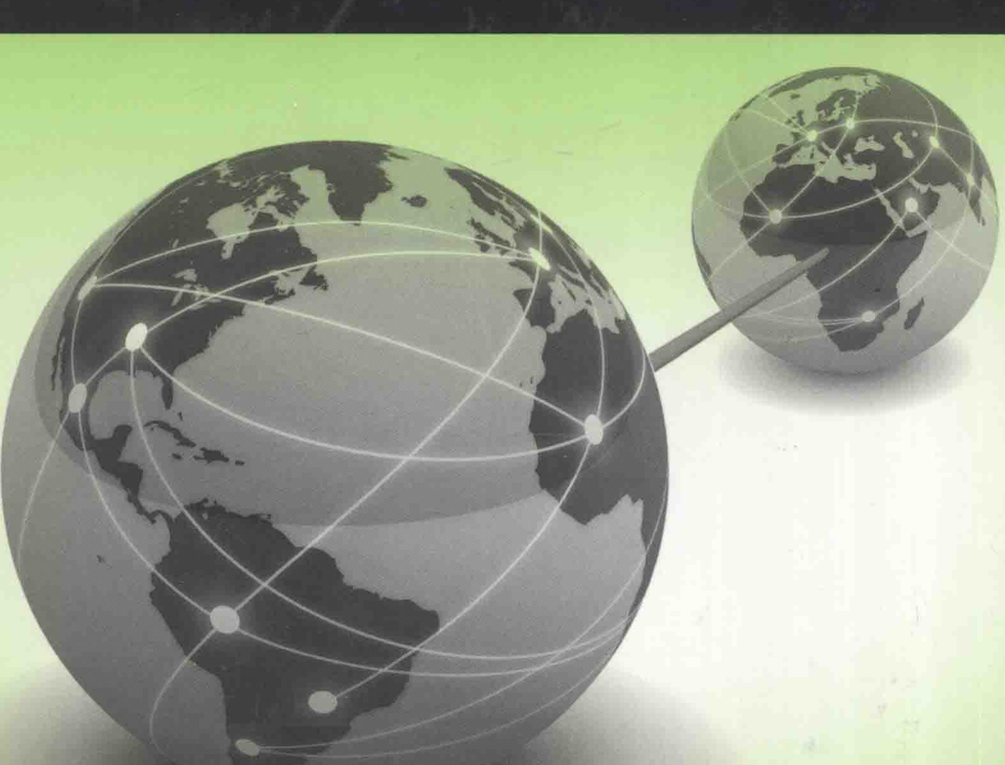
# Distributed Database Management Systems

## A Practical Approach

Saeed K. Rahimi and Frank S. Haug



WILEY IEEE IEEE computer society WWW. LINK AVAILABLE

# DISTRIBUTED DATABASE MANAGEMENT SYSTEMS

## A Practical Approach

**SAEED K. RAHIMI**
University of St. Thomas

**FRANK S. HAUG**
University of St. Thomas

IEEE computer society

WILEY

A JOHN WILEY & SONS, INC., PUBLICATION

Published by John Wiley & Sons, Inc., Hoboken, New Jersey.
Published simultaneously in Canada.

***Library of Congress Cataloging-in-Publication Data is available.***

ISBN 978-0-470-40745-5

Printed in the United States of America.

10 9 8 7 6 5 4 3 2 1

# DISTRIBUTED DATABASE MANAGEMENT SYSTEMS

*To my mother, Behjat, and my father, Mohammad—though they were not given the opportunity to attend or finish school, they did everything they could to make sure that all of their seven children obtained college degrees.*
S. K. R.

*To my mother who taught me to love reading, learning, and books; my father who taught me to love mathematics, science, and computers; and the rest of my family who put up with me while we wrote this book —this would not have been possible without you.*
F. S. H.

# PREFACE

A centralized **database management system (DBMS)** is a complex software program that allows an enterprise to control its data on a single machine. In the past two decades, there have been many mergers and acquisitions, which have resulted in organizations owning more than one DBMS. Not all of these systems came from the same vendor. For example, it is common for an enterprise to own a mainframe database that is controlled by IBM DB2 and a few other smaller workgroup (or departmental) databases controlled by Oracle, Microsoft SQL Server, Sybase, or other vendors. Most of the time, users need to access their own workgroup database. Yet, sometimes, users need to access data in some other workgroup's database, or even in the larger, enterprise-level database. The need to share data that is dispersed across an enterprise cannot be satisfied by centralized DBMS software. To address this requirement, a new breed of software to manage dispersed (or distributed data) called a **distributed database management system (DDBMS)** is required.

