems. Because there are no standards, message-oriented middleware products are difficult to combine. It is therefore important that a product operates on lots of platforms so that it satisfies the needs of its users.
- Selective message reads: BEAmessageQ allows servers to read message in an order different from that of arrival but following certain criteria. These criteria depend on the sender, on the priority level of the message, on its type and its class. They are defined by the user so that the content or the aim of the message can be identified.
- Message broadcast: The BEAmessageQ software contains an operation that allows a client simultaneously to send a message to a group of servers (multicast). The receiving applications are those which have explicitly expressed the desire to receive such messages. Thus, the sender knows neither the number nor the location of the destinations. The separation of the sender and the distribution system allows changes to the latter without affecting the sender's code.

Advantages

Today, one of the advantages determined by message-oriented middleware is its great reliability. The existing products have sufficient maturity to be used in applications that are at the core of a business' operations.

A second advantage is due to the fact that this technology uses traditional programming. In particular, it does not appear to the latest object-oriented programming techniques. This is important because the implementation of message-oriented software implies intervention in the communicating applications at the level of code.

Disadvantages

Applications exchanging messages must construct and interpret these messages.

Thus, message-oriented technology imposes no restriction at the level of message structure. Messages must be constructed by the sending application

and interpreted by the receiver. This implies that the client application must possess code to construct the message and that the server application must have code to decode the message. The code for each of the applications also contains, therefore, communications-related code (Figure 1.21).

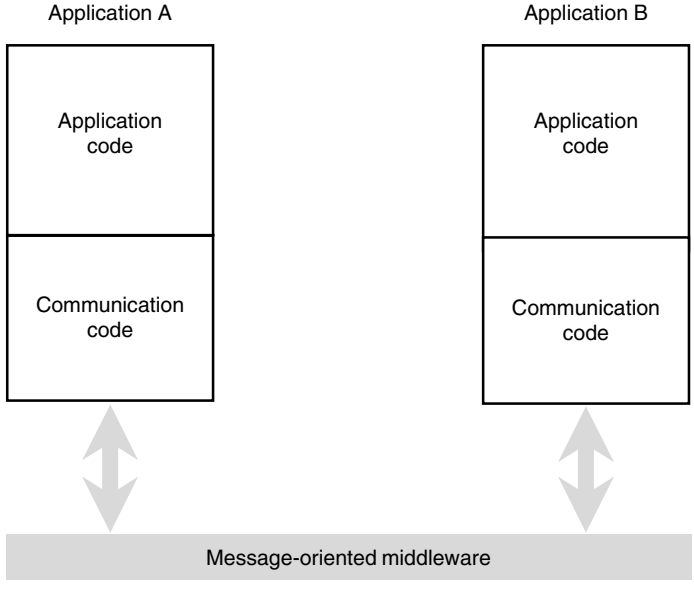

Figure 1.21 *Using message queues, applications contain code for communications*

This is not considered to be elegant from the architectural viewpoint, because every modification in the communication system affects the applications themselves and the application programmer must also know the communication system. This knowledge represents a not inconsiderable expertise. For example, certain types of data are not represented in the same way on different platforms. Current middleware products only perform very limited transcoding and this job rests in the hands of the application programmer.

No standard exists for this technology. This means that a user cannot combine two middleware products each from a different vendor. It is equally impossible to create messages using a standard format in the hope that they will be portable across distinct middleware products. All of this implies that a user must select a product and keep with it. This choice is important because middleware represents the infrastructure of the information system which cannot be easily modified.

Recent Developments

Message-oriented middleware technology is highly dynamic and in each version of each product, new functions are offered. To show this tendency, here are the principal functions offered by version 4.0 of BEAmessageQ:

- *Self-describing messages*: This function includes in the message all the elements for its interpretation. This means that one application can communicate with another without precisely defining the structure of the messages they exchange. Thus, the sender of a message can add data to an existing message or change its length without causing problems for code in the receiving application if the latter does not access the additional data. This system can also transcode data from one format into another when the exchange takes place between different platforms. This newly offered function shows the tendency to simplify matters for application designers, by increasing the flexibility of the communications system and by progressively integrating in it all the functions that are appropriate for it.
- *Naming system*: This system allows the naming of the destination not only by the number of its message queue but also by a symbolic name.
- *Large messages*: The maximum size of message transferred is increased to 4 Mb. This allows the simplification of the exchange of complete files between applications.

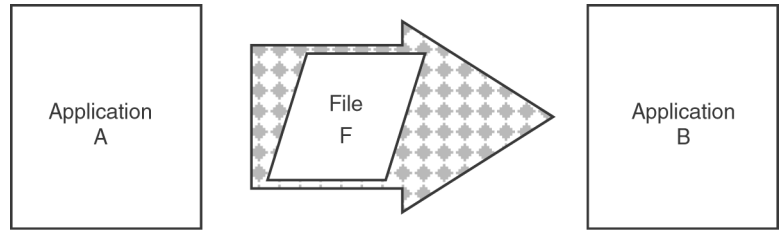

Figure 1.22 Entire files can be exchanged between applications thanks to the maximum message size of 4 Mb

In conclusion, message-queue middleware offers to system designers a flexible asynchronous tool that is reliable, has high performance, and allows them better to solve the problem of how to integrate pre-existing applications. However, the absence of standards and the fact that the application code must contain communications code have led some groups to see that other ways of integrating applications should be found. These thoughts led to the production of a new generation of MOM products that employ the semantic level in message exchange. This new generation is described in Section 1.2.

