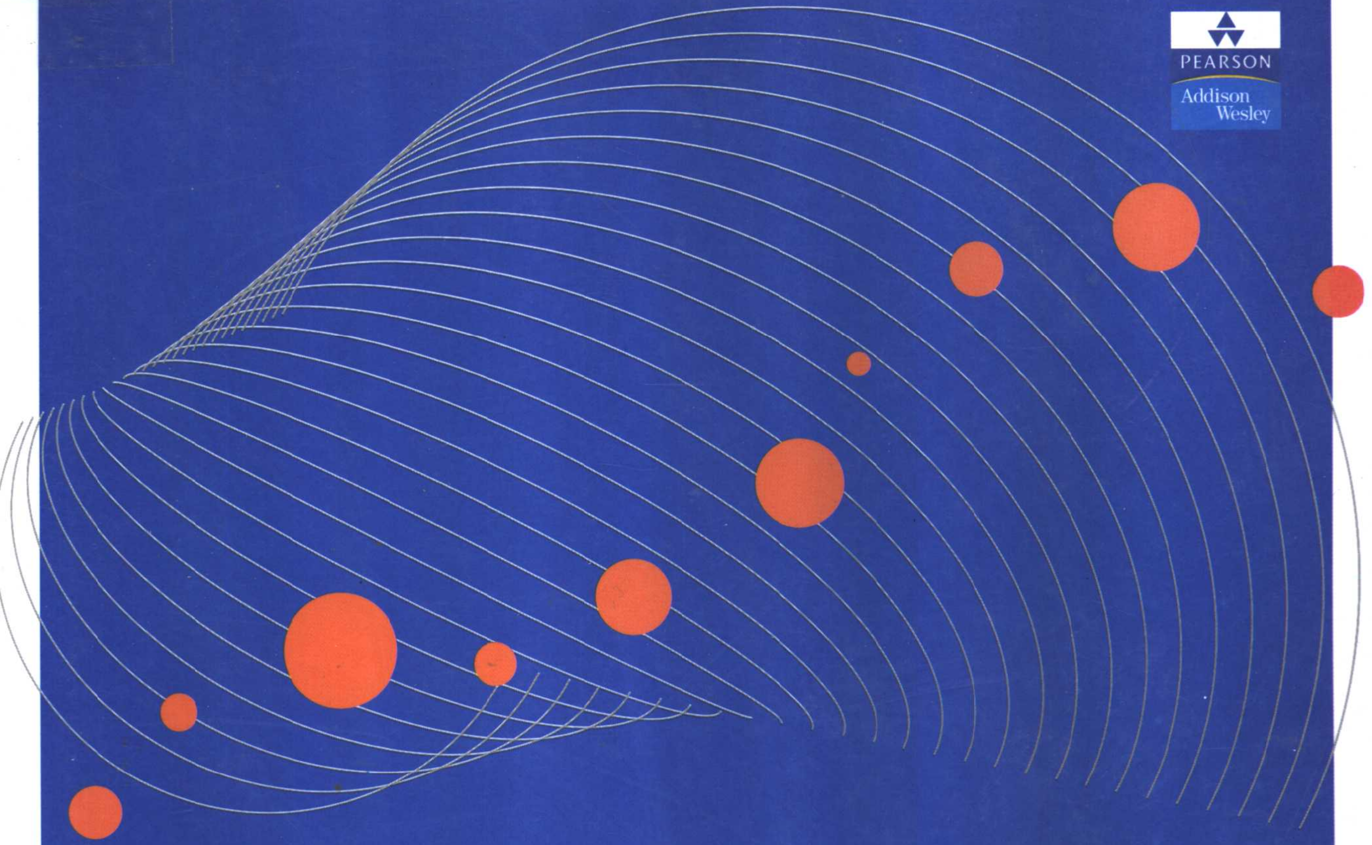
印版·

PEARSON

Addison Wesley



C++ Network Programming

Volume 1

Mastering Complexity with ACE and Patterns

Douglas C. Schmidt, Stephen D. Huston 著

C++网络编程 卷1

运用ACE和模式消除复杂性

Pearson Education

清华大学出版社

·影印版·

C++ Network Programming

Volume 1

Mastering Complexity with ACE and Patterns

Douglas C. Schmidt, Stephen D. Huston 著

C++网络编程 卷1

运用ACE和模式消除复杂性

清华大学出版社
北京

图书在版编目(CIP)数据

C++网络编程 卷 1：运用 ACE 和模式消除复杂性＝C++ Network Programming, Volume 1: Mastering Complexity with ACE and Patterns / 施米特（Schmidt, D. C.），休斯顿（Huston, S. D.）著. —影印本. —北京：清华大学出版社，2003.12
ISBN 7-302-07644-8

