

Douglas Schmidt  
Michael Stal  
Hans Rohnert  
Frank Buschmann



# PATTERN-ORIENTED SOFTWARE ARCHITECTURE

**Volume 2**

**Patterns for Concurrent  
and Networked Objects**

 **WILEY**



WILEY SERIES IN  
**SOFTWARE DESIGN PATTERNS**

# Foreword

Middleware is the set of services, protocols, and support utilities providing the ‘plumbing’ that makes modern distributed systems and applications possible—the infrastructure that underlies web services, distributed objects, collaborative applications, e-commerce systems, and other important platforms. Not long ago, the term middleware was rarely heard, and middleware developers were rarer still. But over the past decade, the term, the research and practice, and its impact have become ubiquitous. Yet, until now, there has not been a book describing how to construct networked and concurrent object-oriented (OO) middleware, so its design has remained something of a black art. This book demystifies middleware construction, replacing the need to have an expert looking over your shoulder with well-reasoned, empirically-guided accounts of common design problems, forces, successful solutions, and consequences.

As is true for most concepts, nailing down the boundaries of middleware is hard. Conventionally, it consists of the software needed to build systems and applications, yet is not otherwise an intrinsic part of an operating system kernel. But it is not always possible to find middleware where you first look for it: middleware can appear in libraries and frameworks, operating systems and their add-ons, Java virtual machines and other run-time systems, large-grained software components, and in portions of end-products such as web services themselves.

This book is not a textbook surveying middleware or the types of applications and distributed system architectures you can devise using middleware. It instead presents a pattern language that captures the design steps leading to

the construction of the OO communication support involved in most middleware. Many of the patterns described in this book also have utility in both higher-level and lower-level systems and applications that are not based directly upon middleware.

This book emphasizes practical solutions over theoretical formalisms. The basic ideas behind many presented patterns are well-known to experienced system developers—for example, dispatching, demultiplexing, callbacks, and configuration—and are sometimes variants of more general OO patterns—for example, proxies, adapters, and facades. This book's main contribution centers on in-depth engineering solutions based upon these ideas. Middleware developers must resolve a wide range of forces including throughput, responsiveness, dependability, interoperability, portability, extensibility, and accommodating legacy software. The diversity and severity of these forces accounts for the complexity of middleware patterns, as opposed to those seen in smaller-scale OO applications and concurrent programming.

The multitude of such forces, combined with years of engineering experience, often lead to a multitude of design considerations and engineering trade-offs separating an idea from its expression in middleware frameworks. The pattern description format used in this book helps to simplify this process by presenting solutions as series of concrete design steps. Many of these steps in turn invoke additional patterns. Together they form a pattern language, enabling developers to traverse from pattern to pattern while designing services and applications.

As mentioned by the authors, some of the ideas and techniques discussed in this book are complementary to those seen for example in W. Richard Stevens's pioneering books (e.g., [Ste98]) on network programming. The main

point of departure is the unrelenting focus on higher-level design issues. Rather than, for example, discussing the ins and outs of the Unix `select()` call, this book explains how to build a composable and flexible framework—a Reactor—based on `select()` and other operating system calls.

One of the implicit themes of this book is how to apply the bits and pieces of functionality dealing with I/O, threading, synchronization, and event demultiplexing offered by contemporary platforms as the foundation for constructing higher-level frameworks and components. The primary emphasis on C/C++ on Unix and Microsoft operating systems does not detract from this theme. For example, Java programmers will find a few minor disconnects in cases where Java already directly implements some of the patterns discussed in this book, for example, Scoped Locking, or provides frameworks structured in accord with particular implementations of patterns, such as the JavaBeans framework's support of configurable components, as well as a few where Java lacks access to underlying system mechanisms, such as synchronous event demultiplexing.

However, readers most familiar with Java, Smalltalk, and other OO programming languages will still profit from the central ideas conveyed by the patterns, can better appreciate how and why some became directly supported in language features and libraries, and will be able to construct useful components based upon other patterns. As an example, until the advent of `java.nio`, Java did not provide access to system constructs useful for asynchronous I/O. However, after referring to a description of the Proactor pattern described in this book, I once put together a Java version that simulated the demultiplexing step via a simple spin-loop thread that checked for I/O availability across multiple channels. This was less



efficient, but was perfectly adequate within its intended usage context.