Middleware for Applications External to a Business: HTTP and XML

Warning: In order to understand this technology, it is assumed that the reader is familiar with the operation of Web servers. If this is not the case, the reader is invited to read Section 1.4 in this chapter. They will then be able to read this section with confidence.

The Internet environment allows the construction of applications using the three-tier client–server model. The user interface stage is formed of one or more HTML pages that are downloaded and then displayed on the user's PC. This page has the address of the processing server stage (which is located on the Web server machine) in the form of a URL (Uniform Resource Locator). It can also transmit requests by adding a set of parameters to a URL. These requests are received by the Web server that transfers them to the applications processing server. This server replies by returning the data in HTML format so that they can be displayed by the browser. This dialogue forms the HTTP (HyperText Transfer Protocol) protocol.

This communication protocol is synchronous and it only transfers characters, a property that allows it to operate in any environment. On the other hand, a URL provides a simple universal medium for denoting an entity on the Internet. From this fact, HTTP is very similar to RPC, but is much simpler. Its only weakness is the lack of standard defining the parameters that can be passed. Microsoft has suggested using XML to describe this data. XML allows data to be put into a form that is easy to transmit and decode. The combination of HTTP and XML is called SOAP (Simple Object Access Protocol). SOAP is not a new technology but a combination of two already existing ones. It simply denotes the use of XML with HTTP.

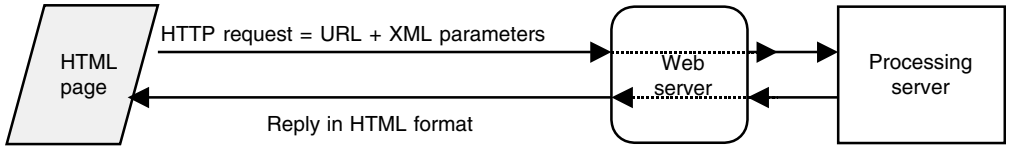

Figure 1.23 *The SOAP protocol*

1.3.3 Integration of Processing: Semantic Level

Hub and Spoke Architecture

The concept of application integration is no longer limited to the system level but also includes the semantic level. It is now possible to talk of business application integration so as to emphasize several aspects, which

are as follows:

- The nature of what is exchanged. Messages are no longer simple character strings but business elements (order form, stock exchange form, etc.).
- The fact that the elements exchanged have business significance implies that they require particular forms of processing. For example, they could be tracked individually, stored in various ways, subjected to the processing associated with data warehousing.
- The exchange data model imposes conversion mechanisms between models.
- Integration of two applications is the consequence of a business analysis. The need to implement a new business process includes a sequence of tasks that are implemented as a sequence of programs.

These new constraints induce functions that are very "heavy" such as task sequencing or transformation of data models. Because these functions make use of complex software, it is seemed more efficient to implement this new type of middleware using a centralized architecture of the hub and spoke type. The hub represents the centre of the activity and the spokes represent the links which connect the hub to applications.

Hub

In this scheme, the functions offered by the hub are as follows:

- The handling of message queues: They are grouped at the level of the hub and not decentralized at the level of each application. A unique queue handler also allows the use of elaborate communications structure: from one-to-one (question/response type) or one-to-many (broadcast or publish/subscribe type).
- The workflow function: This function is activated by each message sent in order to decide its destination. The destination is defined either by
  - The message header: this contains the name of the destination application.
  - The value of certain parameters in the body of the message: for example, if the message represents a order form, following the value of this, the message will be sent to different people.
  - The workflow to which the message belongs: this message belongs to a business process which is described in a table to which the routing software has access. This table contains the list of steps in the workflow and therefore the name of the destination application.