Ⅰ. C…　Ⅱ. ①施…　②休…　Ⅲ. C 语言－程序设计－高等学校－教材－英文　Ⅳ. TP312

中国版本图书馆 CIP 数据核字（2003）第 103806 号

出 版 者：	清华大学出版社	地　　址：	北京清华大学学研大厦
	http://www.tup.com.cn	邮　　编：	100084
	社总机：（010）6277 0175	客户服务：	（010）6277 6969

印 刷 者：北京四季青印刷厂
装 订 者：三河市兴旺装订有限公司
发 行 者：新华书店总店北京发行所
开　　本：185×230　印张：20.75
版　　次：2003 年 12 月第 1 版　2003 年 12 月第 1 次印刷
书　　号：ISBN 7-302-07644-8/TP・5606
印　　数：1~3000
定　　价：29.00 元

出 版 说 明

进入 21 世纪，世界各国的经济、科技以及综合国力的竞争将更加激烈。竞争的中心无疑是对人才的争夺。谁拥有大量高素质的人才，谁就能在竞争中取得优势。高等教育，作为培养高素质人才的事业，必然受到高度重视。目前我国高等教育的教材更新较慢，为了加快教材的更新频率，教育部正在大力促进我国高校采用国外原版教材。

清华大学出版社从 1996 年开始，与国外著名出版公司合作，影印出版了“大学计算机教育丛书（影印版）”等一系列引进图书，受到了国内读者的欢迎和支持。跨入 21 世纪，我们本着为我国高等教育教材建设服务的初衷，在已有的基础上，进一步扩大选题内容，改变图书开本尺寸，一如既往地请有关专家挑选适用于我国高校本科及研究生计算机教育的国外经典教材或著名教材，组成本套“大学计算机教育国外著名教材系列（影印版）”，以飨读者。深切期盼读者及时将使用本系列教材的效果和意见反馈给我们。更希望国内专家、教授积极向我们推荐国外计算机教育的优秀教材，以利我们把“大学计算机教育国外著名教材系列（影印版）”做得更好，更适合高校师生的需要。

清华大学出版社
2002 年 10 月

Foreword

As I write this foreword I'm traveling through Europe, relying on the excellent European public transportation infrastructure. Being an American, I'm fascinated and amazed by this infrastructure. Wherever I land at an airport I have easy access to trains and buses that are fast, clean, reliable, on time, and perhaps most importantly, going directly to my destination. Departure and arrival announcements are available in multiple languages. Signs and directions are easy to follow, even for non-native speakers like me.

I live and work in the Boston area, and like most Americans I rely almost entirely on my automobile to get from one place to the next. Except for an occasional use of the Boston subway system, I use my car to get around because the public transportation infrastructure is too limited to get me to my destination. Since millions of others in Boston and elsewhere are in the same predicament, our highway infrastructure is now well past the point of coping with the traffic volume. I know I'd be appalled if I knew exactly how much of my life I've wasted sitting in traffic jams.

There are some interesting similarities between networked computing systems and transportation systems, the most significant of these being that the success of both depends on scalable infrastructure. Scalable transportation systems comprise not just obvious infrastructure elements, such as trains and rails or airplanes and airports. They also require scheduling, routing, maintenance, ticketing, and monitoring, for example, all of which must scale along with the physical transportation system itself. Similarly, networked computing requires not only host machines and networks—the physical computing and communication infrastructure—

but also software-based scheduling, routing, dispatching, configuration, versioning, authentication, authorization, and monitoring that allows the networked system to scale as necessary.