Over the years, some of the accounts in this book, such as Reactor, have evolved from descriptions of design inventions to design patterns. Everyone constructing portable OO middleware has written or used at least one Wrapper Facade. But early presentations of several other patterns now contained in this book also discussed novel contributions about their design. It was at first a bit uncertain whether such descriptions should be considered as patterns, which must be time-proven, independently (re)discovered solutions. However, over time, the authors and the OO middleware community have become more and more confident that the patterns in this book do indeed capture the essence of key forces and design issues, and have witnessed the described solutions being used over and over again across diverse usage contexts.

I invite you to share in this phenomenon. By reading—and especially, using—the material in this book, you’ll see why pattern names such as Reactor and Proactor have become as common among OO middleware developers as have Decorator and Observer among OO GUI developers.

*Doug Lea*  
*State University of New York at Oswego*

## About This Book

Patterns have taken the software development community by storm. Software developers have been enthusiastic about patterns ever since the seminal work *Design Patterns – Elements of Reusable Object-Oriented Software* [GoF95]. Its successors, such as the Pattern Languages of Programming Design (PLoPD) series [PLoPD1] [PLoPD2] [PLoPD3] [PLoPD4] and *A System of Patterns* [POSA1]<sup>1</sup> have further fanned the burning interest in patterns kindled originally by earlier work on software idioms [Cope92], patterns for building architectures [Ale79] [AIS77], and patterns for cultural anthropology [Bat97].

This book, *Patterns for Concurrent and Networked Objects*, is the second volume in the *Pattern-Oriented Software Architecture* (POSA) series. Like its predecessor, *A System of Patterns* [POSA1], it documents patterns and best practices that represent concrete, well-proven and useful techniques for building industrial-strength software systems. These patterns and best practices can and have been applied to applications in a wide range of domains, including telecommunications and data communications, financial services, medical engineering, aerospace, manufacturing process control, and scientific computing. They also form the basis of popular distributed object computing middleware, such as CORBA [OMG98c], COM+ [Box97], Java RMI [WRW96], and Jini [Sun99a].

Moreover, all the patterns in this book build on the same solid conceptual foundation as those in the first POSA volume. For example, we use the same pattern categorization schema, the same pattern description format, and present examples and known uses in multiple programming languages, including C++, Java, and C.

*Patterns for Concurrent and Networked Objects* thus follows the same philosophy and path as *A System of Patterns* and has the same ‘look and feel’.

In contrast to *A System of Patterns*, however, which covered a broad spectrum of general-purpose patterns, this book has a more specific focus: *concurrency* and *networking*. All the patterns in this book center on these two areas, allowing us to discuss many topics related to concurrency and networking in more depth than would be possible if the book contained patterns from many unrelated domains. The patterns in this book therefore complement the general-purpose patterns from *A System of Patterns* in these increasingly important areas of software development.

Yet we focus on *general, domain-independent* patterns for concurrent and networked applications and middleware. Our goal is to increase the likelihood that the patterns in this book will help projects in your daily work. Therefore, we do not cover patterns in this book that are *specific* to a particular application domain, such as those in [DeBr95] [Mes96] [ACGH+96], which address networking aspects that pertain to the telecommunication domain.

By focusing on general domain-independent patterns for concurrency and networking, this book also complements existing literature in concurrent network programming and object-oriented design:

- Literature on concurrent network programming generally focuses on the *syntax* and *semantics* of operating system APIs, such as Sockets [Ste98], POSIX Pthreads [Lew95], or Win32 threads [Ric97], that mediate access to kernel-level communication frameworks, such as System V STREAMS [Ris98] [Rago93], available on popular operating systems. In

contrast, this book describes how to *use* these APIs effectively in the design and programming of high-quality concurrent and networked systems.

- Literature that addresses higher-level software design and quality factors [Boo94] [Mey97] [DLF93] generally has not focused on the development of concurrent and networked applications. Bridging this gap is the topic of this book.

Another way in which *Patterns for Concurrent and Distributed Objects* differs from *A System of Patterns* is that its patterns constitute more than just a catalog or system of patterns. Instead, they augment each other synergistically, providing the foundation of a *pattern language* for concurrent and networked software. When combined with patterns from other patterns literature, we describe how this pattern language can and has been used to build sophisticated concurrent and networked software systems and applications, web services, and distributed object computing middleware, as well as the underlying operating system networking protocols and mechanisms.

Yet we separate the description of the individual patterns from the discussion of how they form a pattern language. The patterns themselves are first described in a self-contained manner, so that they can be applied in the context that is most useful. A subsequent chapter then describes how the patterns interact and how they are complemented by other patterns.

It is important to note, however, that many patterns in this book can be applied outside the context of concurrency and networking. To illustrate the breadth of their applicability we present known uses from other domains, such as component-based or interactive software systems. In

addition, we give examples of how these patterns apply to situations experienced in everyday life.

