Chapter 2: Enterprise Applications from a Middleware Perspective

In this chapter, we give an introduction to enterprise applications from a middleware perspective. Some aspects have already been outlined in Section 1.1 – here we give a more detailed overview. The chapter is structured as follows: Section 2.1 gives a definition of enterprise applications and outlines some of their main properties. Section 2.2 describes transactions in enterprise applications. Then, in Section 2.3, we explain the commonly used concept of structuring enterprise applications into multiple tiers. Section 2.4 discusses the underlying middleware of enterprise applications. Finally, Section 2.5 gives a brief summary of this chapter.

2.1 Enterprise Applications

Although the term *enterprise application* is frequently used, there is no commonly accepted definition of enterprise applications. As a basis for this thesis, we provide the following definition:

> *Enterprise applications are transactional, distributed multi-user applications that are employed by organizations to control, support, and execute business processes.*

The main task of an enterprise application is to process business data that is relevant to business processes of an organization. For example, an enterprise application in a travel agency may process customer, hotel, flight, and reservation data. Low-level, data-related operations include deleting, updating, and querying existing business data items and inserting new business data items. By combining these primitive operations, more complex functions can be realized. For many organizations, their digitally stored business data is an important asset and it is essential to ensure the integrity of that data even under exceptionally circumstances. Therefore, enterprise applications typically access business data within transactions. This is reflected in our definition of enterprise applications given above[1].

Typically, business data items are persistently stored in *transactional data stores*, where they are accessed by the enterprise application. Today, mainly relational database management systems (RDMBS) [EN00] are used as transactional data stores. Examples of widely used RDBMS products are the Oracle database management system [LK02], IBM's DB2 [Cham98], and MySQL [DuBo03]. However, it should be noted that there are other types of systems (including non-database systems) that may serve as transactional data stores – for instance, object database management systems [Catt94], hierarchical database management systems [TL76], file systems with transactional properties [Hagm87], or transactional message queues [MD94].

In contrast to single-user or stand-alone applications, enterprise applications typically allow concurrent access by multiple users (possibly to the same data items). This feature is important because business processes run in parallel, may involve several users, and different activities may require access to the same data items.

Enterprise applications are distributed systems, i.e., consist of pieces of software that run on different machines connected by a communication network. There are many reasons for distribution: For instance, multi-user access usually requires a client/server architecture. Also, an enterprise application may have to process data items stored in different transactional data stores. There are many more possible reasons for

[1] We consider transactional protection as a property of fundamental importance. There are enterprise applications that choose to use a lower level of protection or no transactions at all. However, we regard non-transactional access as exception rather than the rule.

distribution, e.g., better performance, higher fault tolerance, or legal requirements. In Chapter 3, we discuss the distributed structure of enterprise applications in more detail.

2.2 Transactions

To protect organizations against data loss and data inconsistencies in case of failures and concurrent access, business data items are transactionally accessed by enterprise applications. A *transaction* [EGLT76] [GLPT76] combines a group of operations on data items into a single unit of work. Transactions are always executed in compliance with the ACID properties (Atomicity, Consistency, Isolation, Durability) [HR83]:

- *Atomicity.* A transaction is executed as an indivisible unit. Either all operations of a transaction are executed and their effects become visible (the transaction commits) or the transaction has no effect on data (the transaction is rolled back).
- *Consistency.* A transaction always transforms one consistent state of data into another consistent state of data.
- *Isolation.* Concurrent transactions are isolated from each other. A transaction can only commit if it is executed as if there were no other concurrently executing transactions.
- *Durability.* The effects of a committed transaction are permanent. They are persistently stored so that they survive failures.

To a large degree, transaction support in enterprise applications is implemented as part of transactional data stores. However, in many situations, data store transactions are complemented by transaction mechanisms of enterprise application middleware, which is discussed in Section 2.4. This is especially the case for transactions that access data items in more than one transactional data store, i.e., are *distributed transactions*. To guarantee global consistency across multiple data stores, distributed commit protocols such as the two-phase commit protocol [Gray78] are employed. A standard for distributed transaction processing (including the execution of the two-phase commit protocol) is the X/Open Distributed Transaction Processing (DTP) Model [XO91] [XO95], which defines an architecture for distributed transaction processing. The standard ensures interoperability across heterogeneous transactional middleware and database products. Most major transactional data stores implement the standard's XA interface and thus are able to participate in distributed transactions. Often, distributed transactions that follow the X/Open DTP model are simply referred to as XA transactions.

2.3 Distributed Multi-Tier Structure

In practice, many enterprise applications are developed with a special emphasis on *multi-tier* structures. That approach is well-tried, helps to reduce complexity, and improves modularity.