A DDBMS maintains and manages data across multiple computers. A DDBMS can be thought of as a collection of multiple, separate DBMSs, each running on a separate computer, and utilizing some communication facility to coordinate their activities in providing shared access to the enterprise data. The fact that data is dispersed across different computers and controlled by different DBMS products is completely hidden from the users. Users of such a system will access and use data as if the data were locally available and controlled by a single DBMS.

This book addresses the issues in design and development of a DDBMS. In the chapters that follow we will address the issues involved in developing such a system and outline different approaches for solving these problems. We will also present two Java-based frameworks and one Microsoft .NET-based framework that provide the underlying structure needed to develop a DDBMS.

## ACCOMPANYING WEBSITE

The official website for this book (www.computer.org/Rahimi_Haug) contains important information for the book, such as links to the starter kit software bundles for all three starter kits (JMS-SKIT, J2EE-SKIT, and DNET-SKIT), error corrections from the text, and any updates to the source code, book content, or starter kits.

## STRUCTURE OF THE BOOK

This book has been organized into 3 distinct units. Unit 1 addresses distributed database theory. These chapters explore the issues with various aspects of a distributed database system and discuss the various techniques and mechanisms that are available to address these issues. Unit 2 (Chapters 10 through 14) focuses on the "state of the practice" for distributed databases. The initial chapters in this unit present a general data modeling overview and discuss several data modeling alternatives. The later chapters focus on the architectural requirements and present architectural alternatives including the traditional, top–down design of a homogenous distributed database, the bottom–up design of a heterogeneous federated database, and two new, nontraditional architectural approaches that focus on environments with more dynamic deployment characteristics. Unit 3 (Chapters 15 through 19) focuses on distributed database implementation. The chapters in this unit examine three platforms suitable for distributed database development. Three starter kits are discussed, along with the platforms on which they are built. The platforms discussed include the Java Message Service (JMS), the Java 2 Enterprise Edition (J2EE), and the Microsoft .NET Framework.

### Unit 1: Theory

In order to design and ultimately build a DDBMS, we must first understand what a distributed database is, what a DDBMS does, and the issues it must overcome. First, this unit provides an overview of the DDBMS and its required functionality. Then, it examines each major subset in detail. By examining both the "big picture" and the "little details," this unit lays the theoretical foundation for understanding the internal processing of the DDBMS.

### Unit 2: State of the Practice

Because most practitioners will probably not have the luxury of designing a distributed database environment (DDBE) completely from scratch, this unit examines the architectural alternatives (and issues) that a practitioner will likely encounter in the real world. First, we review the differences between conceptual, logical, and physical data models. Then, this unit focuses on several logical data modeling alternatives (along with some practical physical considerations) used "in practice" in the real world today. In order to address the "state of the practice," this unit explores four (two traditional and two alternative) DDBE architectures that are designed to be used in an environment with existing DBMS implementations. Then the general requirements for any platform capable of implementing a DDBE architectural alternative (including those that are yet-to-be-invented by the reader) are explored.

### Unit 3: Implementation

Because architectures are large, complicated designs, even the "little details" discussed in them can seem "miles above the ground" compared to the more "down to earth" considerations that need to be addressed when attempting to create an actual implementation. Recent advances in software development have made it much easier to implement complex, distributed applications. In fact, there are multiple alternatives to choose from when selecting the platform, development, and deployment details. These alternatives all have various facilities to support distributed application development and deployment. In order for a practitioner to become proficient in any particular alternative, they need to make a substantial investment of time, effort, and perhaps money as well. Therefore, in the real world there are several alternatives to choose from, but there are also strong pressures to use a particular general approach, or even a particular, specific alternative. Having said that, we must recognize that there are no alternatives specifically targeted at DDBE implementation. This unit attempts to alleviate some of the confusion and provide a concrete place to start development of a real DDBE. This unit examines three implementation alternatives in detail and then for each of the three alternatives, this unit provides a detailed overview and an extensible framework, called a starter kit (SKIT). Each SKIT is discussed in detail. An example extension is presented to demonstrate how the SKIT can be used as the starting point for a real DDBE project implementation.

## CHAPTERS OF THE BOOK

This book consists of 19 chapters as outlined below.