Some patterns may be familiar, because preliminary versions of them were published in the PLoP book series [PLoPD1] [PLoPD2] [PLoPD3] [PLoPD4], and the C++ Report magazine. In this book, however, we have improved upon the earlier versions considerably:

- This is the first time they have been woven into a single document, which helps to emphasize the pattern language they express.
- We have rewritten and revised these patterns substantially based on many suggestions for improvement we received at conferences and workshops, via e-mail, as well as from intensive internal reviewing and reviews provided by our shepherds.
- The patterns have been converted to the POA pattern format and have a consistent writing style.

## **Intended Audience**

Like our earlier book *A System of Patterns*, this volume is intended for professional software developers, particularly those who are building concurrent and networked systems. It helps these software professionals to think about software architecture in a new way and supports them in the design and programming of large-scale and complex middleware and applications.

This book is also suitable for advanced undergraduates or graduate students who have a solid grasp of networking and operating systems, and who want to learn the core principles, patterns, and techniques needed to design and implement such systems effectively.

# Structure and Content

*Patterns for Concurrent and Distributed Objects* can be used as a text book and read from cover to cover, or used as a reference guide for exploring the nuances of specific patterns in detail.

The first chapter, *Concurrent and Networked Objects*, presents an overview of the challenges facing developers of concurrent and networked object-oriented applications and middleware. We use a real example, a concurrent Web server, to illustrate key aspects of these domains, including service access and configuration, event handling, synchronization, and concurrency.

[Chapters 2](#) through [5](#) form the main part of the book. They contain patterns, the ‘real things’ [U2], that codify well-established principles and techniques for developing high-quality concurrent and networked systems. We hope these patterns will be useful role models for developing your own concurrent and networked applications, and for documenting patterns that you discover.

[Chapter 6](#), *Weaving the Patterns Together*, discusses how the patterns in [Chapters 2](#) through [5](#) are interconnected. We also show how they can be connected with other patterns in the literature to form a pattern language for concurrent networked systems and middleware. As mentioned earlier, some patterns are also applicable outside the context of concurrent and networked systems. For these patterns we summarize the scope of their applicability.

[Chapter 7](#), *The Past, Present, and Future of Patterns*, revisits our 1996 forecast on ‘where patterns will go’, published in the first volume of the *Pattern-Oriented Software Architecture* series. We discuss the directions that patterns actually took during the past four years and

analyze where patterns and the patterns community are now. Based on this retrospection, we revise our vision about future research and the application of patterns and pattern languages.

The book ends with a general reflection on the patterns we present, a glossary of frequently used terms, an appendix of notations, an extensive list of references to work in the field, a pattern index, a general subject index, and an index of names that lists all persons who helped us shaping this book

Supplementary material related to this book is available online at <http://www.posa.uci.edu/>. This URL also contains links to the ACE and TAO source code that contains C++ and some Java examples for all the patterns in this book.

There are undoubtedly aspects of concurrent and networked object systems that we have omitted, or which will emerge over time when applying and extending our pattern language in practice. If you have comments, constructive criticism, or suggestions for improving the style and content of this book, please send them to us via electronic mail to [patterns@mchp.siemens.de](mailto:patterns@mchp.siemens.de). We also welcome public discussion of our entire work on patterns. Please use our mailing list, [siemens-patterns@cs.uiuc.edu](mailto:siemens-patterns@cs.uiuc.edu), to send us feedback, comments, and suggestions.

Guidelines for subscription can be found on the patterns home page. Its URL is <http://hillside.net/patterns/>. This link also provides an important source of information on many aspects of patterns, such as available and forthcoming books, conferences on patterns, papers on patterns, and so on.

## Acknowledgments

It is a pleasure for us to thank the many people who supported us in creating this book, either by sharing their knowledge with us or by reviewing earlier drafts of its parts and providing useful feedback.

Champion review honors go to Regine Meunier, Christa Schwanninger, Martin Botzler, Lutz Dominick, Prashant Jain, Michael Kircher, Karl Pröse, and Dietmar Schütz, our esteemed colleagues. They spent much of their valuable time helping to review the manuscript in the countless writer's workshops we ran, thus helping us to polish and shape the final content of this book. Similarly, we are grateful to members of the Distributed Object Computing (DOC) group—Tim Harrison, Prashant Jain, Carlos O’Ryan, and Irfan Pyarali—who co-authored initial versions of six patterns in this book. Together with the four lead authors, these researchers form the POSA team at Siemens in Munich, Washington University in St. Louis, and the University of California, Irvine.