An ironic fact about infrastructure is that it's extremely difficult to do well, and yet the more transparent to the user it is, the more successful we consider it to be. Despite the rugged terrain of the Swiss Alps, for example, a few architects, engineers, and builders have applied their expertise to provide an efficient transportation system that millions of people in Switzerland use daily with ease. In fact, the system is so reliable and easy to use that you quickly take it for granted, and it becomes transparent to you. For example, when boarding the Swiss railway your focus is simply on getting from one point to another, not on the machinery used to get you there. Unless you're a tourist, you probably miss the fact that you're traversing a tunnel that took years to design and build, or ascending an incline so steep that the railway includes a cog rail to help the train climb. The rail infrastructure does flawlessly what it's supposed to do, and as a result, you don't even notice it.

This book is about infrastructure software, normally called *middleware*, for networked computing systems. It's called middleware because it's the "waist in the hourglass" that resides above the operating system and networks, but underneath the application. Middleware comes in a wide variety of shapes, sizes, and capabilities, ranging from J2EE application servers, asynchronous messaging systems, and CORBA ORBs to software that monitors sockets for small embedded systems. Middleware must support an ever-wider variety of applications, operating systems, networking protocols, programming languages, and data formats. Without middleware, taming the ever-increasing diversity and heterogeneity in networked computing systems would be tedious, error prone, and expensive.

Despite the variety of types of middleware, and the variety of issues that middleware addresses, different types of middleware tend to use the same patterns and common abstractions to master complexity. If you were to peek inside a scalable and flexible application server, messaging system, or CORBA ORB, for example, you would likely find that they employ similar techniques for tasks such as connection management, concurrency, synchronization, event demultiplexing, event handler dispatching, error logging, and monitoring. Just as the users of the Swiss railways far outnumber those who designed and built it, the number of users of successful middleware far exceeds the number of people who designed and built it. If you design, build, or use middleware, your success depends on knowing, understanding, and applying these common patterns and abstractions.

While many understand the need for scalability and flexibility in middleware, few can provide it as effectively as the ADAPTIVE Communication Environment (ACE) that Doug Schmidt and Steve Huston describe in this book. ACE is a widely used C++ toolkit that captures common patterns and abstractions used in a variety of highly successful middleware and networked applications. ACE has become the basis for many networked computing systems, ranging from real-time avionics applications to CORBA ORBs to mainframe peer-to-peer communication support.

Like all good middleware, ACE hides the complexity of the diverse and heterogeneous environments beneath it. What sets ACE apart from most other infrastructure middleware, however, is that even though it allows for maximum flexibility wherever needed by the application, it doesn't degrade the performance or scalability of the system. Being a long-time middleware architect myself, I know all too well that achieving both performance and flexibility in the same package is hard.

In a way, though, the flexibility and performance aspects of ACE don't surprise me. Due to my long-time association with Doug, I'm well aware that he is a pioneer in this area. The wide variety of scalable, high-performing, and flexible middleware that exists today clearly bears his mark and influence. His teaming with Steve, who's a gifted C++ developer and author whose work on ACE has led to many improvements over the years, has yielded a work that's a "must read" for anyone involved in designing, building, or even using middleware. The increasing pervasiveness of the World Wide Web and of interconnected embedded systems means that the number, scale, and importance of networked computing systems will continue to grow. It's only through understanding the key patterns, techniques, classes, and lessons that Doug and Steve describe in this book that we can hope to supply the middleware infrastructure to make it all transparent, efficient, and reliable.

Steve Vinoski
Chief Architect & Vice President, Platform Technologies
IONA Technologies
September 2001

About This Book

Over the past decade, concurrent object-oriented network programming has emerged as an effective paradigm for developing software applications whose collaborating objects can either be

1. Collocated within one process or computer or
2. Distributed across a set of computers connected by a network, such as an embedded system interconnect, a local area network (LAN), an enterprise intranet, or the Internet.

When objects are distributed, the various entities that constitute these objects must communicate and coordinate with each other effectively. Moreover, they must continue to do so as applications change over their lifetimes. The placement of objects, the available networking infrastructure, and platform concurrency options allow for a level of freedom that's powerful, yet challenging.