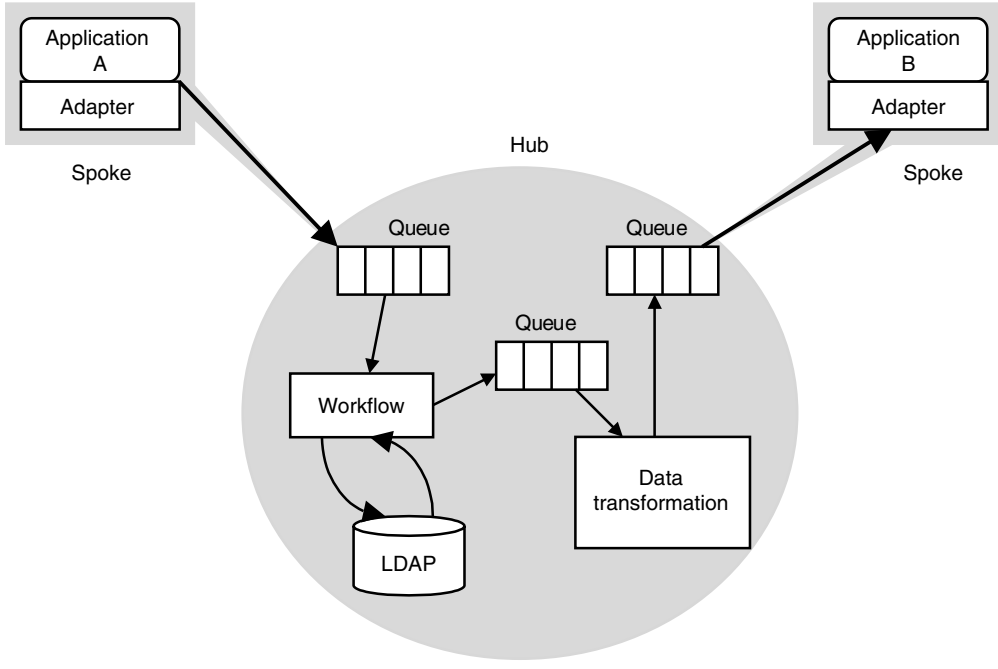

Figure 1.24 *Hub and Spoke architecture*

- Security and access rights function: In this kind of architecture, it is important to verify that a given application does really have the access rights required to send message of a certain type to another application. The access rights of each application and/or of each user are stored in a database whose standard representation is LDAP (Lightweight Data Access Protocol, a standard protocol for describing access rights in networks). This database is consulted before routing the message to the data transformation function's message queue.
- The data transformation function: When a message is placed in its input queue, this program is activated. It knows the transformation function for transforming the sending application's data model into the one used by the receiving application, and it generates a new message which it sends to the input queue of the destination application.

Spoke

Spokes constitute the link between the hub and the application. This link is formed of two parts: the network protocol part which allows the circulation of messages and the adapter which is ad hoc software for connecting applications. This complex software is described in the following section.

Figure 1.25 MS message format

In the hub and spoke structures, the network protocol does not matter. It must allow the reading of messages from and the sending of messages to the message queues that are located most often in a database. On the other hand, the structure of the messages must be specified. One attempt at a specification, initiated by Sun, seems to hold the consensus of the major vendors. It is called JMS (Java Message Service, a standard for asynchronous messages). JMS is characterized by the fact that its message has the following three parts:

- The header: contains information allowing the identification and routing of the message.
- Properties: this field allows the addition of information specific to the sending application.
- Message body: five types of data format are possible:
  - StreamMessage: contains primitive Java values;
  - MapMessage: sequence of (name, value) pairs;
  - TextMessage: text in XML format;
  - ObjectMessage: serialized Java object;
  - ByteMessage: sequence of uninterpreted bytes.

In the message body, different data formats are possible but the consensus favours XML. XML is an extensible language for the description of data. An XML document is formed of labels surrounding each data element (before and after). Associated with this document is a DTD (Document Type Definition) file which contains the description of the labels. This file (Figure 1.26) allows the unambiguous interpretation of the XML document. The flexibility and the power of expression in XML make it the language of choice for the exchange of information between applications.

Adapters

Two applications are said to be integrated when they are able to exchange information between each other. The use of exchange middleware implies that these applications are connected to it. It offers a well-defined interface which must be used by the applications in order to communicate. The applications having not been designed for using middleware, two solutions

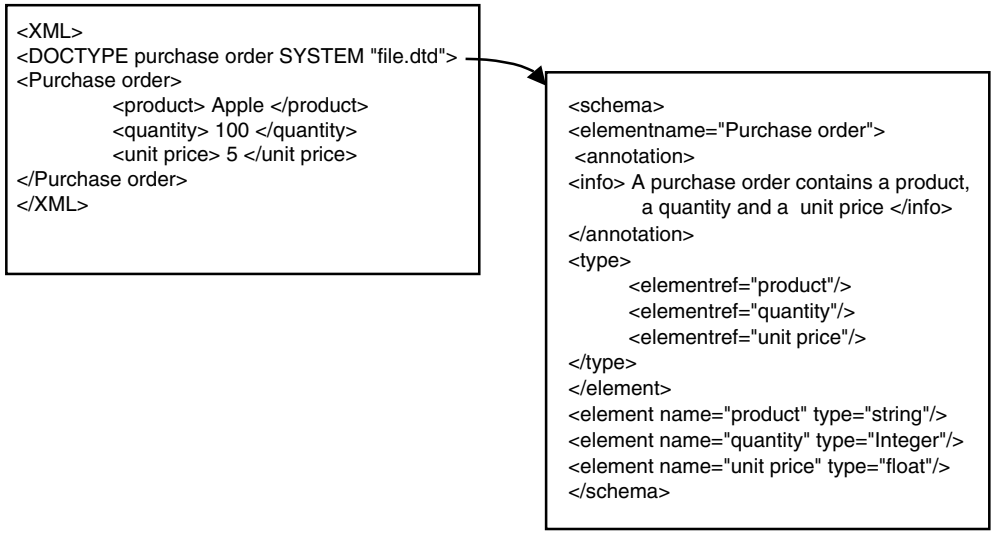

Figure 1.26 XML and DTD files

are conceivable:

- Modify the application code so that they are able to use the middleware.
- Associate the applications with an external software module that knows how to talk to the middleware as well as to the application.

The first solution is rarely used because very often the software was bought and in this case it is impossible to modify. In the case in which they were developed by the company, the document is very often unavailable and no-one is ready to take the risk of modifying something which still works.

The second solution is almost always preferred and it leads to the development of a software module called an adapter.

Structure of an Adapter

An adapter is a piece of software that was developed especially to connect a given application to some particular middleware. It must allow the application either to initialize a request or to receive a request. It must therefore provide the following functions:

- Communicate with the application using its API. Each large piece of software on the market has one or more well-defined APIs. For example, the SAP business software has two APIs: BAPI and iDoc.

- When the application sends a request to another application, the adapter must receive the type of request and the associated data through the application API. It must then construct the message in the middleware format (e.g. JMS and XML) and place the message in the message queue in the hub.

- When a message comes from the hub, the adapter must read it and transform it into one or more function calls that are offered by the application API. This task is generally complex to be implemented by the developer because it requires a very good knowledge of the application.

