

A Middleware Architecture for Transactional, Object-Oriented Applications

Dissertation am Fachbereich Mathematik und
Informatik der Freien Universität Berlin

eingereicht von Christoph Hartwich
am 14. November 2003, verteidigt am 17. Dezember 2003

Betreuer:

Prof. Dr. Heinz Schweppe

Dr. Ralf-Detlef Kutsche

Gutachter:

Prof. Dr. Heinz Schweppe

Prof. Dr. Herbert Weber

Prof. Johann-Christoph Freytag, Ph.D.

Abstract

Enterprise applications are transactional, distributed multi-user applications that are employed by organizations to control, support, and execute business processes. Traditionally, data-intensive enterprise applications were built on top of centralized transaction processing monitors. Nowadays, these TP monitors are replaced by object-oriented multi-tier architectures. In this thesis, we focus on two aspects of object-oriented, data-intensive enterprise applications: distribution and data management.

We first define an architectural style for multi-tiered enterprise applications. In addition to the tiered structure, the style focuses on an application's *process topology*, which consists of distributed processes as defined by the operating system, data stores, and client/server communication relationships. Usually, the design of an adequate process topology for a concrete application is the task of a software architect. Typically, a design has to address a broad range of requirements, such as scalability, performance, availability, security, the need to integrate existing subsystems, or organizational and legal requirements. As a guide to the design of process topologies, we present a catalogue of patterns (so called *topology patterns*) that form a pattern language for our architectural style.

Traditionally, 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 expensive to change later on. Moreover, existing enterprise application middleware supports only a small set of standard topologies well. However, we argue that demanding enterprise applications require *custom* and *adaptable* process topologies:

- *Custom* topologies are needed to address application-specific requirements, e.g., those concerning replication, distributed data, caching, or integration of existing subsystems. Using simple standard topologies instead of custom topologies can significantly restrict the capabilities of demanding enterprise applications.
- During the life cycle of an enterprise application, requirements are likely to change. For example, an application has to serve an increasing number of concurrent users or has to provide more fault tolerance. A process topology should be *adaptable*, i.e., easy to adapt to changing requirements without having to re-design the application.

Unfortunately, such topologies are difficult to realize with existing enterprise application middleware. We analyze the difficulties and identify six key requirements for middleware to efficiently support custom and adaptable process topologies. Among other things, it is essential to pairwise decouple topology, application code, and data distribution scheme. Furthermore, we present our *Flexible Process Topology* (FPT) architecture, which realizes all these requirements. The architecture defines principles of a middleware framework that enables custom and adaptable distributed process topologies. The architecture relies on a network of object manager components that collectively provide data management services to application code.

As a proof-of-concept, we present an exemplary implementation of a middleware framework for enterprise applications that is based on the concepts of our FPT architecture. The prototype demonstrates that, in typical cases, the distributed structure of an enterprise application can be defined and adapted through *(re)configuration* – without affecting the implementation. This gives developers more flexibility in constructing and customers more flexibility in deploying their enterprise applications. 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.

Finally, we evaluate adaptability and performance of our proof-of-concept implementation for several typical scenarios.

The main contributions of this thesis are:

- the definition of an architectural style for multi-tiered enterprise applications, along with a pattern language for designing process topologies,
- the identification of requirements for enterprise application middleware to support custom and adaptable process topologies, and
- a middleware architecture (including an implementation and an evaluation) that explicitly supports custom and adaptable process topologies.