When designed properly, concurrent object-oriented network programming capabilities can add a great deal of flexibility to your application options. For instance, in accordance with the requirements and resources available to your projects, you can use

- Real-time, embedded, or handheld systems
- Personal or laptop computers
- An assortment of various-sized UNIX or Linux systems
- "Big iron" mainframes and even supercomputers

You'll likely encounter complex challenges, however, when developing and porting networked applications on multiple operating system (OS) platforms. These complexities appear in the form of incompatible networking protocols or component libraries that have different APIs and semantics on different hardware and software *platforms*, as well as *accidental complexities* introduced by limitations with the native OS interprocess communication (IPC) and concurrency mechanisms themselves. To alleviate these problems, the *ADAPTIVE Communication Environment* (ACE) provides an object-oriented toolkit that runs portably on dozens of hardware and OS platforms, including most versions of Win32 and UNIX, as well as many real-time and embedded operating systems.

Some would have you believe that *de facto* or *de jure* OS standards, such as POSIX, UNIX98, or Win32, are all programmers need to shield their applications from portability challenges. Unfortunately, the adage that "the nice thing about standards is that there are so many to choose from" [Tan96] is even more applicable today than it was a decade ago. There are now dozens of different OS platforms used in commercial, academic, and governmental projects, and the number of permutations grows with each new version and variant.

We've developed many multiplatform, concurrent, and networked systems for the past two decades. We can therefore assure you that OS vendors often choose to implement different standards at different times. Moreover, standards change and evolve. It's likely that you'll work on multiple platforms that implement different standards in different ways at different times. Programming directly to OS APIs therefore yields the following two problems:

1. **It's error-prone** since native OS APIs written in C often lack typesafe, portable, reentrant, and extensible system function interfaces and function libraries. For example, endpoints of communication in the widely used Sockets API (discussed in Chapter 2) are identified via weakly typed integer or pointer I/O handles, which increase the likelihood of subtle programming errors at run-time.
2. **It encourages inadequate design techniques** since many networked applications written using OS APIs are based upon algorithmic design, rather than object-oriented design. Algorithmic design decomposes the structure of an application according to specific functional requirements, which are volatile and likely to evolve over time. This design paradigm therefore yields nonextensible software architectures that can't be customized rapidly to meet changing application requirements [Boo94].

In this age of economic upheaval, deregulation, and stiff global competition, it's becoming prohibitively expensive and time consuming to develop applications entirely from scratch using native OS APIs and algorithmic design techniques.

If you've been developing networked software systems for many years, you may have learned to accept some of these problems as a fact of life. There is a better way, however. In this book, we show how C++ and ACE provide object-oriented capabilities that allow you to avoid many traps and pitfalls, while still leveraging standards—and even certain platform-specific features—whenever possible. Object-oriented designs exhibit greater stability over time than algorithmic designs, which makes them the preferred basis for developing many types of networked applications.

Not surprisingly, there's a price for all this flexibility: you may need to learn some new concepts, methods, patterns, tools, and development techniques. Depending on your background, this learning curve may be trivial or it may initially seem steep. The bottom line, however, is that the object-oriented paradigm can offer you a mature set of techniques that alleviates many challenges of networked application development. This book presents a series of concrete examples to illustrate the object-oriented techniques used to develop and apply the classes in the ACE toolkit. You can use the same techniques and ACE classes to simplify your own applications.

Intended Audience

This book is intended for "hands-on" developers or advanced students interested in understanding the strategies and tactics of concurrent network programming using C++ and object-oriented design. We describe the key design dimensions, patterns, and principles needed to develop flexible and efficient concurrent networked applications quickly and easily. Our numerous C++ code examples reinforce the design concepts and illustrate concretely how to use the core classes in ACE right away. We also take you "behind the scenes" to understand how and why the IPC and concurrency mechanisms in the ACE toolkit are designed the way they are. This material will help to enhance your design skills and to apply C++ and *patterns* more effectively in your own object-oriented networked applications.