We also owe most grateful thanks to Peter Sommerlad, Chris Cleeland, Kevlin Henney, and Paul McKenney. Peter, our shepherd, reviewed all our material in depth, focusing on its correctness, completeness, consistency, and quality. Chris, Kevlin, and Paul, our peer reviewers, provided us with additional detailed feedback. All four contributed significantly to improving *Patterns for Concurrent and Networked Objects*.

We are also grateful to the Software Architecture Group at University of Illinois at Urbana Champaign, including Federico Balaguer, John Brant, Brian Foote, Alejandra Garrido, Peter Hatch, Ralph Johnson, Dragos Manolescu, Brian Marick, Hiroaki Nakamura, Reza Razavi, Don Roberts, Les Tyrrell, Joseph W. Yoder, Wanghong Yuan, Weerasak Witthawaskul, and Bosko Zivaljevic, who ran writer's workshops on many POSA2 patterns. They sent



comments that helped us improve the correctness and comprehensibility of the book.

Many others from around the world provided feedback on earlier versions of the book, including Giorgio Angiolini, Brad Appleton, Paul Asman, David Barkken, John Basrai, Joe Bergin, Rainer Blome, Don Box, Martina Buschmann, Tom Cargill, Chuck and Lorrie Cranor, James O. Coplien, Ward Cunningham, Gisela Ebner, Ed Fernandez, Erich Gamma, Sonja Gary, Luciano Gerber, Bob Hanmer, Neil Harrison, Michi Henning, David Holmes, Tom Jordan, Fabio Kon, Bob Laferriere, Greg Lavender, Doug Lea, John MacMillan, Mittal Monani, Duane Murphy, Jaco van der Merwe, Michael Ogg, Bill Pugh, Dirk Riehle, Linda Rising, Wolfgang Schroeder, Richard Toren, Siva Vaddepuri, John Vlissides, Roger Whitney, and Uwe Zdun. The *Credits* section of our patterns outline how their valuable contributions helped us polish this book.

We are also heavily indebted to all members, past and present, of the DOC group at Washington University in St. Louis, the University of California, Irvine, Object Computing Inc., and Riverace, who reified, refined, and optimized all the patterns presented in this book into components and frameworks in the ACE and TAO middleware projects. This inspirational group includes Everett Anderson, Alex Arulanthu, Shawn Atkins, Darrell Brunsch, Luther Baker, Matt Braun, Chris Cleeland, Angelo Corsaro, Sergio Flores-Gaitan, Chris Gill, Pradeep Gore, Andy Gokhale, Priyanka Gontla, Myrna Harbison, Tim Harrison, Shawn Hannan, John Heitmann, Joe Hoffert, James Hu, Steve Huston, Prashant Jain, Vishal Kachroo, Ray Klefstad, Yamuna Krishnamurthy, Michael Kircher, Fred Kuhns, David Levine, Ebrahim Moshiri, Michael Moran, Sumedh Mungee, Bala Natarjan, Ossama Othman, Jeff Parsons, Kirthika Parameswaran, Krish Pathayapura, Irfan Pyarali, Carlos O’Ryan, Malcolm Spence, Marina

Spivak, Naga Surendran, Selcuk Uelker, Nanbor Wang, Seth Widoff, and Torben Worm. We would also like to acknowledge the substantial contribution of the thousands of ACE and TAO users around the world who have applied and enhanced the patterns and framework components described in this book over the past decade. Without their support, constant feedback, and encouragement we would never have written this book.

Special thanks go to Johannes Nierwetberg, Lothar Borrmann, and Monika Gonauser for their managerial support and backing at the software engineering labs of Corporate Technology of Siemens AG, Munich, Germany. We also thank Calinel Pasteanu at the Communication Devices business unit of Siemens AG in Munich for understanding the conflict between the chores of writing this book and the pressures of delivering products in 'Internet time'.