Contents

Chapter 1:	Introduction.....	1
1.1	Background	1
1.2	Focus	2
1.3	Motivation and Problem Statement	2
1.4	Objectives.....	3
1.5	Approach	3
1.6	Contributions.....	4
1.7	Thesis Outline	5
Chapter 2:	Enterprise Applications from a Middleware Perspective.....	7
2.1	Enterprise Applications	7
2.2	Transactions	8
2.3	Distributed Multi-Tier Structure.....	8
2.4	Enterprise Application Middleware	10
2.5	Summary	12
Chapter 3:	The Distributed Architecture of Multi-Tiered Enterprise Applications	15
3.1	Introduction to Patterns and Architectural Styles.....	15
3.1.1	Christopher Alexander's Patterns	15
3.1.2	Patterns for Software Systems	16
3.1.3	Architectural Styles.....	17
3.2	Architectural Styles for Enterprise Applications.....	19
3.2.1	Existing Pure Styles	19
3.2.2	The Multi-Tiered Enterprise Application Style	20
3.2.3	Examples.....	23
3.3	Distributed Process Topologies.....	27
3.4	A Pattern Language for Process Topologies	30
3.4.1	Pattern 1: Wrapper Insertion.....	31
3.4.2	Pattern 2: Process Replication	32
3.4.3	Pattern 3: Meshing.....	33
3.4.4	Pattern 4: Data Distribution	34
3.4.5	Pattern 5: Facade Insertion.....	35
3.4.6	Pattern 6: Integration of Subsystems	36
3.5	Discussion	37
3.5.1	Enterprise Applications that Follow Other Styles.....	37
3.5.2	Modeling Tiers as Components	37
3.5.3	Architecture Development Process.....	37
3.5.4	Two-Level Topology Structure.....	38
3.5.5	Patterns Based on Graph Transformations.....	38
3.5.6	Homogeneous Clusters of Data Stores.....	39
3.5.7	Extensions to the Graphical Notation	39
3.6	Summary	39
Chapter 4:	Custom and Adaptable Process Topologies.....	41
4.1	Motivation for Custom and Adaptable Process Topologies	41
4.1.1	Custom Process Topologies.....	41
4.1.2	Adaptable Process Topologies.....	42
4.2	Case Study.....	42
4.3	Challenge: Cross-Process Management of Data Objects	46
4.4	Requirements for Enterprise Application Middleware.....	46
4.4.1	Copying Data Objects across Process Boundaries (Requirement R1)	47
4.4.2	Preserving Object Identity (Requirement R2).....	48
4.4.3	Transitive Data Management (Requirement R3).....	49
4.4.4	Limited Visibility (Requirement R4).....	50
4.4.5	Decoupling of Topology, Data Distribution Scheme, and Application (Req. R5).....	51
4.4.6	A Mechanism for Object and Query Routing (Requirement R6)	51

4.5	Limitations of Existing Middleware.....	52
4.5.1	C1 Connectors from Processes to Data Stores.....	53
4.5.2	C1 Connectors between Processes.....	53
4.5.3	Consequences for Application Developers.....	54
4.6	Discussion.....	56
4.7	Summary.....	57
Chapter 5:	The FPT Architecture.....	59
5.1	Overview of the FPT Architecture.....	60
5.2	Data Distribution Scheme (Part 1).....	62
5.2.1	Nesting of Data Distribution Variants.....	62
5.2.2	Defining a Data Distribution Scheme in Three Steps.....	64
5.2.3	RI-Trees.....	65
5.2.4	Integration of Data Distribution Schemes.....	67
5.2.5	Data Distribution Variants and Nesting in Existing Systems.....	69
5.3	Import/Export Scheme (Part 2).....	70
5.4	Process Topology and Configuration (Part 3).....	71
5.5	Decoupling (Part 4).....	72
5.6	Overview of Copies and Caching (Part 5).....	73
5.7	Transaction Management and Concurrency Control (Part 6).....	75
5.7.1	Approach to Concurrency Control.....	76
5.7.2	Isolation through Multiple Versions.....	76
5.7.3	Concurrency Control and Caching.....	81
5.7.4	Three-Phase FPT Transactions.....	81
5.7.5	Guarantees of Consistency.....	84
5.8	Object and Query Routing (Part 7).....	84
5.8.1	Query Routing.....	85
5.8.2	Object Routing.....	85
5.8.3	Selection of Target Data Stores and Paths.....	86
5.9	Discussion.....	88
5.9.1	Optimistic Concurrency Control and Contention.....	88
5.9.2	Simple Implementation a Design Rationale.....	89
5.9.3	Fixed Home Data Stores per Data Object.....	89
5.9.4	Import/Export Scheme versus Limited Visibility.....	89
5.9.5	Data Distribution in RAID Systems.....	90
5.10	Summary.....	90
Chapter 6:	Proof-of-Concept Implementation.....	93
6.1	Overview.....	94
6.2	Configuration.....	97
6.3	Communication between Object Managers.....	101
6.4	Copies and Versions.....	104
6.4.1	Type-safe Access through Interfaces.....	104
6.4.2	Implementation of Copies and Versions.....	106
6.4.3	Home Data Stores.....	109
6.5	Basic and Advanced Concurrency Control.....	110
6.5.1	Check Types.....	111
6.5.2	Delta Writes.....	112
6.5.3	Automatic and Manual Check Mode.....	113
6.5.4	Advantages.....	114
6.6	Object Caching.....	114
6.7	Database Access and Transaction Management.....	116
6.7.1	Object/Relational Mapping.....	116
6.7.2	Access to Relational Data.....	116
6.7.3	Distributed Transactions.....	119
6.8	Multi-Threading.....	121
6.8.1	Multi-Threaded Communication Plug-ins.....	121
6.8.2	Multi-Threaded Database Access.....	122
6.8.3	Multi-Threaded Object Manager Core.....	122