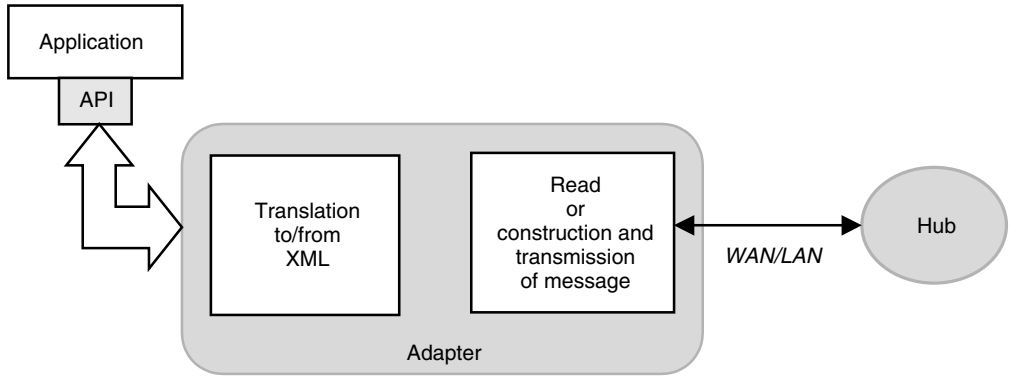

Figure 1.27 Structure of an adapter

The integration of the application implies the development of one adapter per application before integration. This is not, however, sufficient because if an application communicates with several other applications, the messages that it will exchange will vary according to the destination. It will therefore be necessary that the adapter knows how to handle each of them. This means that to say that an adapter exists for a given application does not have much sense. It is also necessary to specify the dialogue for which it was designed.

Data Transformation

Let us recall first of all that the data transformation function is different from that of data encoding. The latter is concerned with the way in which data, for example a whole number, is represented in binary. Here, all data being coded as character strings, the transcoding problem is eliminated.

The messages that are exchanged represent business entities such as an order form or a stock form. Data appearing in these business entities must

be understandable by the communicating applications. One necessary condition is that the data are structured in an identical fashion. For example, for one application, the name of a person might be represented using four fields (title, family name, middle name and first name), and in another application, two fields are deemed enough (family name and first name). Some data can be missing but understood. Thus a date for payment might be in one case deduced from the delivery date (e.g. end of the same month) and in the other case, it must be calculated.

The aim of the transformation function is to make the correspondence between the data models in the two communicating applications. This job can become very complex and tools exist on the market to make it easier (e.g. the Mercator product from the Mercator company, market leader in data transformation).

Two approaches can be considered when it is necessary to integrate several applications.

The first approach consists of converting the data into a common model. This implies two conversions: a conversion from the sending application's model to the common model, and a second conversion from the common model to the destination application's. The common model must not be confused with the general corporate data model. The latter exists only very rarely and if it existed, the common communications model would be, at best, only a subset of the general model.

This approach has the advantage of separating the data models of each application. Thus it is possible to change an application's model without affecting communications with the other applications. It has, however, the disadvantage that it is necessary to perform two conversions for each exchange.

The second approach consists of implementing a one-to-one mapping between each application model. The transformation is done between the sending application's data model and that of the receiver. If an application communicates with n others, n transformations will be necessary. In this scheme, there is no common data model to be defined. However, in terms of the transformation links, this approach can lead to a spaghetti system.

Figure 1.28 shows two applications, A and A′, which communicate with three other applications, X, Y and Z. The use of a common data model leads to a star-shaped model (the number of transformations is optimized), and the other model leads to a many-to-many scheme.

The Open Applications Group (OAG) has defined specifications to integrate applications in particular at the level of data models. Currently, more than 100 models in XML format are available. For the OAG, a message is a

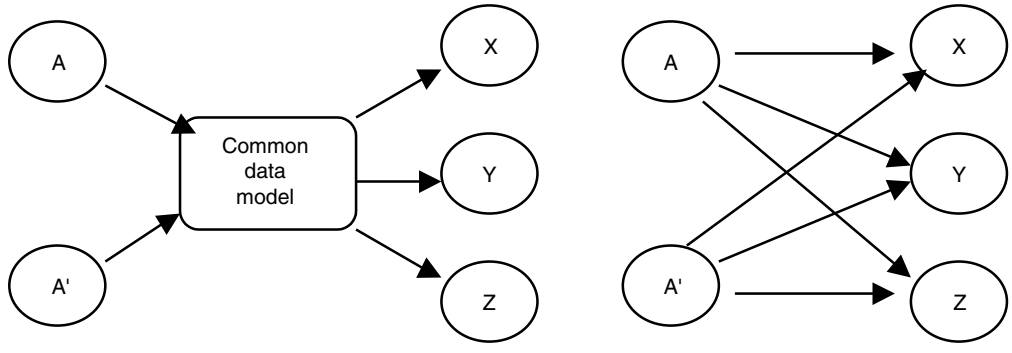

Figure 1.28 Two schemata for transforming data models

Business Object Document (BOD) which is described in XML. Some standard formats exist for each industry. They are strongly recommended since they represent a common data model that can be used in B2B commerce.

Workflow

Application integration always corresponds to a business need. This is defined by an objective and the process that will achieve it. The process consists of the steps to which their correspond applications and documents exchanged by these applications. Experience shows that when a company wants to adapt to market changes, that will change the workflow and not the tasks themselves.