This book is not a comprehensive tutorial on object-oriented development, patterns, UML, C++, systems programming, or networking. We therefore assume readers of this book have some familiarity with the following topics:

- **Object-oriented design and programming techniques,** for example, frameworks [Joh97, FJS99b, FJS99a], patterns [GHJV95, BMR+96, SSRB00], modularity [Mey97], information hiding [Par72], and modeling [Boo94]
- **Object-oriented notations and processes,** such as the Unified Modeling Language (UML) [RJB98], eXtreme Programming [Bec00], and the Rational Unified Process (RUP) [JBR99]
- **Fundamental C++ language features,** such as classes, *inheritance*, *dynamic binding*, and *parameterized types* [Bja00]
- **Core systems programming mechanisms,** such as event demultiplexing, process and thread management, virtual memory, and IPC mechanisms and APIs commonly available on UNIX [Ste98, Ste99, Ste92, Lew95, KSS96, But97] and Win32 [Ric97, Sol98, JO99] platforms
- **Networking terminology and concepts,** such as TCP/IP [Ste93], remote operation invocations [Obj01], and client/server architectures [CS92]

We encourage you to use the extensive bibliography to locate sources of information on topics about which you want to learn more.

This book is also not an ACE programmer's manual; that is, we don't explain every method of every class in ACE. For that level of detail we refer you to the extensive online ACE documentation, generated by Doxygen [Dim01], at `http://ace.ece.uci.edu/Doxygen/` and `http://www.riverace.com/docs/`. Instead, this book focuses on

- The key concepts, patterns, and C++ features that shape the design of successful object-oriented networked applications and middleware and
- The motivation behind, and basic usage of, the most commonly used ACE TCP/IP and concurrency wrapper facade classes

Structure and Content

This book describes how C++ and middleware help address key challenges associated with developing networked applications. We review the core native OS mechanisms available on popular OS platforms and illustrate how C++ and patterns are applied in ACE to encapsulate these mechanisms in class library wrapper facades that improve application portability and robustness. The book's primary application example is a networked logging service that transfers log records from client applications to a logging server over TCP/IP. We use this service as a running example throughout the book to

- Show concretely how C++ and ACE can help achieve efficient, predictable, and scalable networked applications and
- Demonstrate key design and implementation considerations and solutions that will arise when you develop your own concurrent object-oriented networked applications

The book is organized into 11 chapters as follows:

- **Introduction**—Chapter 0 presents an introduction to C++ network programming. It starts by outlining the problem space and presenting the challenges that can arise when applications extend beyond a single thread in a single process. We then introduce a taxonomy of middleware layers and describe how host infrastructure middleware and the ACE toolkit can be applied to address common network programming challenges.
- **Part I**—Chapters 1 through 4 outline communication design alternatives and describe the object-oriented techniques used in ACE to program OS IPC mechanisms effectively. The resulting classes form the basis of the first version of the book's running example, a networked logging service.
- **Part II**—Chapters 5 through 10 outline concurrency design alternatives and describe the object-oriented techniques used in ACE to program OS concurrency mechanisms effectively.

Throughout Parts I and II we present a series of increasingly sophisticated implementations of our networked logging service to illustrate how the ACE IPC and concurrency wrapper facades can be applied in practice.

Appendix A summarizes the class design and implementation principles that underlie the ACE IPC and concurrency wrapper facades. Appendix B explains the inception and open-source evolution of ACE over the past decade and outlines where it's heading in the future. The book concludes with a glossary of technical terms (including the *italicized* terms in this book), an extensive list of references for further research, and a general subject index.

Related Material

This book focuses on resolving complexity using specific C++ features, patterns, and ACE. The second volume in this series—*C++ Network Programming: Systematic Reuse with ACE and Frameworks* [SH]—extends our coverage to include object-oriented network programming frameworks provided by ACE. These frameworks reify common usage patterns of the ACE wrapper facade classes presented in this book to support broader, more extensible levels of systematic reuse. A distinguishing factor between the ACE wrapper facade classes covered in this book and the ACE framework classes covered in Volume 2 is that the ACE wrapper facade classes have few virtual methods, whereas the ACE framework classes have mostly virtual methods.

This book is based on ACE version 5.2, released in October 2001. The ACE software and all the sample applications described in our books are open-source and can be downloaded at `http://ace.ece.uci.edu` and `http://www.riverace.com`. These sites also contain a wealth of other material on ACE, such as tutorials, technical papers, and an overview of other ACE wrapper facades for IPC and synchronization mechanisms that aren't covered in this book. We encourage you to obtain a copy of ACE so you can follow along, see the actual ACE classes and frameworks in complete detail, and run the code examples interactively as you read through the book. Precompiled versions of ACE can also be purchased at a nominal cost from `http://www.riverace.com`.

