distributed system as local communication tends to be more efficient than communication with components that are far away.

Distributed systems are executed in concurrent processes.

As a consequence of component autonomy, distributed systems execute components concurrently. Hence, there are generally as many processes involved in the system as there are components. In addition, components are often multi-threaded; they may create a new thread whenever they start to perform a service for a user or another component. In that way the component is not blocked while it is executing a service but is available to respond to further service requests. In addition, the processes of a distributed system are generally not executed on the same processor. Hence, inter-process communication involves communication with other machines through a network. Different levels of abstraction are involved in this communication.

Distributed systems have multiple points of failure.

Unlike centralized systems, distributed systems have multiple points of failure. The system may fail because a component of the system has failed. It may also fail if the network has broken down, or if the load on a component is so high that it does not respond within a reasonable time frame. Hence, in a distributed system, situations occur in which parts of the system are fully operational while other parts, which rely on a currently unavailable component, do not function properly.

1.2 Examples of a Distributed System

In this section, we review case studies of distributed systems. The first three case studies have been developed in industry and are operational. The last example is artificial and will be used to explain concepts throughout the book.

There is an overwhelming body of distributed systems. It is probably fair to say that most of these systems were not developed using the object-oriented techniques that we discuss in this book, for these techniques are still rather new. When we selected the case studies, we deliberately chose case studies that have been built using object-oriented techniques in order to illustrate that it is feasible to deploy object technology for distributed system construction.

1.2.1 Video-on-Demand

A video-on-demand service subscriber can rent videos without having to obtain videotapes. The videos are, rather, transmitted electronically. From a customer's point of view, there are many advantages. The video-on-demand service can be operational 24 hours a day; movies are always available as there is no limitation on physical media; and the time between deciding to watch a movie and the availability of the video is reduced drastically.

Example 1.2
Hong Kong Telecom Interactive Multimedia Services

Hong Kong Telecom Interactive Multimedia Services have built a video-on-demand service using distributed objects. In this system, the client components that are used to display videos and the server components that load and transmit videos are considered as objects. They are distributed: clients are hosted by set-top boxes in the homes of the customers of the service and several video-on-demand servers were needed in order to make the system scale to its current 90,000 subscribers.

Video-on-demand system uses heterogeneous hosts.

The client components in the case study were implemented in Java, which means that they can be downloaded from a web server. Updating the copy on the web server implements migration to new versions of the client. Moreover, Java was chosen in order to ensure that the clients run on a number of different platforms, including web browsers and set-top boxes. The server objects were implemented in C++, mainly because they had to perform efficiently on Unix servers. Hence, the video-on-demand system is a distributed system consisting of heterogeneous components.

Video-on-demand system is built using middleware.

The video-on-demand system uses an object-oriented middleware. We will revisit this form of middleware throughout the book. It supports the definition of interfaces of the server objects in a way that is independent of programming language, network and hardware. It manages the communication between client and server objects that is necessary to locate the server that holds a desired video and to select the video to be downloaded.

Video-on-demand system has all the characteristics of a distributed system.

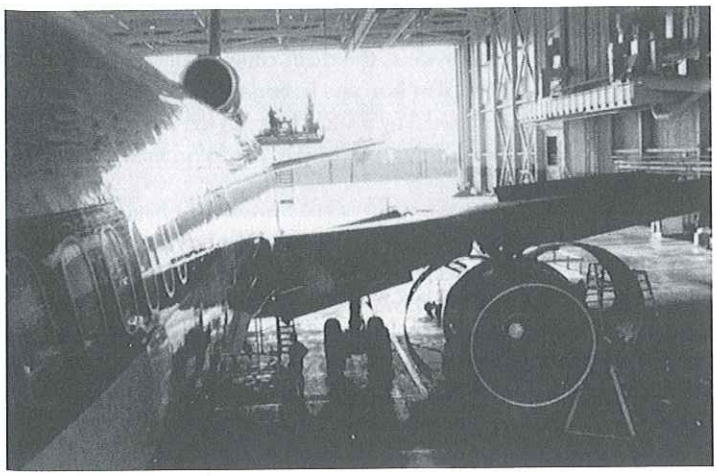
The system exhibits all of the characteristics of distributed systems that we discussed in the previous section. The components are autonomous and heterogeneous: client components are written in Java. They operate in web browsers or set-top boxes. The servers operate on Unix hosts. The servers are shared components while each customer has an unshared client object. The system is inherently concurrent, given that there are 90,000 potential users. Although it is unlikely that all subscribers will decide to watch a video at the same time, there are probably several hundred concurrent users at any point in time. Finally, there are also multiple points of failure as each of the set-top boxes/web browsers, the network link or the servers might be overloaded or fail.