The design of an integrating system must be accomplished in such a fashion that workflow descriptions are external to applications. Thus, a message sent by one application must not contain the name of the destination but, rather the name of the stream to which this message belongs. The routing system, which knows the different streams, decides on the destination. This approach allows description in one and only one place, in a centralized fashion, at the level of the hub and the workflow connected to a given type of processing. The update of such a stream is facilitated because it is not affected by applications but only by the routing software which provides the modification function for the tasks or for the workflow.

Security Functions

When two applications want to communicate across an information network such as Internet, the issue of security is raised. This problem has several aspects: client identification, access rights determination and the transfer of the exchanged data.

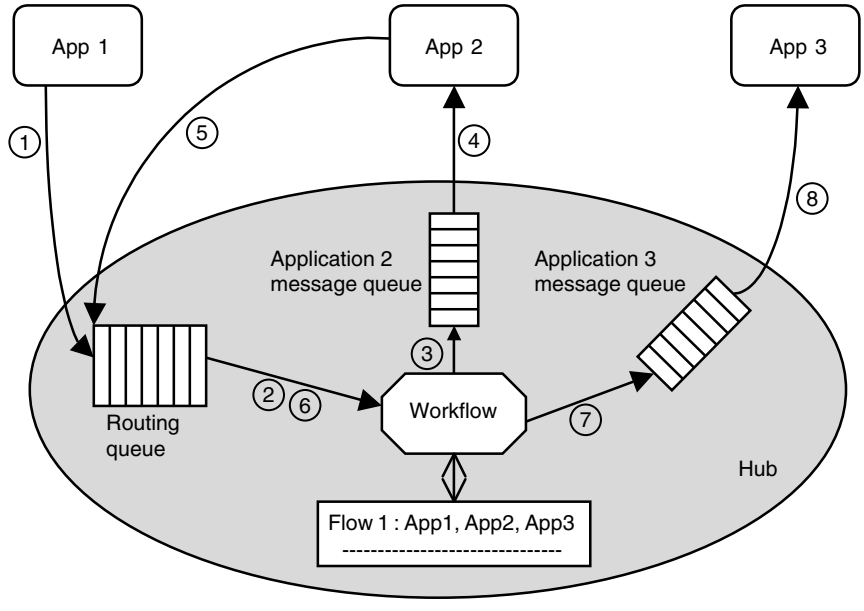

Figure 1.29 *The chaining of tasks at the hub level*

Client Identification

A client generally authenticates itself using a password. This can be improved by mechanisms requiring two items for identification. This is the case, for example, with smart cards which assume that the user has a card and that they know the associated secret code. In distributed architectures, a client can have access to several applications and must identify themselves to each one of them, a process that can become tiresome. Technologies have been designed to permit the user to identify themselves once and once only and to be able to access all the resources to which they are authorized. The Kerberos system at MIT represents an implementation of this type of technology, called single sign-on.

Client Access Rights

Having identified the client, the server must know its access rights for the variously available resources. To do this, the server can either use access control lists (e.g. each resource is associated with a list of authorized clients) or a database containing the clients' rights in the form of the LDAP standard.

Data Transfer

Communications security and Internet transactions must be secured. They are implemented by combining cryptographic software and encoding using

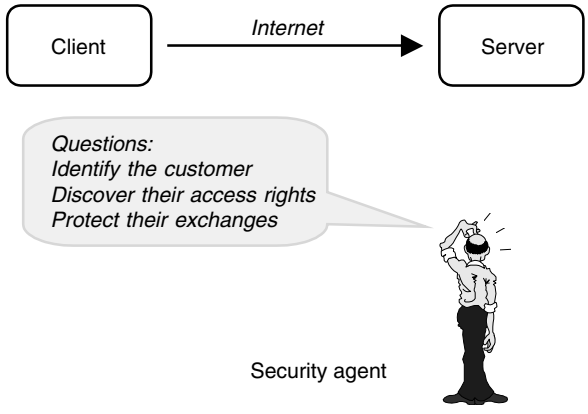

Figure 1.30 The problems of Internet security

keys (PKI, or Public Key Intrastructure). PKI uses an architecture that includes digital certificates, cryptography and certification authorities. The introduction of PKI in a company means assigning certificates to users and to servers, providing cryptographic software, offering connections to certificate servers, making available tools to generate, renew or cancel certificates. Of the existing technologies, the following can be cited:

- SSL: A standardized security mechanism that is defined at the level of network connections between two communicating machines. This technique is the most often used because it is the easiest to put in place. It ensures confidentiality and reliability of exchanges, protects against unauthorized eavesdropping, the counterfitting for the alteration of messages. This protocol allows the client and the server to authenticate themselves, to negotitate an encryption algorithm and keys before any data exchange.
- S/MIME (Secure/Multipurpose Internet Mail Extension): This is an extension of the MIME (Mulitpurpose Internet Mail Extension, an Internet standard that defines mail message content types), allowing the sending and reception of electronic messages in complete security and confidentiality. It provides authentication, message integrity, sender anonymity assurance and message content confidentiality.
- RSA-RC4 (a product of RSA Security Inc.): This industrial standard provides rapid encoding of data using a key of 40, 56 or 128 bits. The length of the key determines the level of resistance to external attack.
- The industrial standard SET (Secure Electronic Transaction) for credit card payment over the Internet.

1.4 Middleware and e-Business Architectures

The middle of the 1990s saw the emergence of a new set of technologies called the Internet. The Internet represents a global network of networks allowing computers that are connected to it to communicate with each other. The Internet is interesting for this study because it suggests a new medium for communication between entities that were previously called client and server.

