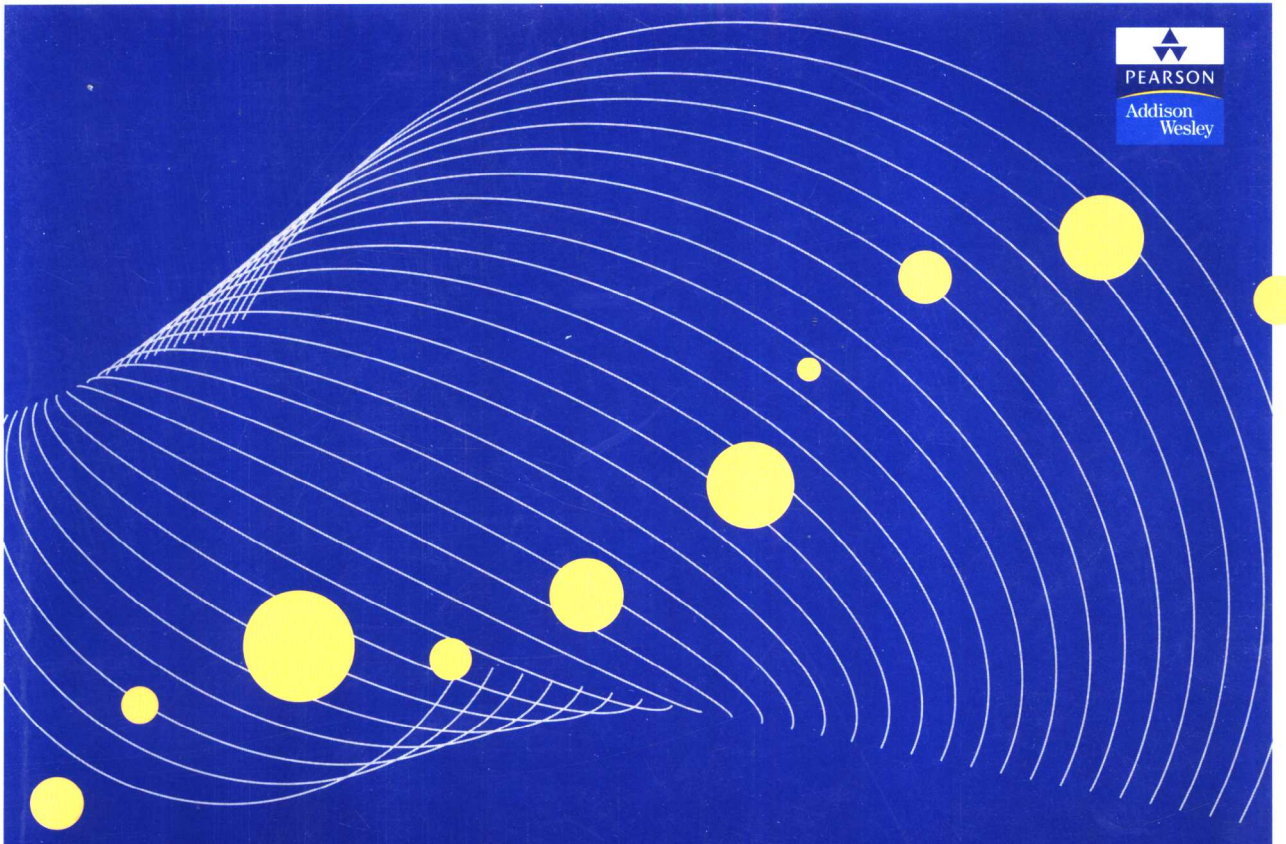
·影印版·



PEARSON
Addison Wesley

C++ Network Programming

Volume 2

Systematic Reuse with ACE and Frameworks

Douglas C. Schmidt, Stephen D. Huston 著

C++网络编程 卷2

基于ACE和框架的系统化复用

清华大学出版社

·影印版·

C++ Network Programming

Volume 2

Systematic Reuse with ACE and Frameworks

Douglas C. Schmidt, Stephen D. Huston 著

C++网络编程 卷2

基于ACE和框架的系统化复用

清华大学出版社
北京0270

北京市版权局著作权合同登记号　图字 01-2003-8795

本书封面贴有 Pearson Education(培生教育出版集团)激光防伪标签，无标签者不得销售。

图书在版编目(CIP)数据

C++网络编程 卷 2：基于 ACE 和框架的系统化复用 = C++ Network Programming, Volume 2: Systematic Reuse with ACE and Frameworks / 施密特（Schmidt, D. C.），休斯顿（Huston, S. D.）著. —影印本. —北京：清华大学出版社，2004.2
ISBN 7-302-07964-1