1.2.2 Aircraft Configuration Management

Support systems for aircraft configuration management must be scalable.

The scale of aircraft design and manufacturing processes at Boeing demands extensive IT systems. Aircraft return to the manufacturer regularly for maintenance and safety checks. Maintenance engineers therefore require an up-to-date database of the engineering documents for that particular aircraft, covering every part and the models and individual designs in which parts are used. As Boeing constructs approximately 500 new aircraft each year, the database of engineering information has documents about 1.5 billion parts added to it each year. The centralized and mainframe-based system with which the aircraft manufacturer was working was not capable of bearing this increasing load any more and a new system had to be constructed.

Example 1.3
Construction of an Aircraft



The above photograph gives an impression of the construction floor where aircraft are built and maintained. A modern aircraft consists of approximately three million individual parts. The process of integrating and maintaining these parts as a cohesive and functional whole is long and complicated for two reasons. First, every aircraft is unique because different airlines have different needs. Operators of scheduled flights, for instance, order aircraft with fewer seats and bigger overhead lockers than charter airlines. Thus, each combination of parts is put together according to specific customer demands. Secondly, in the process of revising a basic design, additional revisions may become necessary in other areas of the design. Substituting one part for another in a specific area may then require the design team to ensure the switch has no detrimental effects in other areas. If changes are required, such further revisions must be looped back through the system until a new stable design is created.

Integration of components from different vendors leads to heterogeneity.

Rather than building components from scratch, Boeing decided to use commercial off-the-shelf components, such as product data management, enterprise resource planning and computer-aided project planning systems. The decision to deploy unmodified third-party software allowed the vendor to use the best available systems. However, in order to build a system that is perceived by its users as a single integrated computing facility, Boeing had to integrate heterogeneous components into a distributed system.

Object-wrapping can be used to integrate off-the-shelf components.

Boeing made a decision early on to integrate its chosen software packages as a whole. They adopted a technique called *object wrapping*. Packages are wrapped with an interface encapsulating large chunks of application functionality, rather than breaking the applications down into smaller business objects and attempting to encapsulate each of these on an individual basis. This approach allowed the project to make rapid initial steps by avoiding extensive application-level revamping and allowed the producer to take full advantage of its decision to buy instead of build. The deployment of a distributed system for the aircraft configuration management system enabled the manufacturer to build a scalable system quickly (because it was composed from existing components). The system will be able to respond to changing needs more easily as new components can be added. Moreover, the system improved the

overall performance of aircraft design and maintenance as it is better integrated. Hence, the manufacturer has gained a competitive advantage because it can design, customize and maintain aircraft more swiftly.

The integration was achieved by using an object-oriented middleware to integrate a data centre which will, in the end, have more than 45,000 total users, with as many as 9,000 at any one time. 20 large Sequent database machines run Oracle relational databases to store engineering data. 200 Unix servers host components for the application software and numerous workstations are used to provide construction and maintenance engineers with graphical user interfaces to the system.

Middleware is used to resolve heterogeneity and distribution.

Again, the aircraft configuration management system exhibits all the characteristics of a distributed system. The degree of autonomy in this case study is even higher than in the previous case study as Boeing did not have access to the source code of the components they bought off the shelf. The system is also more heterogeneous with components running on a larger variety of hardware platforms. The system consists of components that operate in concurrent processes. Some of these components are shared (e.g. the components providing the database access) and some components are not shared (the browsers with which aircraft configurations are displayed on a workstation). Again, the system is a concurrent system as the 20 Sequent machines run database servers that are concurrently accessed by the system's users. The system also has multiple points of failure. Any of the databases may be overloaded; the network may fail; or servers operating the aircraft configuration management applications may be unavailable.

Configuration management system has all characteristics of a distributed system.

1.2.3 IT Infrastructure of a Bank

The IT infrastructure in banks are still dominated by mainframe systems. The Swiss bank from which this case study was taken is a very typical example. It needed to integrate application components on these mainframe systems with application components that provide services on more modern platforms.

An example of a service is obtaining the mortgage balance of a customer or executing an equity transaction. Services are provided by reusable components. A component encapsulates a particular application. This application might be a new application or it might be a legacy application that the bank has used for a long time. Components are integrated with other components for the implementation of business processes. The integration of components that reside on heterogeneous platforms, however, means that distributed systems are constructed.

Service architecture consists of heterogeneous new and legacy components.

The bank has used transaction-oriented middleware, which could integrate distributed transactions across multiple hosts. It then decided to migrate towards using an object-oriented middleware in order to implement the service architecture in a systematic way. The components were implemented as distributed and heterogeneous objects. Objects export operations, which are the implementations of services. Objects might request the execution of services from other objects. Objects might combine the results of service

Different types of middleware can be used to resolve distribution and heterogeneity.