We are also grateful for the support from colleagues and sponsors of our research on patterns and the ACE and TAO middleware frameworks, notably the contributions of Ron Akers (Motorola), Al Aho (Lucent), Steve Bachinsky (SAIC), Detlef Becker (Siemens), Jim Blaine (Washington University), John Buttitto (Motorola), Becky Callison (Boeing), Wei Chiang (Nokia), Russ Claus (NASA), Joe Cross (Lockheed Martin), Bryan Doerr (Boeing), Karlheinz Dorn (Siemens), Sylvester Fernandez (Lockheed Martin), Andreas Geisler (Siemens), Helen Gill (DARPA), Trey Grubbs (Raytheon), Jody Hagins (ATD), Andy Harvey (Cisco), Thomas Heimke (Siemens), Kalai Kalaichelvan (Nortel), Arvind Kaushal (Motorola), Steve Kay (Tellabs), Chandra Kintala (Lucent), Gary Koob (DARPA), Sean Landis (Motorola), Rick Lett (Sprint), Joe Loyall (BBN), Mike Masters (NSWC), Ed Mays (US Marine Corps), John Mellby (Raytheon), Dave Meyer (Virtual Technology), Eileen Miller (Lucent), Stan Moyer (Telcordia), Russ Noseworthy (Object

Sciences), Guru Parulkar (Cisco), Dan Paulish (Siemens), James Plamondon (Microsoft), Dieter Quehl (Siemens), Lucie Robillard (US Air Force), Allyn Romanow (Cisco), Rick Schantz (BBN), Steve Shaffer (Kodak), Dave Sharp (Boeing), Naval Sodha (Ericsson), Brian Stacey (Nortel), Paul Stephenson (Ericsson), Umar Syid (Hughes), Dave Thomas (OTI), Lothar Werzinger (Krones), Shalini Yajnik (Lucent), and Tom Ziomek (Motorola).

Very special thanks go to Steve Rickaby, our copy editor, for enhancing our written material. In addition, we thank our editor, Gaynor Redvers-Mutton, and everyone else at John Wiley & Sons who made it possible to publish this book. This is the second book fostered by Gaynor and Steve. Their support has been superb and we look forward to working with them on forthcoming POSA volumes.

Finally, we would like to express our deepest gratitude to the late Richard Stevens, whose seminal books inspired us to explore the wonders of network programming so many years ago. His spirit pervades this book.

1. We reference *A System of Patterns* as [POSA1] rather than by author. The same is true for this book, which we reference as [POSA2]. We use this convention to avoid a particular POSA volume being associated with a single author in reader's minds, in particular the first name on the book's cover.

# About The Authors

## Douglas C. Schmidt

Dr. Douglas Schmidt is an Associate Professor in the Electrical and Computer Engineering department at the University of California, Irvine, USA. He is also serving as a Program Manager in the Information Technology Office (ITO) at DARPA, leading the national research effort on middleware. Before this he was an Associate Professor and Director of the Center for Distributed Object Computing in the Department of Computer Science at Washington University in St. Louis, Missouri. His research focuses on design patterns, optimization principles, and empirical analyses of object-oriented techniques that facilitate the development of high-performance and real-time distributed object computing middleware running over high-speed networks and embedded system interconnects.

Doug is an internationally-recognized expert on distributed object computing patterns, middleware frameworks, real-time CORBA, and open-source development. He has published widely in top technical journals, conferences, and books. He was editor of the *C++ Report* magazine for several years and has co-edited several popular books on patterns [PLoPD1] and frameworks [FJS99a] [FJS99b]. In addition to his academic research, Doug has led the development of ACE and TAO, which are widely-used open-source middleware frameworks that contain a rich set of reusable components implemented using the patterns presented in this book.

In his 'spare' time, he enjoys ballroom dancing with his wife Sonja, weight-lifting, guitar playing, world history, and

Chevrolet Corvettes.

## **Michael Stal**

Michael Stal joined the Corporate Technology department of Siemens AG in Munich, Germany in 1991. In his previous work he gained extensive experience developing software for compilers and computer graphics. He worked on runtime type information for C++ and served on the C++ standardization group X3J16. Since 1992 Michael's work has focused on the development of concurrent and distributed object-oriented systems using Sockets, CORBA, COM, and Java. Michael is Siemens' primary contact on CORBA at the OMG and is head of the Distributed Object Computing Group at Siemens's Corporate Technology department. He co-authored the first POSA volume *A System of Patterns*.

Michael's main research interests focus on methods for developing distributed systems efficiently and on patterns for describing the architecture of middleware platforms. In this context he has published articles in many magazines and given talks at many conferences. In addition, he is Editor-in-Chief of Java Spektrum, the major German magazine on the Java platform, as well as a columnist and member of the advisory board for Objektspektrum, the major German magazine on object technology.

In his spare time Michael attends soccer matches, supports his favorite team Bayern München, visits Munich beer gardens, tries to prevent his cats from destroying his apartment, watches movies, and reads books on physics, philosophy and humor. He is fan of Douglas Adams, Scott Adams, and Terry Pratchett.