A tier is a special kind of *layer*. Layered architectures are, for example, discussed in [BMR+96] and [SG96]. In contrast to the more general concept of layers, tiers correspond to the distributed structure of an application (see [Hart01a]). Usually, a tier corresponds to one or more heavyweight operating-system–level *processes*. A process [SGG02] is a fundamental concept of most modern operating systems. A *heavyweight* process is a program in execution; among other things, code, data, and resources are associated with it. A heavyweight process may in turn consist of several *lightweight* processes (sometimes called *threads*). On a single machine, multiple processes can run concurrently, and each one has a separate flow of control. The operating system is responsible for assigning local processors to processes by means of a scheduler. Each heavyweight process runs in a separate address space that is protected from other heavyweight processes. Lightweight processes within a heavyweight process have separate control flows but share data and resources. Note that, although the same term is used, operating-system–level processes do not correspond to business processes of an organization.

Heavyweight operating-system–level processes of the same tier or of different tiers can be – and often are – deployed on different machines. For communication between processes of different tiers, a remote communication mechanism is employed like, for example, remote procedure calls or a messaging service. In principle, arbitrary criteria can be used to group processes into tiers. In practice, tiered structures are usually defined such that each tier implements a specific subset of the application's functionality, e.g., presentation or data access. Figure 2-1 shows an example of such a multi-tier structure that consists of a client tier, a business logic tier, a data access tier, and a database tier.

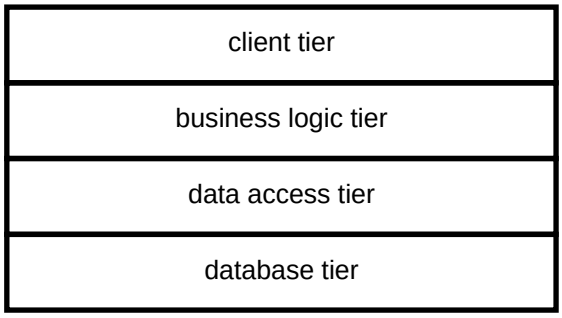

Figure 2-1: An example of a (totally ordered) multi-tier structure.

In simple cases, the tiers in a multi-tier structure are totally ordered and each tier communicates with neighboring tiers only. However, there are more complex multi-tier applications that cannot be described with such a restricted structure, e.g., because different subsystems have their own tiered structures. Such applications are based on *partially* ordered tiered structures. An example of an enterprise application with a partially ordered multi-tier structure is illustrated in Figure 2-2.

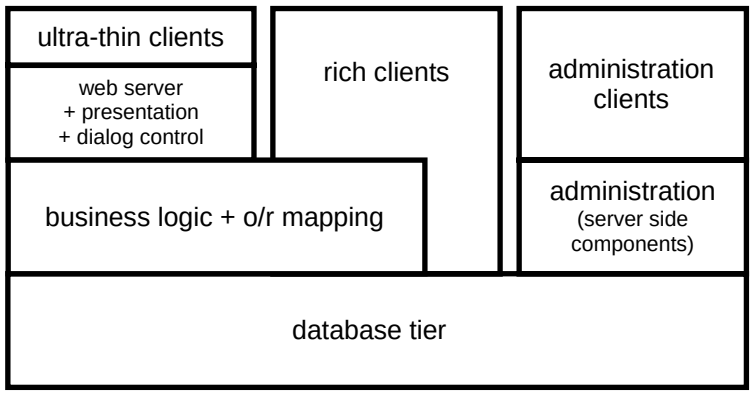

Figure 2-2. An example of an enterprise application with a partially ordered multi-tier structure.

The example shows the multi-tier structure of a complex enterprise application: The application stores business data in relational databases, which are associated with the database tier. Three kinds of clients are supported: Web browsers as ultra-thin clients, rich clients, and administrative clients. Ultra-thin and rich clients are utilized by regular users of the application and rely on business logic, which is implemented on the server side. Between ultra-thin clients and the business logic an additional tier is placed: The tier is responsible for dialog control and generating dynamic HTML documents for the presentation. Rich clients mostly communicate with the tier that implements the business logic. But for a few selected, performance critical statistical functions, rich clients directly access databases in the database tier. Administration clients talk to server side administration components, which have low-level

access to the databases. Low-level access bypasses all business logic and also the object/relational mapping required by the implementation of the business logic.

We can clearly see that the given enterprise application does not have a totally ordered multi-tier structure. Instead, we can identify subsystems (for example, administration) that have their own multi-tier structures that differ from those of other subsystems. Chapter 3 provides more details on partially ordered multi-tier structures.

