Chapter 1: Introduction

1.1 Background

In this thesis, we examine a family of applications that are frequently referred to as *enterprise applications*. Enterprise applications play a crucial role in many organizations – to mention only systems for accounting, online shopping, securities trading, and flight reservations as examples.

There is no precise, commonly accepted definition of enterprise applications. However, they are often associated with the following characteristics:

- Enterprise applications are employed by organizations to control, support, and execute business processes. For that purpose, they process business data relevant to their organizations, for instance, customer entries and order data. Typical data-related operations include deleting, updating, and querying existing business data items as well as inserting new business data items.
- Enterprise applications support concurrent access by multiple users.
- Enterprise applications are distributed, i.e., consist of pieces of software that run on different machines connected by a communication network.
- Enterprise applications provide transactional access to business data: A group of operations on business data can be combined into a single unit of work called a *transaction*. Transactions are executed in compliance with the ACID properties (atomicity, consistency, isolation, durability). Transaction semantics are especially important for mission-critical enterprise applications because they protect applications against data loss and inconsistencies caused by failures and concurrent access.

Traditionally, data-intensive enterprise applications have been built on top of centralized transaction processing (TP) monitors. Nowadays, these TP monitors are replaced by object-oriented multi-tier architectures, for instance, architectures based on Sun's Enterprise JavaBeans and/or the Object Management Group's Common Object Request Broker Architecture (CORBA). Within these architectures, entities of an application's business domain, e.g., customers and orders, are no longer represented as records or as rows in a table. Instead, they are typically mapped to object instances of an object-oriented programming language. We will refer to such objects as *business data objects* (or short: *data objects*). Business data objects are transactionally accessed by the enterprise application, and their persistent state is stored in one or more transactional data stores, for instance, relational database management systems.

Few enterprise applications are developed from scratch and rely only on functions of the operating system. Instead, most enterprise applications rely on underlying services that are commonly referred to as *middleware*. Middleware services can be regarded as an intermediate layer between applications and operating systems and typically handle distribution, heterogeneity, and transaction management for enterprise applications. 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 allows application developers to focus on the business logic of their enterprise application.

1.2 Focus

The design of an enterprise application comprises many aspects, such as distribution, security, class structure, module structure, user interface, data model, and many more. In our experience, the distribution aspect frequently dominates most of the other aspects when developing enterprise applications (in the sense that distribution influences, structures, and sometimes even determines other design decisions). Therefore, we believe that the distribution aspect deserves special attention.

In data-intensive enterprise applications, large numbers of business data objects are managed and accessed. Thus, efficient data management mechanisms have to be employed to access, load, store, synchronize, copy, etc. data objects in a scalable manner.

In this thesis, we take a middleware-centric view of data-intensive, object-oriented, multi-tiered enterprise applications. In particular, we focus on two aspects: distribution and data management.

1.3 Motivation and Problem Statement

In many cases, the distributed structure of an enterprise application directly influences important non-functional properties such as performance, scalability, availability, and security. For some enterprise applications, simple standard distributed structures such as the plain three-tier architecture (informally) shown in Figure 1 (a) might be sufficient. However, many more demanding enterprise applications require more sophisticated distributed structures to individually address specific contexts and requirements. An example of such a *custom* distributed structure is sketched in Figure 1 (b).

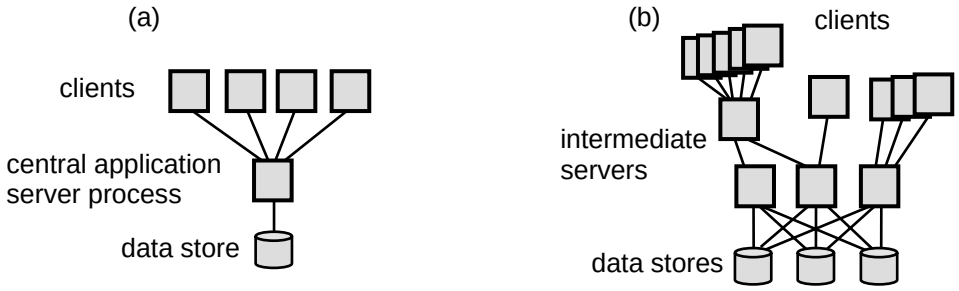

Figure 1: A simple three-tier structure (a) and an example of a more complex, application-specific distributed structure (b).

Usually, the design of an adequate distributed structure for a concrete application is a task of a software architect. Typically, a design has to take into account a broad range of factors, such as performance, costs, the need to integrate existing subsystems, or organizational and legal requirements. Finding a reasonable trade-off is highly application-specific and can be a challenging task. In addition, when relevant factors change during the life cycle of an enterprise application (e.g., more scalability or fault tolerance is required), it may be necessary to *adapt* its existing distributed structure.

Unfortunately, software architects and application developers experience two severe problems in practice:

- *Problem 1: With existing middleware, it is difficult to build enterprise applications with custom distributed structures.*

  Typically, only a small set of standard structures is well supported by current middleware. More complex, application-specific distributed structures are either not supported or lead to significant performance problems or restrict full middleware services to a subset of distributed components only. To create efficient, custom distributed structures, application developers often have to im-

plement typical middleware functionality (e.g., object-oriented access to data objects, transaction management, and caching) as part of their application code. This is a severe problem because implementing middleware functionality is a complex, costly, and time-consuming task that diverts application developers from the business logic.