## **Hans Rohnert**

Dr. Hans Rohnert is a Senior Software Engineer at the Communication Devices business unit of Siemens AG in Munich, Germany. His primary aims are exploiting promising software technologies and introducing them into new products, such as next-generation mobile phones. His professional interests are software architecture, design patterns, and real-world programming. He has presented numerous talks on subjects ranging from dynamic graph algorithms to embedded Java virtual machines.

Hans is currently a member of the expert groups defining the small footprint KVM Java virtual machine and its libraries for use in small devices. His programming projects have included server-side modules for embedded servers, work flow in C++, base support for ATM switching, Java GUI front-ends for CORBA clients, and HTTP clients. He is also a co-author of the first POSA volume *A System of Patterns* and a co-editor of the fourth book in the PLoPD series [PLoPD4]. As a graduate student he performed original research on combinatorial algorithms, publishing and lecturing on them early in his career.

Hans is an ambitious tennis player, with more matches lost than won. He also enjoys exploring the nearby mountains, rock-climbing, and cross-country skiing. His most important 'hobby', however, is his family, most notably a new baby born during the hectic final phase of writing this book.

## **Frank Buschmann**

Frank Buschmann is Principal Senior Software Engineer at Siemens Corporate Technology in Munich, Germany. His research interests include object technology, software architecture, frameworks, and patterns. He has published widely in all these areas, most visibly in his co-authorship of the first POSA volume *A System of Patterns*. Frank was a member of the ANSI C++ standardization committee X3J16

from 1992 to 1996. Frank initiated and organized the first conference on patterns held in Europe, EuroPLoP 1996, and is also a co-editor of the third book in the PLoPD series [PLoPD3]. In his development work Frank has led design and implementation efforts for several large-scale industrial software projects, including business information, industrial automation, and telecommunication systems.

When not at work Frank spends most of his time enjoying life with his wife Martina, watching the time go by in Munich beer gardens, having fun biking, skiing, and horse-riding, getting excited when watching his favorite soccer team Borussia Dortmund, dreaming when listening to a performance at the Munich opera, and relaxing with rare Scotch single malts before bedtime.

# Guide To The Reader

*“Cheshire-Puss will you tell me, please, which way I ought to go from here?”*

*“That depends a good deal on where you want to get to,” said the Cat.*

*“I don’t much care where—,” said Alice.*

*“Then it doesn’t matter which way you go,” said that cat.*

*“—so long as I get somewhere” Alice added as an explanation.*

*“Oh, you’re sure to do that,” said the Cat, “if you only walk long enough.”*

*Louis Carroll, “Alice in Wonderland”*

This book is structured so you can read it cover-to-cover. If you know where you want to get to, however, you may want to choose your own route through the book. In this case, the following hints can help you decide which topics to focus upon and the order in which to read them.

## Introduction to Patterns

If this book is your initial exposure to patterns, we suggest you first read the introduction to patterns in [POSA1] and [GoF95], which explore the concepts and terminology related to patterns for software architectures and designs. In particular, all the patterns presented in this book build upon the conceptual foundation for patterns specified in [POSA1]:

- The definition of patterns for software architectures



- The categorization of these patterns into architectural patterns, design patterns, and idioms<sup>1</sup> and
- The pattern description format

Moreover, the implementations of many patterns in this book are enhanced by using patterns from [POSA1] and [GoF95]. To guide the application of the patterns in production software development projects we therefore suggest you keep all three books handy.

## Structure and Content

The first chapter in this book, *Concurrent and Networked Objects*, describes the key challenges of designing concurrent and networked systems. It also outlines the scope and context for the patterns we present. Finally, it presents a case study that applies eight patterns in this book to develop a concurrent Web server.

Sixteen pattern descriptions and one idiom form the main part of this book. We group them into four chapters, corresponding to key problem areas—*service access and configuration*, *event handling*, *synchronization* and *concurrency*—in the development of concurrent and networked middleware and applications. The order in which you read this material is up to you. One approach is to read important core patterns first:

- The Wrapper Facade design pattern (47)<sup>2</sup>
- The Reactor architectural pattern (179)
- The Acceptor-Connector design pattern (285)
- The Active Object design pattern (369)

The other twelve patterns and one idiom in the book are arranged to minimize forward references. You can read

them in any order of course, and we provide page numbers for following references to other patterns within the book. This material completes and complements the concepts defined by the four patterns listed above and covers a range of issues relevant to designing and implementing concurrent and networked objects effectively.

You can also use this book to find solutions to problems you encounter in your projects. Use the overview of our patterns in Chapter 6, *Weaving the Patterns Together*, to guide your search, then locate in Chapters 2 through 5 the detailed descriptions of the patterns you select as potential solutions.