### Chapter 1: Introduction

There are two fundamental types of architecture suitable for implementing a database management system (DBMS). The most well-known and ubiquitous architectural type is called the Centralized DBMS architecture. It is very well defined, very mature, and so common that, when the "Centralized" part of the name is omitted and the term "DBMS architecture" is used, this architecture is assumed by default. The other architectural type is not as common, not as mature, and not as well defined; it is called the Distributed DBMS (DDBMS) architecture. In this introductory chapter, we describe both architectural types and then compare and contrast them. Chapter 1 discusses the strengths and weaknesses of each and motivates the reader to understand which architecture is most appropriate for his/her situation. This chapter also examines different implementation alternatives for distributed query execution, and analyzes master–slave and triangular distributed execution control mechanisms. How to calculate the communication costs for each execution alternative using two examples is also explained.

### Chapter 2: Data Distribution Alternatives

This chapter studies and compares different data distribution design alternatives for a distributed database management system. We analyze fragmentation alternatives—specifically, horizontal, vertical, and hybrid fragmentation. We discuss the impact of supporting data replication and consider data placement issues. The correctness criteria for each of these approaches will also be outlined. We explain how a DDBMS provides fragmentation, location, and distribution transparencies and also discuss how to design a global data dictionary to support these transparencies. We present some examples of different distribution designs and use them to illustrate the impact of different data distribution transparencies on user queries.

**Chapter 3: Database Control**

This chapter focuses on the issues related to security and semantic integrity control of centralized and distributed database systems. If not controlled, the integrity of a database is jeopardized when the database is exposed to concurrent transactions. One of the most fundamental reasons for using transactions is to preserve the integrity of the data; therefore, both centralized and distributed semantic integrity issues are presented and discussed. We consider how to leverage the semantic integrity control mechanisms in our local (centralized) database management systems when implementing distributed integrity control and discuss the impact of data distribution on the distributed and local enforcement implementation. We examine four implementation alternatives for implementing semantic integrity enforcement and evaluate the advantages and disadvantages of each and present a mechanism for analyzing the costs associated with enforcing semantic integrity in a distributed system by focusing on the number of messages required for different implementation alternatives.

**Chapter 4: Query Optimization**

In a centralized system query optimization is either rule-based or cost-based. Regardless of the approach to optimization in a centralized system, the main object of the query optimizer is to minimize response time for the query. The cost components of query optimization are CPU time and disk access time. It turns out that disk access time is the dominant factor in the cost of the query. As such, most query optimizers focus on minimizing the number of disk I/Os and therefore minimizing the disk access time. This chapter will focus on processing queries in a distributed database management system. We extend the concept of query processing in a centralized system to incorporate distribution. In a distributed database management system, the cost components of a query processing time are local CPU time, local disk access time for all servers involved in the query, and the communication cost required for synchronization of these servers activities. Because the communication cost is the dominant cost factor in a distributed query, distributed query optimization needs to focus on minimizing this cost. We discuss optimization alternatives for distributed queries that focus on reducing the number of messages required to execute a query in a distributed system. To minimize the communication cost, one needs to make sure that the system has an optimized distribution design and that the proper allocation of information to different database servers has been made. It has been shown that to solve this problem considering all factors involved is an NP-hard problem. Therefore, the goal is to solve the problem not in an optimal fashion but close to it.

**Chapter 5: Controlling Concurrency**

This chapter covers the concurrency issues in a distributed database management system. We start by defining what a transaction is and the properties that a transaction must satisfy in a database management system and then review concurrency control alternatives for a centralized database management system and outline two classes of concurrency control algorithms (pessimistic and optimistic). We analyze how these alternatives are changed to accommodate the needs of a distributed database management system concurrency control. Because each class of algorithms can be implemented using either locks or timestamps, we will evaluate the advantages and disadvantages for both locking-based and timestamp-based implementations of concurrency control algorithms. Most concurrency control approaches use locking as the main mechanism for providing isolation.

**Chapter 6: Deadlock Handling**

When locks are used, transactions that concurrently run in the system may get into a deadlocked state. A deadlock is defined as a state of the system where two or more transactions are stuck in an infinite waiting cycle. This means that each transaction in the cycle is holding one or more locks and waiting to obtain another lock that is currently being held by one of the other transactions in the cycle. Because each transaction does not release the locks it is holding, all transactions will wait indefinitely. This chapter discusses the three classical approaches for dealing with the deadlock problems in a centralized system: namely, deadlock prevention, deadlock avoidance, and deadlock detection and how these approaches can be used within a distributed database management system, as well as the advantages and disadvantages of each approach. It explains the use of preacquisition of all locks for deadlock prevention approaches and discusses how the wait–die and wound–wait algorithms provide for deadlock avoidance. A centralized and a distributed deadlock detection and removal approach is also presented.