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（2004）第 003408 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：北京牛山世兴印刷厂
装 订 者：北京市密云县京文制本装订厂
发 行 者：新华书店总店北京发行所
开　　本：185×230　印张：23.25
版　　次：2004 年 2 月第 1 版　2004 年 2 月第 1 次印刷
书　　号：ISBN 7-302-07964-1/TP • 5784
印　　数：1~3000
定　　价：34.00 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

Foreword

The ADAPTIVE Communication Environment (ACE) toolkit has achieved enormous success in the area of middleware for networked computing. Due to its flexibility, performance, platform coverage, and other key properties, ACE enjoys broad acceptance by the networked application software community, as evidenced by its use in thousands of applications, in scores of countries, and in dozens of domains. ACE has also received considerable attention beyond the middleware community since it's an open-source role model for high-quality and well-designed pattern-oriented software architectures.

But why is ACE so successful? Addressing this question properly takes some thought. To start off, let's reconsider the Foreword from *C++ Network Programming: Mastering Complexity with ACE and Patterns* (C++NPv1) and resume the mass transit analogy presented there by my colleague Steve Vinoski. Steve's right that a high-quality mass transit system consists of more than just aircraft, airports, trains, train stations, and rails. It also needs less obvious infrastructure, such as scheduling, routing, ticketing, maintenance, and monitoring. But even a complete collection of ingredients is still not sufficient to develop an effective mass transit system. Arranging these ingredients so they seamlessly fulfill their primary objective—fast and reliable transportation of people—is equally important. Would you use a mass transit system whose ticketing was located in a train maintenance location or an airport hangar, or whose planned and actual scheduling and routing weren't available to the public? I doubt it!

The success of mass transit systems depends on more than the knowledge of the infrastructure parts that are provided—it depends on how these different parts must be connected and integrated with their environment. This knowledge enables architects of mass transit systems to integrate individual parts into higher-level building blocks and to connect these building blocks effectively. For example, ticketing, information points, baggage offices, and boarding are integrated in train stations located at city centers or major suburban centers. Likewise, airports are often located near large cities and connected by frequent express trains.

Even mass transit centers themselves are arranged so that activities can be performed effectively. For example, when you enter a train station or airport via the main entrance, you find ticket agents, information centers, and timetables. You also find shops to satisfy your travel needs. As you enter the main train hall or airport concourse, you find other information centers, up-to-date scheduling information, and the platforms and gates for boarding the trains and planes. Mass transit centers thus not only provide all necessary services to begin and end a journey, they also organize their internal "control flows" effectively. While the core structures and control flows in most train stations and airports are similar, their concrete realization can differ widely. Yet we all recognize these mass transit center patterns immediately since they follow key invariants that we've learned through years of experience.

So what's the connection between successful mass transit system design and the success of ACE? The answer is simple: In addition to the basic network computing ingredients (the wrapper facades that Doug and Steve introduced in C++NPv1), ACE also includes useful object-oriented frameworks that build upon these wrapper facades and provide useful higher-level communication services, such as event demultiplexing and dispatching, connection management, service configuration, concurrency, and hierarchically layered stream processing. The ACE framework services satisfy many networked software needs by organizing the structures and internal control flows of your applications effectively via key patterns learned through years of experience.