Initially, in order to familiarize ourselves with the Internet, we will briefly consider its architecture (there is more detail to be found in Chapter 8). In the second part, we will see how it can be used in conjunction with middleware technologies to provide very powerful architectures.

1.4.1 Using the Internet

The Internet aspect that interests us here is the World Wide Web (WWW). (The WWW is formed of a set of multimedia documents distributed over thousands of computers, and by a set of tools that allow access to these documents.) The Web allows access to multimedia documents, in particular alphanumeric ones, that are stored on a global network of computers. To gain access to these documents, the user must use a personal computer (PC) and a connection to the telephone network. Their computer must contain a special program called a browser.

This computer connected to the network and wishing to appear as a document server must have a special piece of software called a Web server.

The WWW operates as follows (Figure 1.31):

- Having started the browser program, the user enters the address of a document.
- The browser sends the request to the network, which knows how to locate the destination machine and sends it the name of the document being sought. The protocol used over the network is called HTTP (HyperText Transfer Protocol). It runs on top of the standard TCP/IP network protocol.
- The Web server program retrieves the document from one of its disks and sends it to the machine that requested it.
- On reception of the document, the browser processes it according to its type. If the type is alphanumeric, the document is displayed on the user's screen; if the type is audio, the sound is generated by the sound card in the PC, and so on.

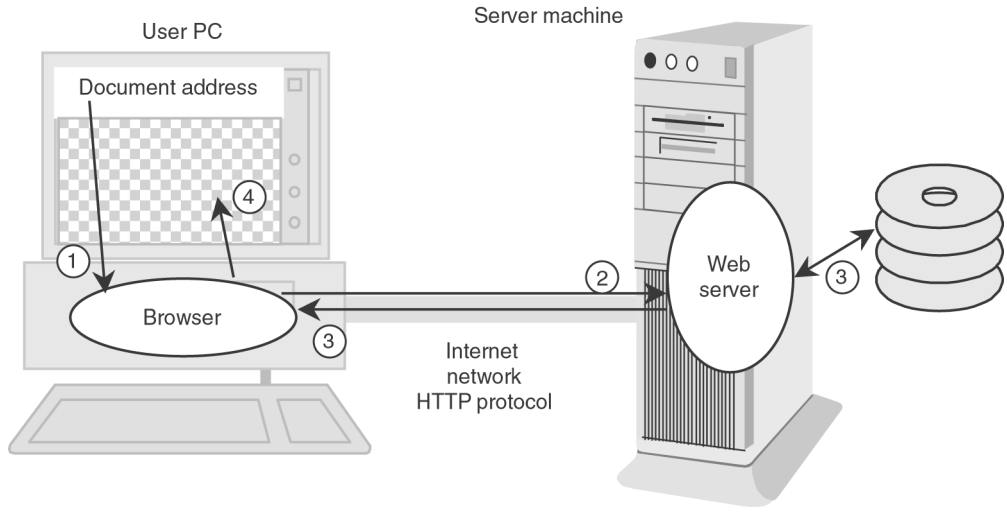

Figure 1.31 *Use of the World Wide Web from a PC*

The mechanism described above quite simply allows the transfer of files (or documents). There are two very important points that emerge:

- Access to a document can be had from anywhere in the world. The only condition is the use of a telephone and a computer.
- Every alphanumeric document is displayed on the user's computer screen. In order to be "displayable" on any type of machine, such documents must be written in a standard language called HTML (HyperText Markup Language). Such documents can contain fields to capture data and to display results. Therefore nothing prevents such a document from being the graphical interface of an application.

It is thus possible, throughout the world, to download the part of an application which forms its interface. In the three-tier client–server model introduced above, this part forms the client's graphical interface stage in the application. There remains, however, communication with the rest of the application, that is with the processing server. To this end, we will see another characteristic of the WWW.

Web Server

The Web server is a piece of software located on a machine connected to the Internet that sends replies to user requests. The requests always refer to

documents or files. The type of document dictates the action of the Web server. It returns all the documents requested of it except those of type program. In this particular case, the active Web server activates the program in question, passing it the parameters that were received in the request and finds the results in order to send them to the client that sent the request.

Thus, in this mode of operation, the WWW allows the execution of code at a distance from the application.

1.4.2 Architectures Combining Internet and Middleware Technologies

The mechanisms of the WWW allow the construction of applications using the client–server model. Let us take the case of a two-tier application composed of the user interface stage and the processing server stage. Let us make two assumptions:

- The interface code is written in HTML and is stored in a file of type alphanumeric, called *interface.html*.
- The processing server is already in executable form and is stored in a file of type program (or binary) called *server.bin*.

The WWW allows use to execute this two-tier application on any machines anywhere. The mechanism is the following:

- It downloads the interface stage, that is the *interface.html* file (see above).
- The browser displays the *interface.html* file on the user's screen. The user has before them the interface to their application which executes on their own machine. It can also capture data and send a request to the *server.bin* document, that is to the processing server of their application.
- The request travels across the network and arrives at the Web server on the machine on which the requested file is stored. This file, containing as it does an executable image, is executed by the Web server which passes its data that formed part of the request. Thus, the processing server part of the application executes on the Web server machine.
- The program's results in the form of an HTML document are returned to the client by the Web server.
- The user's browser displays the results it has received.

This description shows the fact that the PC user can execute the client part (interface) of any application at all. For this, it is enough merely to download it.

In this approach, the HTTP protocol combined with the browser and Web server appear as a particular type of middleware. The interest of this approach can easily be understood. No application need reside on the PC. Interface parts are downloaded on request. The gain in application maintenance is enormous.