**Chapter 7: Replication Control**

Replicating a table or fragment can improve the availability, reliability, and resiliency of our data (as discussed in Chapter 2) and can greatly improve the performance of our system (especially with respect to query optimization as discussed in Chapter 4). Unfortunately, environments with replication also have more complicated modification issues. This chapter discusses the effect of replication in a distributed database management system. We focus on replication control, which is the techniques we use to ensure mutual consistency and synchronization across all replicated tables and table fragments. Controlling replication is an issue specific to distributed database management systems. There are many alternatives to controlling replication in a DDBMS environment, which we categorize as asynchronous or synchronous replication control algorithms. We discuss the Unanimous, Primary-Copy, Distributed, Voting-Based, Quorum-Based, and Token-Passing-Based replication control algorithms.

**Chapter 8: Failure and Commit Protocols**

A DBMS must guarantee that the contents of a database are maintained consistent even when failures are present in the system. This chapter discusses fault tolerance in a DBMS, which deals with transaction failures, power failures, and computer hardware failures. We examine the commit protocols that are used by a DDBMS to provide for such a guarantee discuss how the DDBMS determines whether a particular transaction is safe to commit or not. Also examineed is how the DBMS fault tolerance mechanism aborts the transaction when it is unsafe, and how the system commits the transaction when all modifications of a transaction have been successfully written to the log. In a centralized system, the log manager is responsible for working with the transaction manager to correctly implement the commit protocols. In a distributed system, on the other hand, multiple DBMS servers need to coordinate their activities to make sure that a global transaction can successfully commit or abort at all DBMS sites where the transaction has made some changes. Therefore, we discuss how the DDBMS implements distributed commit protocols examine alternative distributed commit protocols such as two-phase and three-phase commits. One of the most difficult failures in a distributed system is network partitioning (when a set of computers are networked together but they are isolated from another set of computers that are interconnected). When network partitioning happens, there is a danger that computers in each partition unilaterally decide to commit or abort the transaction. If the two partitions do not decide on the same action (either both committing or both aborting the transaction), the integrity of the database will be lost. Therefore, we also examine a quorum-based commit protocol that deals with network partitioning.

**Chapter 9: DDBE Security**

This chapter discusses the security issues, including authentication, authorization, encryption, and programming techniques for improving data privacy and security in a distributed database environment (DDBE). Since communication is the basis for cooperating database servers in a DDBE, we have to make sure that server communication and user access to data are secure; therefore, we examine private-key and public-key security approaches discuss Secure Sockets Layer (SSL) and Transport Layer Security (TLS) and explain how certificates of authority are used in a distributed database system for components and applications (including web-based applications). We also consider "tunneling" approaches including Virtual Private Network (VPN) and Secure Shell (SSH) and discuss their potential role in a distributed database system.

**Chapter 10: Data Modeling Overview**

This chapter provides an overview of data modeling concepts and techniques, examines data modeling, and discusses its purpose. We consider different ways to create and categorize data models and data modeling languages and next, focus on conceptual data modeling and present three different languages for creating conceptual data models and diagrams. We explore entity relationship modeling in detail and consider some other conceptual modeling techniques. We briefly discuss logical data modeling and how its purpose is different from conceptual data modeling. Similarly, we discuss physical data modeling and how its purpose is different from both conceptual and logical data modeling and then briefly consider some of the nomenclature and notations used in capturing these various types of data models. Finally, we examine how these different types of data modeling coexist within a heterogeneous database environment.