To learn more about ACE, or to report any errors you find in the book, we recommend you subscribe to the ACE mailing list, `ace-users@cs.wustl.edu`. You can subscribe by sending e-mail to the Majordomo list server at `ace-users-request@cs.wustl.edu`. Include the following command in the body of the e-mail (the subject line is ignored):

```
subscribe ace-users [emailaddress@domain]
```

You must supply `emailaddress@domain` only if your message's `From` address is not the address you wish to subscribe.

Postings to the ACE mailing list are also forwarded to the USENET newsgroup `comp.soft-sys.ace`. Archives of postings to the ACE mailing list are available at `http://groups.yahoo.com/group/ace-users`.

Acknowledgments

Champion reviewing honors go to Christopher Allen, Tomer Amiaz, Alain Decamps, Don Hinton, Susan Liebeskind, Dennis Mancl, Patrick Rabau, Eamonn Saunders, and Johnny Willemsen, who reviewed the entire book and provided extensive comments that improved its form and content substantially. Naturally, we are responsible for any remaining problems.

Many other ACE users from around the world provided feedback on drafts of this book, including Mark Appel, Shahzad Aslam-Mir, Kevin Bailey, Barry Benowitz, Emmanuel Croze, Yasir Faiz, Gillmer Derge, Iain Hanson, Brad Hoskins, Bob Huston, Christopher Kohlhoff, Serge Kolgan, Andy Marchewka, Jeff McNiel, Phil Mesnier, Arturo Montes, Aaron Nielsen, Jeff Parsons, Pim Philipse, Yaron Pinto, Stephane Pion, Nick Pratt, Paul Rubel, Shourya Sarcar, Leo Stutzmann, Tommy Svensson, Alain Totouom, Roger Tragin, and Reuven Yagel.

We are indebted to all the members, past and present, of the DOC groups at Washington University, St. Louis and the University of California, Irvine, and the team members at Object Computing Inc. and Riverace Corporation, who developed, refined, and optimized many of the ACE capabilities presented in this book. This group includes Everett Anderson, Alex Arulanthu, Shawn Atkins, John Aughey, Darrell Brunsch, Luther Baker, Don Busch, Chris Cleeland, Angelo Corsaro, Chad Elliot, Sergio Flores-Gaitan, Chris Gill, Pradeep Gore, Andy Gokhale, Priyanka Gontla, Myrna Harbibson, Tim Harrison, Shawn Hannan, John Heitmann, Joe Hoffert, James Hu, Frank Hunleth, Prashant Jain, Vishal Kachroo, Ray Klefstad, Kitty Krishnakumar, Yamuna Krishnamurthy, Michael Kircher, Fred Kuhns, David Levine, Chanaka Liyanaarachchi, Michael Moran, Ebrahim Moshiri, Sumedh Mungee, Bala Natarajan, Ossama Othman, Jeff Parsons, Kirthika Parameswaran, Krish Pathayapura, Irfan Pyarali, Sumita Rao, Carlos O'Ryan, Rich Siebel, Malcolm Spence, Marina Spivak, Naga Surendran, Steve Totten, Bruce Trask, Nanbor Wang, and Seth Widoff.

We also want to thank the thousands of C++ developers from over fifty countries who've contributed to ACE during the past decade. ACE's excellence and success is a testament to the skills and generosity of many talented developers and the forward-looking companies that have had the vision to contribute their work to ACE's open-source code base. Without their support, constant feedback, and encouragement, we never would have written this book. In recognition of the efforts of the ACE open-source community, we maintain a list of all contributors, which is available at `http://ace.ece.uci.edu/ACE-members.html`.