In the case of applications organized along the lines of the three-tier model, it is quite possible to use, between the graphic interface and the processing server, the HTTP protocol and, between this and the data server, message-based, RCP or object-based middleware. Thus in Figure 1.32 nothing prevents us from having the three components of the application on three different machines.

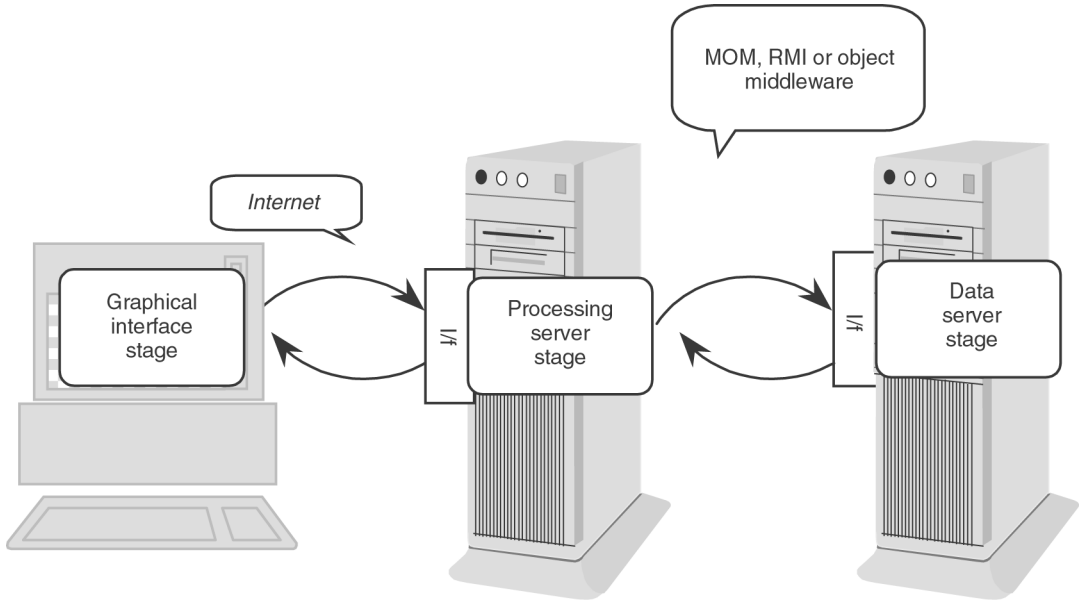

Figure 1.32 *The three-tier client–server model allows the harmonious combination of Internet and middleware technologies*

The different architectures that are possible by combining these technologies are described in detail in Chapter 9. These architectures are classified according to the model used by the Gartner Group (Figure 1.33) for describing the different business functions that can possibly be offered over the Internet, presents in order of increasing complexity. This model includes four stages which are: information publication, interaction with the site server, transaction and B2B functions.

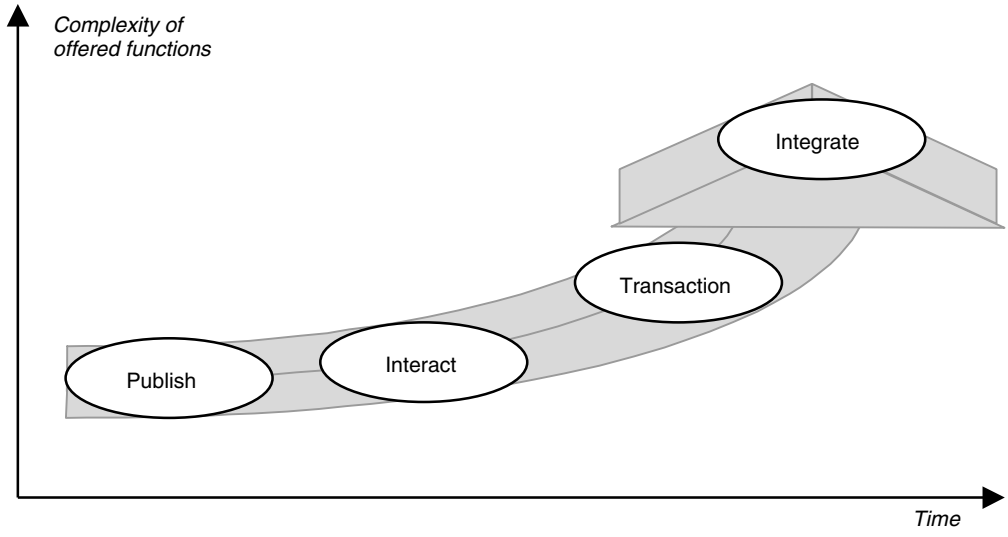

Figure 1.33 Stages in the evolution of companies in the use of Internet technologies (Gartner Group's maturity model)

These architectures should be the richness and the power of middleware technologies when they are combined over the Internet.

1.5 Object-oriented Modelling for Constructing Architectures

In today's industrial world, everyone is rushing to construct distributed information systems. The appearance of the PC at the start of the 1980s was the initiating element. The emergence of the Internet in the middle of the 1990s was software elements and marketing formed the detonator. Anyone can see the interest in distributing data, but above all applications. In any case, many of advantages arise from it, but a major disadvantage does also: complexity. So

Distributed systems are complex to design and manage.