### Chapter 11: Logical Data Models

This chapter examines four logical data modeling languages used by databases in the real world. The relational data model is the first modeling language considered because it is the foundation of many centralized DBMSs. It is also the most-likely logical modeling language for a DDBE and we examine this language in detail. The hierarchical data model and the network data model are two logical modeling languages that are both found in some legacy database systems, particularly on the mainframe. Therefore, we also look at these languages. Object-oriented programming (OOP) has become the de facto standard for many organizations. While OOP applications can access data stored in any format, the most natural format is the format found in an object-oriented DBMS (OODBMS). This format is also found in so-called object persistence engines. Object-oriented databases use an object-oriented modeling language that is fundamentally different from the other modeling languages discussed in this chapter. Some similarities exist between the languages however, and these are discussed. For each modeling language, we discuss the notation and nomenclature, and the rules that need to be followed when converting a conceptual data model into a data model created using one of these logical modeling languages. We briefly compare and contrast the four languages and consider some of the issues that need to be addressed when forward and reverse engineering data models use these languages.

### Chapter 12: Traditional DDBE Architectures

In the real world, several different architectural alternatives can be used instead of the traditional, homogeneous DDBMS, which is designed top–down. Federated databases or multidatabases are integrated existing database servers that are built from the bottom up. In a federated database management system, the data that an organization needs already exists in multiple database servers. These databases may or may not all be of the same type (same data model and/or data modeling language). This heterogeneous set of databases needs to be integrated to provide for uniform and relational based interface for the users of the federated database system. Issues involved in such an integration are data model translation to relational data model and integration of a set of relational data models into a federated view.

### Chapter 13: New DDBE Architectures

Traditionally, a component or subsystem in a DDBE architecture is not an independent process. This chapter explores the architecture of distributed database environments. Most components in a traditional architecture are controlled by one or more subsystems, and most subsystems are themselves designed with a "chain of command" defined for the components inside them. Therefore, the first new architecture presented is one where the components and subsystems work in cooperation with each other rather than working under the control of each other. Another traditional characteristic found in most architectures is the simple fact that the subsystems within a DDBE architecture are not "all created equal." Certain subsystems in the architecture are designed to be more important, more authoritative, or simply larger and more sophisticated than others. This also means that the architecture is somewhat rigid, even when the development and deployment environments provide dynamic deployment and configuration services. Therefore, this chapter also presents another new architecture where the subsystems are closer to being equal peers. This also allows the environment to be more dynamic and allow the peer systems to join and leave the environment with relative ease and flexibility.

**Chapter 14: DDBE Platform Requirements**

A distributed database environment (DDBE) runs on top of a network of computers. Each of these computers has its own operating system and perhaps other systems, such as a local DBMS, that need to be incorporated into the DDBE architecture. Although each individual computer has its own operating system, there is no such thing as a completely distributed operating system environment. This means that we must write a layer of software to sit on top of existing software deployed at each individual computer. We call this layer of software the DDBE Platform. This layer can be architected (and implemented) in many different ways. Rather than focusing on a specific implementation, this chapter focuses on the architectural issues and general DDBE Platform Requirements, in particular, how our DDBE Platform needs to deal with communication, component naming, architectural security, deployment, distributed transaction, and general portability/interoperability issues. This chapter also briefly explores some of the implementations that are capable of providing some (or possibly all) of the DDBE Platform requirements. We examine RPC and RMI mechanisms for Remote Calls, the Java Message Service (JMS) implementation for Remote Messaging. We also present a high-level overview of XML Web Services, which is a relatively new technique for implementing service providers in a cross-platform fashion.

**Chapter 15: The JMS Starter Kit**

This chapter considers how to develop a distributed database management system called a DDBE that can be used to integrate databases from Oracle, Microsoft, IBM, Sybase, and others. We provide a DDBE starter kit, which serves as a starting point for developing our own DDBE components and subsystems and includes a framework and an example extension. It is implemented using Java 2 Standard Edition (J2SE) augmented by an implementation of the Java Message Service (JMS). We can use the framework as a base to get a jumpstart in building a heterogeneous distributed database management system using Java without the additional complexity (or power) of a full J2EE Application Server. We discuss how to use this framework and how to create new extensions in order to implement any additional functionality not currently provided by the framework. Finally, we present the example extension (written using the starter kit) that uses three databases and discuss the steps required to set up and run the example.

**Chapter 16: The J2EE Platform**

This chapter discusses the J2EE platform by presenting a brief overview of the platform and looking at the specific implementation details for various DDBE Platform Services. In particular, we identify the J2EE Services defined in the J2EE Specification that satisfy the requirements we identified in Chapter 14. We also consider the different implementations to choose from at this time, including Apache Geronimo and JBoss and examine the remote-ability support provided by Enterprise Java Beans (EJB) and discuss how other J2EE facilities can be used for DDBE projects. Facilities discussed include the Java Naming and Directory Interface (JNDI), XML Web Services, and RMI, as well as an overview of the Security, Deployment, and Transaction Management facilities that J2EE provides.