- *Problem 2: With existing middleware, it is difficult to adapt the distributed structure of an existing enterprise application to meet new requirements.*

  Traditionally, the distribution of enterprise applications is considered a high-level, architectural concern. Decisions on distribution have to be made at an early design stage of a development project and are hard to change later on. Changing an existing distributed structure often requires major re-design and/or re-implementation of large parts of the enterprise application. This is not only costly – in the worst case, this prevents organizations from quickly responding to changes.

1.4 Objectives

In this thesis, we address the two problems outlined in the previous section. More specifically, the thesis has the following objectives:

- The first objective is to analyze why existing middleware (a) provides only very limited support for enterprise applications with custom distributed structures and (b) makes it hard to adapt the distributed structure of an existing enterprise application later on.

  The description of problems in the previous section is merely based on observations and does not explain causes. A thorough analysis will help software architects and application developers to understand what kind of problems they are facing and, in particular, why (and in which context) these problems occur.

- The second objective is to develop concepts for an enterprise application middleware that (a) explicitly supports custom distributed structures and (b) also allows to easily adapt them to new requirements during the life cycle of an enterprise application.

  The concepts can serve as a basis for developing new enterprise application middleware as well as for extending existing middleware products and standards.

The first objective is an intermediate step on the way to achieving the second one as our main objective.

1.5 Approach

Up to this point, we have used the term *distributed structure* in a rather informal way. However, a middleware-centric approach requires a much more detailed notion of the distributed architecture of enterprise applications. To lay a solid theoretical foundation for our thesis, we first define an architectural style [SG96] for multi-tiered enterprise applications. An architectural style describes the architecture of a family of systems in terms of components, connectors, and constraints. Our style focuses on an application's tiered structure and its *process topology*, which consists of distributed processes (as defined by the operating system), data stores, and client/server communication relationships.

The design of a suitable process topology for a concrete application is a task of a software architect. Typically, a design has to address a broad range of conflicting requirements. A particularly successful approach to complex software design problems is the use of software patterns [GHJV95] [BMR+96]. To guide the design of process topologies, we present a catalogue of software patterns – so called *topology patterns* – that form a pattern language for our architectural style.

Based on the architectural style and the pattern language, we argue that, for many enterprise applications, simple standard process topologies are not sufficient. We explain that, instead, many demanding enter-

prise applications require custom and adaptable process topologies. *Custom* topologies are needed to address application-specific requirements. *Adaptable* means that existing process topologies are easy to adapt to changing requirements without having to re-design the enterprise application. To illustrate the need for custom and adaptable process topologies and to give an example, we present a detailed case study. The case study describes a tiny system that develops into a large, complex, multi-tiered enterprise application. During its life cycle, its process topology is constantly subject to change to meet new requirements.

Unfortunately, it is difficult to realize enterprise applications with custom and adaptable process topologies on top of existing middleware. In principle, we could discuss these problems for each relevant combination of middleware and each possible usage of middleware – that is, however, not realistic. Instead, we decided to take a different approach: First, we identify and discuss six key requirements for middleware to efficiently support custom and adaptable process topologies. Then we show that existing enterprise application middleware does not fulfill all of these requirements. By discussing the requirements and pointing out limitations of current middleware, we address our first objective (see Section 1.4).

To address our second objective, we present our *Flexible Process Topology* architecture (FPT architecture), which realizes all of our requirements. The architecture defines principles of a middleware framework that explicitly supports custom and adaptable distributed process topologies. The architecture relies on a network of object manager components that collectively provide data management services to enterprise applications. One of the main advantages of the FPT architecture is that the underlying distributed structure of an enterprise application can be defined and adapted through (re)configuration without having to re-design the application. This gives developers more flexibility in constructing and customers more flexibility in deploying and adapting their enterprise applications.

As a proof-of-concept, we present the implementation of a middleware framework that is based on the concepts of our FPT architecture. The framework scales from simple two-tier architectures to large-scale distributed structures with features like an arbitrary number of tiers, distributed data stores, and replication of application processes. We describe its design and outline selected implementation details.

To evaluate the FPT architecture and our implementation, we implement an example enterprise application (a marketplace for accommodations) on top of our middleware framework. We demonstrate that, in typical cases, the construction of custom process topologies and their adaptation is mainly a matter of configuration. Furthermore, we analyze performance aspects for several typical scenarios. We show that our framework introduces little overhead and provides good performance.

1.6 Contributions

The three main contributions of this thesis are:

(1) the definition of an architectural style for multi-tiered enterprise applications, along with a pattern language for designing process topologies,

(2) the identification of requirements for enterprise application middleware to support custom and adaptable process topologies along with an analysis why current middleware does not fulfill these requirements, and

(3) the FPT architecture, which comprises concepts for enterprise application middleware to support custom and adaptable process topologies. In addition to the architecture itself, a proof-of-concept implementation and a thorough evaluation are provided.

1.7 Thesis Outline

This thesis is structured as follows:

Chapter 1 (the current chapter) provides a brief overview of the background, objectives, approach, and structure of the thesis. In Chapter 2, we give an introduction to enterprise applications from a middleware point of view. Chapter 3 defines an architectural style for multi-tiered enterprise applications and a pattern language for designing process topologies. Chapter 4 motivates the need for custom and adaptable process topologies, identifies key requirements for such topologies, and explains why existing middleware does not fulfill these requirements. In Chapter 5, we present our FPT architecture for custom and adaptable process topologies. Chapter 6 describes our proof-of-concept implementation, which realizes all concepts of the FPT architecture. Chapter 7 is devoted to an evaluation of the FPT architecture and our implementation: We demonstrate an example enterprise application with an adaptable custom topology, analyze performance aspects for typical scenarios, and compare our solution to an application based on object middleware. Chapter 8 gives a summary of our thesis and outlines future work.