- Complex to design: A large system (or applications) can comprise a large number of elements. It is necessary to define the functions offered by each of them while keeping their reuse in mind.
- Complex to handle: The personnel in charge of the production of information systems in a company sees with disquiet the emergence of

distributed software. Currently applications are monolithic. When they no longer function correctly, the system manager knows on which machine to look. Let us imagine the case of an application composed of ten components on a network. When this application crashes, where are they to look? How is one to know all these components and their locations?

In order to reduce this complexity, it is useful to make use of a method. The one presented here is interesting for two reasons: it allows the design of distributed systems but also, and this is its originality, it provides the information necessary for processing, during production, of a distributed information system.

The proposed method is called MethodF (MethodF is a registered name of Digital Equipment Corporation) and was defined by Digital Equipment Corporation. As with every software development method, MethodF consists of phases of specification, analysis, design and implementation. Its primary characteristics are the following (this method is described in more detail in Appendix B):

- It starts with the specification of the system to be constructed and continues to the generation of code. It also includes the system management aspect (see Figure 1.34).
- It is object oriented. The main advantage in our eyes of the concept of an object is that it forms a language that is common to domain expert for whom the application is being constructed and the computing person.
- The object model thus obtained represents the expert's system with their invariants (object). This model is completely independent of the way in which it is to be implemented.
- The specification of the system to be constructed is organized using the scenario concept introduced by Ivar Jacobson in his OOSE method. (OOSE, Object-Oriented Software Engineering.)
- UML (Unified Modelling Language) is the notation used in the analysis and design stages.
- This method is only really interesting if it is used in conjunction with a software modelling tool (The ROSE tool from Rational Corp. for whom the creators of UML work.). Such a tool allows:
  - storing model constructs during the design phase;
  - reuse entirely or in part, those models that have already been constructed. Thus, the concept of reuse is applied not to the programming level but to the design level.

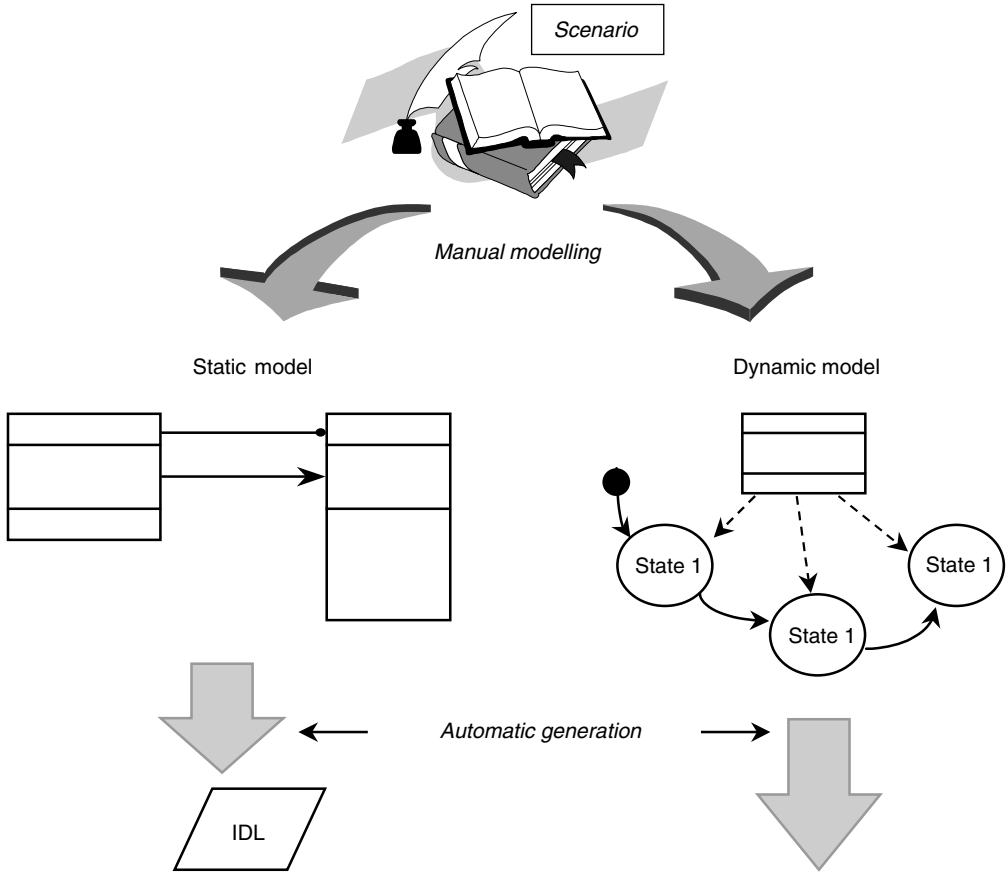

Figure 1.34 Global approach to the design, construction and management of a distributed system

  - generation, during the implementation phase, automatically, the code for objects (e.g. in C++) as well as the IDL code describing the inter-object communication protocol.
- During the design phase, the dynamic behaviour of the objects is modelled. This precious information is used later by the handler software.

1.6 Conclusion

The aim of this book is to show the importance of middleware as a technology that responds to the current needs of industry. These needs are expressed in terms of distribution and cooperation between applications or between application components.

Thus, different technologies implementing middleware are presented and analyzed. It appears that their use profoundly influences the global architecture of an information system. It implies a change in culture since it modifies the structure of applications.

The advantages that result have a price, that of complexity. The fact of using a complete methodology (Figure 1.34) integrating the problems of design, software reuse and system management in the production phase will give feelings of safety to every system manager who must, sooner or later, set out along this road. The gains are as great as the effort expended.