**Chapter 17: The J2EE Starter Kit**

Another DDBE starter kit is presented here similar to the one we presented in Chapter 15. However, this starter kit is implemented using the J2EE platform and is capable of being ported to several J2EE implementation alternatives with minimal modifications. We can use this framework as a base to get a jumpstart in building a heterogeneous distributed database management system using J2EE and the full power of the subsystems it provides. In particular, the remote-ability and distributed transaction management facilities make this a more powerful (and more complicated) implementation than the JMS/J2SE approach we used in Chapter 15. The starter kit includes a framework and an example extension. We discuss how to use this framework and how to create new extensions in order to implement any additional functionality not currently provided by the framework. Finally, we present a simple example extension (written using the starter kit) that uses three databases and discuss its design.

**Chapter 18: The Microsoft .NET Platform**

This chapter uses the Microsoft .NET Framework as a potential DDBE development platform by presenting a brief overview of the platform and looking at the specific implementation details for various DDBE Platform Requirements previously identified in Chapter 14. In particular, we examine the TCP/IP Remoting and HTTP Remoting implementations for Remote-Code Execution and also explore the general messaging facilities and support for XML Web Services. We discuss how the Microsoft .NET Framework-based DDBE platform can provide Directory Services, and by presenting also examine how the platform supports Security, Deployment, and Transaction Management.

**Chapter 19: The DNET Starter Kit**

This chapter presents another DDBE starter kit, similar to the one in Chapter 17. However, this starter kit is implemented using the Microsoft .NET Framework. Therefore, we can use this framework as a base to get a jumpstart in building our own heterogeneous distributed database management system using any of the Microsoft .NET Framework supported programming languages (Visual C#, Visual C++, or Visual Basic). The starter kit leverages the facilities available in the Microsoft .NET Framework-based platform, and we discuss how the facilities identified in Chapter 18 can be used to build our own DDBE projects. The starter kit includes a framework and an example extension we discuss how to use this framework and how to create new extensions in order to implement any additional functionality not currently provided by the framework. Finally, a simple example extension (written using the starter kit) that uses three databases and discuss its design is presented.

## USING THIS BOOK EFFECTIVELY

The following describes different paths or "tracks" through the book:

- When this book is used in a classroom environment, it is recommended that Unit 1 be used in its entirety and in the order presented. Depending on the scope of the class, selected information from Unit 2 should also be included based on the degree of heterogeneity discussed in the class. This track would most likely include all of Unit 1 (Chapters 1–9) and at least some parts of Chapters 10 and 11.
- If the class includes any implementation activities or implementation considerations, then once again we recommend that Unit 1 be included, but now Chapters 10, 11, and 14 also have a significant contribution to make. This track should also include the appropriate chapters from Unit 3 based on the implementation platform chosen. In other words, it should include either the J2EE chapters (Chapters 16 and 17), Microsoft .NET Framework-based chapters (Chapters 18 and 19), or the JMS chapter (Chapter 15).
- For readers planning to design their own DDBMS architecture in the real world, Chapter 1 and the chapters in Unit 2 (Chapters 10–14) provide the necessary background to understand the architectural requirements of the system. Once again, the appropriate chapters from Unit 3 can be selected based on the development platform(s) selected for the reader's implementation.
- Readers working with an existing DDBMS architecture can use Unit 1 as a reference for the theoretical underpinnings behind the component and subsystem designs in their implementation. Unit 2 can be used to understand the architectural issues or perhaps to consider changes to their architecture. Unit 3 can be selected based on the existing development environments or perhaps to implement prototypes in an alternative development environment.

## ACKNOWLEDGMENTS

Our special thanks go to the following people: Dr. Brad Rubin from the University of St. Thomas, Graduate Programs in Software, for contributing the chapter on security; Dyanne Haug and Patricia Rahimi for their valuable feedback and review of the first draft of this book: we greatly appreciate their time and patience; Dr. Khalid Albarrak from IBM, for helping us with acquiring the InfoSphere Federation Server software and installing and debugging it, and for providing feedback on our test setup and test results; and Saladin Cerimagic from the University of St. Thomas, Graduate Programs in Software, for his help with and discussions about concurrency.

SAEED K. RAHIMI
FRANK S. HAUG

# CONTENTS