The strong emphasis on multi-tier structures is a property that distinguishes enterprise applications from many (but not all) other types of distributed applications. In the last years, particularly peer-to-peer systems, like Freenet [CSWH01] or Gnutella [Ripe01], have gained much attention. This is interesting because, in many respects, some of which are presented below, peer-to-peer systems are the exact opposite of enterprise applications:

- While enterprise applications employ different types of processes that implement only specific subsets of the enterprise application's overall functionality (e.g., database access), processes in peer-to-peer systems typically implement equivalent functionality.
- In peer-to-peer systems, in principle, each process can communicate with every other process (peer). In contrast to that, enterprise applications strictly limit the allowed types of connections, define a specific, often centralized distributed structure, and typically enforce strict client/server roles on pairs of communicating processes.
- In peer-to-peer systems, processes often join and leave the system spontaneously. However, in an enterprise application, only pure clients are allowed to do so. Server processes, typically, run permanently because their clients depend on their availability.
- In most enterprise applications, a high quality of service is essential. The application, its usage, and the underlying hardware resources are often carefully managed by a central authority to ensure an appropriate quality of service for users. In many peer-to-peer systems, there is no such authority (or only for parts of the system) and services are provided merely on a best effort basis.
- Enterprise applications are usually designed and/or deployed with a certain number of users or a specific amount of load in mind. A significant part of the underlying hardware and software resources (e.g., database servers) are allocated by, maintained by, and under the control of a specific organization that runs the enterprise application. In contrast to that, for peer-to-peer systems, it is a common design goal to scale from a few users to large, Internet scale systems. Hardware resources become available as more peers join the system.
- While enterprise applications allow both read and write access to shared data, peer-to-peer systems typically focus on read-only access.
- Many enterprise applications play a mission critical role in their organizations. Therefore, for instance, transactional access to data is provided to ensure consistency or data is backed up in frequent intervals. Current peer-to-peer systems do not perform mission-critical tasks and thus do not need such features.

2.4 Enterprise Application Middleware

Enterprise applications are rarely implemented from scratch, i.e., only on top of low-level services of the underlying operating systems. Instead, they are typically built on top of services that are commonly referred to as *middleware* [Bern96] [Emme00]. Middleware in general is used in many application areas. However, in this thesis, we focus on *enterprise application middleware*, i.e., middleware typically used by enterprise applications.

Middleware can be considered an intermediate software layer between applications and the operating system. Particularly for distributed communication, coordination, and data management, enterprise applications typically rely on functions of the underlying middleware. This is advantageous because application developers can focus on the application-specific details of their enterprise applications (such as business logic, information model, and user interface). Common infrastructure functionality required by many applications is implemented as part of third-party middleware products. There are many different middleware architectures, middleware services, and products that implement these services. In the following, we outline selected middleware that is relevant to distributed data management in enterprise applications.

Numerous middleware classifications have been proposed. With respect to the style of inter-process communication, we distinguish three types of middleware approaches (similarly to [Ritt98]): RPC-based, message-oriented, and object middleware.

- *Remote procedure call (RPC) based middleware.* With conventional procedure calls, subroutines may be invoked only in the caller's address space. In contrast to that, an RPC mechanism allows processes to invoke subroutines in other processes (local or remote). A client invokes a remote procedure in the same way as it would invoke a local procedure. RPC middleware is responsible for marshalling the call and its parameter to a wire representation, sending it over the network, unmarshalling it at the remote site, and invoking the procedure at the remote site. Return values, acknowledgements, or errors are transmitted back to the client using the same approach. RPC middleware usually relies on generated client-side stub and server-side skeleton code. Typical RPC-based communication is synchronous, i.e., an RPC client is blocked until the remote procedure has been executed or an error occurs. Examples of RPC middleware are the Open Software Foundation's DCE RPC protocol [SHM94] and XML-RPC [User99]. Also, web services [Cera02] based on the SOAP protocol [BEK+00] can be seen as a kind of RPC middleware.
- *Message-oriented middleware (MOM).* With MOM, processes can asynchronously exchange messages. In contrast to RPCs, typical MOM-based communication is neither type-safe nor does it hide heterogeneity because the format of a message body is up to the client and the server code. In this respect, MOM-based communication is relatively low-level – however, at the same time, many MOM products support powerful features like multicast, persistency, and transactional processing of messages. Many MOM products use transactional message queues, which support advanced delivery guarantees and allow applications to transactionally insert and remove messages [MD94]. Examples are IBM's WebSphere MQ [IBM03d] (formerly MQSeries) or TIBCO Rendezvous [TIBC02]. In Java environments, Sun's Java Message Service Specification [Sun02] is the de-facto standard for MOM products.
- *Object middleware.* Object middleware extends the notion of RPCs by adding object-oriented concepts. While RPCs are invoked on *procedures*, object middleware allows to reference individual *object instances* in remote processes and to perform method invocations on them[1]. Particularly for application developers, this approach is convenient as middleware remote objects can directly be mapped to programming language objects in object-oriented applications. As is the case with RPCs-based communication, remote object invocations are typically synchronous and usually require generated stub and skeleton code. Examples of object middleware are the Object Management Group's Common Request Broker Architecture (CORBA) [OMG02] and Sun's Remote Method Invocation (RMI) [Gros01].