No pattern is an island, completely separated from other patterns. Therefore, Chapter 6, *Weaving the Patterns Together*, also describes how all the patterns in this book can be woven together to form a pattern language for building networked applications and middleware. If you want an overall perspective of the patterns in this book *before* delving into the individual patterns, we recommend you skim the pattern language presentation in Chapter 6 before reading the patterns in Chapters 2 through 5 in depth.

Chapter 7, *The Past, Present, and Future of Patterns* and Chapter 8, *Concluding Remarks* complete the main content of this book. The remainder of the book consists of a glossary of technical terms, an overview of the notations used in the figures, references to related work, and an index of patterns, topics, and names.

## **Pattern Form**

All patterns presented in this book are self-contained, following the [POSA1] pattern form. This form allows us to present both the essence and the key details of a pattern.

Our goal is to serve readers who simply want an overview of the pattern's fundamental ideas, as well as those who want to know how the patterns work in depth.

Each section in our pattern form sets the stage for the subsequent section. For instance, the *Example* section introduces the *Context*, *Problem*, and *Solution* sections, which summarize a pattern's essence. The *Solution* section foreshadows the *Structure* and *Dynamics* section, which then present more detailed information about how a pattern works, preparing readers for the *Implementation* section.

The *Example Resolved*, *Variants*, *Known Uses*, *Consequences* and *See Also* sections complete each pattern description. We include extensive cross-references to help you to understand the relationships between the patterns in this book and other published patterns.

To anchor the presentation of a pattern's implementation activities to production software systems, much of the sample code is influenced by components provided in the ACE framework [Sch97]. If you first want to get an overview of all the patterns you may therefore want to skip over the *Implementation* sections on your initial pass through the book and come back to them when you need to know a particular pattern's implementation details.

Although the pattern form we use in this book incurs some repetition within the pattern descriptions, we have found that this repetition helps readers navigate through the descriptions more effectively by minimizing 'back-tracking'.

In the diagrams that explain the structure and behavior of our patterns we tried to follow standard UML whenever possible. In few cases, however, UML did not allow us to express ourselves precisely enough. Thus we 'extended' the

standard notation slightly, as specified in the *Notations* chapter.

## Background Reading

Many patterns, particularly Reactor (179), Proactor (215), Half-Sync/Half-Async (423), and Leader/Followers (447), assume you are familiar with the following topics:

- *Object-oriented design techniques*, such as patterns [GoF95] [POSA1] and idioms [Cope92], UML notation [BRJ98], and the principles of structured programming, specifically encapsulation and modularity [Mey97].
- *Object-oriented programming language features*, such as classes [Str97], inheritance and polymorphism [AG98], and parameterized types [Aus98]. Many examples in this book are written in C++, though we present Java known uses for most of the patterns.
- *Systems programming concepts and mechanisms*, such as process and thread management [Lew95] [Lea99a] [Ric97], synchronization [Ste98], and interprocess communication [Ste99].
- *Network services and protocols*, such as client-server computing [CoSte92] and the Internet protocols [Ste93] [SW94].

This book contains an extensive glossary and bibliography to clarify unfamiliar terminology, and suggest sources for information on topics you may want to learn more about. It is not, however, an introductory tutorial on concurrency and network programming. Thus, if you are not familiar with certain topics listed above, we encourage you to do some background reading on the material we recommend in conjunction with reading this book.

- [1.](#) See the *Glossary* for a definition of these pattern categories.
- [2.](#) We adopt the page number notation introduced by [GoF95]. (47) means that the corresponding pattern description starts on page 47.

# Chapter 1

## Concurrent and Networked Objects

*“With the exception of music, we have been trained to think of patterns as fixed affairs. It’s easier and lazier that way, but, of course, all nonsense. The right way to begin to think of the pattern which connects is to think of a dance of interacting parts, pegged down by various sorts of limits.”*

*Gregory Bateson — Cultural Anthropologist*

This chapter introduces topics related to concurrent and networked objects. We first motivate the need for advanced software development techniques in this area. Next, we present an overview of key design challenges faced by developers of concurrent and networked object-oriented applications and middleware. To illustrate how patterns can be applied to resolve these problems, we examine a case study of an object-oriented framework and a high-performance Web server implemented using this framework. In the case study we focus on key patterns presented in this book that help to simplify four important aspects of concurrent and networked applications:

- Service access and configuration
- Event handling
- Synchronization and
- Concurrency

### 1.1 Motivation

During the past decade advances in VLSI technology and fiber-optics have increased computer processing power by 3-4 orders of magnitude and network link speeds by 6-7 orders of magnitude. Assuming that these trends continue, by the end of this decade

- Desktop computer clock speeds will run at ~100 Gigahertz
- Local area network link speeds will run at ~100 Gigabits/second
- Wireless link speeds will run at ~100 Megabits/second and
- The Internet backbone link speeds will run at ~10 Terabits/second

Moreover, there will be billions of interactive and embedded computing and communication devices in operation throughout the world. These powerful computers and networks will be available largely at commodity prices, built mostly with robust common-off-the-shelf (COTS) components, and will inter-operate over an increasingly convergent and pervasive Internet infrastructure.