6.9	Object and Query Routing.....	124
6.9.1	Basic Approach.....	124
6.9.2	Initial Parameters and Routing Matrices.....	126
6.9.3	Constraints	127
6.9.4	Routing Function Implementation	128
6.10	Discussion	129
6.10.1	Programming Language.....	129
6.10.2	Object Managers in Thin Clients	130
6.10.3	Heterogeneous Topologies.....	130
6.11	Summary	130
Chapter 7:	Evaluation	133
7.1	Scenario 1: Custom Topologies and Adaptability	133
7.1.1	Scenario Outline	134
7.1.2	Realization of Evolutionary Steps.....	135
7.1.3	Scenario Conclusions.....	142
7.2	Scenario 2: Basic Transaction Performance	142
7.2.1	Setup for Fine-Grained Transactions	142
7.2.2	Results for Fine-Grained Transactions	145
7.2.3	Setup for Coarse-Grained Transactions	146
7.2.4	Results for Coarse-Grained Transactions	147
7.2.5	Scenario Conclusions.....	148
7.3	Scenario 3: Impact of Additional Tiers on Response Time	148
7.3.1	Setup	149
7.3.2	Results	150
7.3.3	Scenario Conclusions.....	151
7.4	Scenario 4: Coarse-Grained versus Fine-Grained Remote Data Access	151
7.4.1	Setup	152
7.4.2	Results	155
7.4.3	Scenario Conclusions.....	157
7.5	Scenario 5: Process Replication	158
7.5.1	Setup	158
7.5.2	Results	159
7.5.3	Scenario Conclusions.....	159
7.6	Scenario 6: Data Distribution	159
7.6.1	Setup	159
7.6.2	Results	161
7.6.3	Scenario Conclusions.....	163
7.7	Scenario 7: Hotspot Data.....	164
7.7.1	Setup	164
7.7.2	Results for the Standard Implementation Variant	167
7.7.3	Results for the Optimized Implementation Variant	168
7.7.4	Scenario Conclusions.....	169
7.8	Summary	170
Chapter 8:	Conclusions.....	171
8.1	Summary of Contributions	172
8.2	Future Work	173
References	177
Appendix A:	Grammar of Configuration Files.....	187
Appendix B:	Source Code of Sample Transactions.....	189
Appendix C:	Hardware and Software Used in Performance Tests.....	193
Appendix D:	Anhang gemäß Promotionsordnung	195

Acknowledgements

First, I would like to thank Prof. Heinz Schweppe, my first advisor, for his guidance, encouragement, and for supporting my work. Many thanks go to my second advisor, Dr. Ralf-Detlef Kutsche, for his support and many fruitful discussions. I am also grateful to Prof. Herbert Weber and Prof. Johann-Christoph Freytag for their valuable comments and for reviewing my dissertation. Special thanks go to all members of the Databases and Information Systems group of the Freie Universität Berlin and to the members of the Graduate School in Distributed Information Systems. I will never forget the great time we had together.

Last but not least, I would like to thank my family, in particular my wife Oxana, for everything.

This research was supported by the German Research Society, Berlin-Brandenburg Graduate School in Distributed Information Systems (DFG grant GRK 316).