We are also grateful for the support from colleagues and sponsors of our research on patterns and development of the ACE toolkit, notably the contributions of Ron Akers (Motorola), Steve Bachinsky (SAIC), John Bay (DARPA), Detlef Becker (Siemens), Dave Busigo (DARPA), John Buttitto (Sun), Becky Callison (Boeing), Wei Chiang (Nokia), Joe Cross (Lockheed Martin), Lou DiPalma (Raytheon), Bryan Doerr (Boeing), Karlheinz Dorn (Siemens), Matt Emerson (Escient Convergence Group, Inc.), Sylvester Fernandez (Lockheed Martin), Nikki Ford (DARPA), Andreas Geisler (Siemens), Helen Gill (NSF), Bob Groschadl (Pivotech Systems, Inc.), Jody Hagins (ATD), Andy Harvey (Cisco), Sue Kelly (Sandia National Labs), Gary Koob (DARPA), Petri Koskelainen (Nokia Inc), Sean Landis (Motorola), Patrick Lardieri (Lockheed Martin), Doug Lea (SUNY Oswego), Hikyu Lee (SoftLinx), Joe Loyall (BBN), Mike Masters (NSWC), Ed Mays (U.S. Marine Corps), John Mellby (Raytheon), Jeanette Milos (DARPA), Stan Moyer (Telcordia), Russ Noseworthy (Object Sciences), Dieter Quehl (Siemens), Vijay Raghavan (Vanderbilt U.), Lucie Robillard (U.S. Air Force), Craig Rodrigues (BBN), Rick Schantz (BBN), Steve Shaffer (Kodak), Tom Shields (Raytheon), Dave Sharp (Boeing), Naval Sodha (Ericsson), Paul Stephenson (Ericsson), Tatsuya Suda (UCI), Umar Syyid (Hughes), Janos Sztipanovits (Vanderbilt U.), Gautam Thaker (Lockheed Martin), Lothar Werzinger (Krones), and Don Winter (Boeing).

Very special thanks go to Susan Cooper, our copy editor, for enhancing our written material. In addition, we are grateful for the encouragement and patience of our editor, Debbie Lafferty, our production coordinator, Elizabeth Ryan, the series editor and inventor of C++, Bjarne Stroustrup, and everyone else at Addison-Wesley who made it possible to publish this book.

Finally, we would also like to express our gratitude and indebtedness to the late W. Richard Stevens, the father of network programming literature.

His books brought a previously unknown level of clarity to the art and science of network programming. We endeavor to stand on his virtual shoulders, and extend the understanding that Richard's books brought into the world of object-oriented design and C++ programming.

Steve's Acknowledgments

I would like to thank God, who gave me an enjoyment of computers and networking. I hope He's pleased. To Jane, my wife of 20 years, thank you for loving me and cheering me on every day. I would not have completed this work without your support—you are a blessing. Thank you to the late David N. Drummond, who took a chance on a kid without a degree. And thank you to Doug Schmidt, a scholar and a gentleman, whose insight, enthusiasm, and creativity impress and challenge me daily.

Doug's Acknowledgments

I've been writing this book for over a decade, so it's an enormous thrill (and relief) to see it in print at last! For this, I'm grateful to Steve Huston, Debbie Lafferty, and Bjarne Stroustrup for their immense help and patience in seeing this project through to fruition. I'd also like to thank my wife Sonja for her love and support during the writing of this book—now that it's finished we'll have more time for ballroom dancing! Finally, thanks to my many friends and colleagues at the College of William and Mary; Washington University, St. Louis; University of California, Irvine; DARPA; and Siemens—as well as the thousands of ACE and TAO developers and users worldwide—who have greatly enriched my intellectual and interpersonal life over the past two decades.

Contents

List of Figures		**ix**
Foreword		**xi**
About This Book		**xv**
Chapter 0	**Design Challenges, Middleware Solutions, and ACE**	**1**
0.1	Challenges of Networked Applications	1
0.2	Networked Application Design Dimensions	5
0.3	Object-Oriented Middleware Solutions	7
0.4	An Overview of the ACE Toolkit	12
0.5	Example: A Networked Logging Service	17
0.6	Summary	19
Part I	**Object-Oriented Network Programming**	**21**
Chapter 1	**Communication Design Dimensions**	**23**
1.1	Connectionless versus Connection-Oriented Protocols	23
1.2	Synchronous versus Asynchronous Message Exchange	26
1.3	Message Passing versus Shared Memory	28
1.4	Summary	31