To maximize the benefit from these advances in hardware technology, the quality and productivity of technologies for developing concurrent and networked middleware and application software must also increase. Historically, hardware has tended to become smaller, faster, and more reliable. It has also become cheaper and more predictable to develop and innovate, as evidenced by 'Moore's Law'. In contrast, concurrent and networked software has often grown larger, slower, and more error-prone. It has also become very expensive and time-consuming to develop, validate, maintain, and enhance.

Although hardware improvements have alleviated the need for some low-level software optimizations, the lifecycle cost [Boe81] and effort required to develop software—particularly mission-critical concurrent and networked applications—continues to rise. The disparity between the rapid rate of hardware advances versus the slower software progress stems from a number of factors, including:

- *Inherent and accidental complexities.* There are vexing problems with concurrent and networked software that result from inherent and accidental complexities. *Inherent* complexities arise from fundamental domain challenges, such as dealing with partial failures, distributed deadlock, and end-to-end quality of service (QoS) requirements. As networked systems have grown in scale and functionality they must now cope with a much broader and harder set of these complexities. *Accidental* complexities arise from limitations with software tools and development techniques, such as non-portable programming APIs and poor distributed debuggers. Ironically, many accidental complexities stem from deliberate choices made by developers who favor low-level languages and tools that scale up poorly when applied to complex concurrent and networked software.
- *Inadequate methods and techniques.* Popular software analysis methods [SM88] [CY91] [RBPEL91] and design techniques [Boo94] [BRJ98] have focused on constructing single-process, single-threaded applications with ‘best-effort’ QoS requirements. The development of high-quality concurrent and networked systems—particularly those with stringent QoS requirements, such as videoconferencing—has been left to the intuition and expertise of skilled software



architects and engineers. Moreover, it has been hard to gain experience with concurrent and networked software techniques without spending considerable time learning via trial and error, and wrestling with platform-specific details.

- *Continuous re-invention and re-discovery* of core concepts and techniques. The software industry has a long history of recreating incompatible solutions to problems that are already solved. For example, there are dozens of non-standard general-purpose and real-time operating systems that manage the same hardware resources. Similarly, there are dozens of incompatible operating system encapsulation libraries that provide slightly different APIs that implement essentially the same features and services.

If effort had instead been focused on enhancing and optimizing a small number of solutions, developers of concurrent and networked software would be reaping the benefits available to developers of hardware. These developers innovate rapidly by using and applying common CAD tools and standard instruction sets, buses, and network protocols.

No single silver bullet can slay all the demons plaguing concurrent and networked software [Broo87]. Over the past decade, however, it has become clear that *patterns* and *pattern languages* help to alleviate many inherent and accidental software complexities.

A pattern is a recurring solution schema to a standard problem in a particular context [POSA1]. Patterns help to capture and reuse the static and dynamic structure and collaboration of key participants in software designs. They are useful for documenting recurring microarchitectures, which are abstractions of software components that

experienced developers apply to resolve common design and implementation problems [GoF95].

When related patterns are woven together, they form a 'language' that helps to both

- Define a vocabulary for talking about software development problems [SFJ96] and
- Provide a process for the orderly resolution of these problems [Ale79] [AIS77]

By studying and applying patterns and pattern languages, developers can often escape traps and pitfalls that have been avoided traditionally only via long and costly apprenticeship [PLoPD1].

Until recently [Lea99a] patterns for developing concurrent and networked software existed only in programming folklore, the heads of expert researchers and developers, or were buried deep in complex source code. These locations are not ideal, for three reasons:

- Re-discovering patterns opportunistically from source code is expensive and time-consuming, because it is hard to separate the essential design decisions from the implementation details.
- If the insights and rationale of experienced designers are not documented, they will be lost over time and thus cannot help guide subsequent software maintenance and enhancement activities.
- Without guidance from earlier work, developers of concurrent and networked software face the Herculean task [SeSch70] of engineering complex systems from the ground up, rather than reusing proven solutions.