Based on the three approaches described above, a broad range of enterprise application middleware services have been developed for naming, caching, replication, security, distributed transactions, clustering, persistency, and many more. While some services are proprietary, other services comply with

middleware standards, e.g., with CORBA, which specifies several services, Sun's Java 2 Enterprise Edition [Sun03a], or Microsoft's .NET [Rich02].

In some cases, implementations of individual enterprise application middleware services are available as standalone products, e.g., services for distributed caching or object/relational mapping frameworks. However, multiple well-integrated services are often bundled into larger products that provide complete platforms for enterprise applications. Traditionally, the term *transaction processing monitor* (TP monitor) had been used for such products. An overview of TP monitor concepts and TP monitor products like IMS, CICS, and TUXEDO can be found in [BN97] and [GR93]. Before the object-oriented paradigm has been widely accepted in software engineering, enterprise applications were typically built on top of such centralized TP monitors.

Nowadays, TP monitors are replaced by a new generation of middleware products for realizing object-oriented multi-tier architectures. Within these architectures, entities of the application's business domain, e.g., customers and accounts, are typically represented as *business data objects*, i.e., mapped to object instances of an object-oriented programming language. Business data objects are transactionally accessed by the enterprise application, and their persistent state is stored in one or more transactional data stores. The new generation middleware products are commonly referred to as *application servers*. Recent application server products are typically based on Sun's Java 2 Enterprise Edition and/or CORBA. Examples of widely used application server products are IBM's WebSphere Application Server [IBM03c], BEA System's WebLogic Server [Bea03], Iona's ASP product line [IONA03], and the JBoss application server [JBos03].

Middleware plays an important role for enterprise applications for the following reasons:

- The use of middleware simplifies application development. Ideally, all infrastructure functionality is implemented as part of the underlying middleware, which is accessed by the enterprise application through standardized application programming interfaces. The use of middleware helps application developers to focus on those software parts that contain the business logic of their enterprise applications. We will refer to these software parts as *application code*.

  Often, middleware is designed to hide (make transparent) selected infrastructure issues from application code in order to decouple application code effectively from infrastructure issues. The Reference Model for Open Distributed Processing (RM-ODP) [Putm00] defines several of such transparencies. For instance, *location transparency* allows application code to access distributed components independently of their location in space. *Replication transparency* lets application code access replicated functions/components in the same way as without replication. In addition, RM-ODP defines access, failure, migration, relocation, persistence, and transaction transparency.

- In addition, many fundamental non-functional properties of an enterprise application (e.g., performance, scalability, adaptability, interoperability, security) are often significantly influenced by the underlying middleware. In this thesis, especially adaptability (with respect to the distributed structure of an enterprise application), performance, and scalability are relevant.

Note that, except for persistence-related aspects, there is a substantial overlap in functionality between database management systems and middleware. The current trend in enterprise application development is that more and more traditional database functionality is realized as part of middleware running in client and intermediate processes. This trend has already been described in [CD96] and, since then, has continued for years. Evidence for that development is the large number and popularity of object-oriented frameworks, object/relational mapping products, and application servers on the market. Nevertheless, database management systems remain an essential part of enterprise applications.

[1] Note that, if desired, object middleware may also be used in a style similar to RPC-based middleware. This can be achieved by representing each process as a single, coarse-grained remote object.

2.5 Summary

In this chapter, we discussed enterprise applications from a middleware perspective. We provided a definition of enterprise applications and outlined their main characteristics. Then we introduced transactions (especially distributed transactions), which are employed to protect against data loss and data inconsistencies. We explained that enterprise applications are often structured into different tiers and there are totally and partially ordered multi-tier structures. Finally, we discussed enterprise application middleware, which is a basis for many enterprise applications. Middleware services implement common infrastructure services for enterprise applications and thus allow application developers to focus on application-specific parts of their enterprise applications.