The ACE frameworks offer you a number of important benefits:

- You needn't develop the capabilities provided by ACE, which will save considerable time and effort. You can therefore focus on your key responsibility: *implementing the application functionality required by your customers and end users.*
- The ACE frameworks reify the extensive network programming expertise that Doug, Steve, and their colleagues have gained over several decades. In particular, the ACE frameworks efficiently implement the canonical classes, class relationships, and control flows common to networked applications. The ACE frameworks are tested regularly by thousands of users from around the world, which has yielded many useful corrections and improvements. As an ACE user, you can directly leverage the correctness, effectiveness, and efficiency of the ACE frameworks in your applications.
- A framework isn't a framework if it can't be adapted to specific user needs. This means you can adapt the ACE frameworks at key points of variation in networked applications. For example, the ACE Reactor framework can be adapted to use different event demultiplexer functions, such as `WaitForMultipleObjects()` or `select()`. Likewise, the ACE Acceptor-Connector framework can be configured with different IPC mechanisms. While this adaptability is beneficial by itself, ACE goes a step further: for many adaptations you can configure the desired strategies from available and interchangeable implementations. In addition to the different Re-

actor implementations mentioned above, for instance, ACE provides wrapper facades for various IPC mechanisms, such as the Sockets, SSL, TLI, and shared memory, that help to configure the ACE Acceptor-Connector framework for specific platforms and applications.

- Last but not least, the ACE frameworks don't exist in isolation. You can therefore combine them in novel ways to create networked applications and entirely new types of middleware. For example, you can integrate the Reactor framework with the Acceptor-Connector framework to separate connection establishment from service processing functionality in event-driven applications. You can likewise introduce various forms of concurrency into your applications using the ACE Task framework.

As a result of advising and leading many software projects over the years, I've found that ACE greatly simplifies the task of employing reusable middleware that can be customized readily to meet the needs of networked applications. Not all networked applications need heavyweight middleware, such as application servers, web services, and complex component models. Yet most networked applications can benefit from portable and efficient host infrastructure middleware like ACE. This flexibility is the core of ACE's success since you needn't commit to an entire middleware suite if you don't use all of it. Instead, you can combine just the essential ACE middleware classes you need to compose applications that are small, but as powerful as necessary. For this reason, I predict that ACE will still be widely used long after the influence of today's heavyweight middleware has waned.

ACE's tremendous flexibility also needn't lead to a sea of incompatible middleware implementations. For example, if you build an embedded system that speaks the CORBA Internet inter-ORB protocol (IIOP) to the outside world, you can use The ACE ORB (TAO), which is a CORBA-compliant, open-source, real-time object request broker (ORB) built using the ACE wrapper facades and frameworks. If CORBA is overkill for your application needs, however, you can build custom, yet interoperable, middleware using the appropriate ACE classes. Both solutions can be based on the same core structures and protocols, such as the ACE Common Data Representation (CDR) classes and its TCP/IP Socket wrapper facades. They can therefore communicate seamlessly with one another, just as you can take a train from Paris to Istanbul—the famous Orient Express—and travel through many European countries without having to change trains due to incompatible railroad networks.

As Steve Vinoski and I have pointed out, there are many similarities between high-quality mass transit systems and high-quality networking middleware. To me and thousands of other C++ developers around the world, ACE is *the* toolkit for building the latter! After saying so many good things about ACE, however, let's return to the main intent of this foreword: introducing the second volume (C++NPv2) of the *C++ Network Programming* series. As with all software technologies and middleware, the more you understand your tools, the better you'll be able to apply them. It turns out that using ACE in your applications is just one aspect of improving your networked software. To benefit significantly

from ACE's many advantages, you therefore also need a sound understanding of the core concepts, patterns, and usage rules that underlie its powerful frameworks.

For years, a common way to learn ACE involved studying its code, comments, and example applications. Clearly, this process was time consuming and error prone. Moreover, even after managing to read the several hundred thousand lines of C++ code in ACE, it was easy to miss the forest for the trees. As the Greek philosopher Thucydides noted two millennia ago: "A man who has the knowledge but lacks the power to clearly express himself is no better off than if he had never any idea at all."

We're therefore fortunate that Doug and Steve found time in their busy schedules to create such a high-quality book on the ACE frameworks. C++NPv2 explains the ideas and concepts underlying the ACE frameworks in an easily accessible form using the popular concurrency and networking patterns from the POSA [POSA1, POSA2] and "Gang of Four" [GoF] patterns books. These patterns, in turn, reify thoughtful and time-proven solutions to common networking problems. For example, they tell you what the problems are, why these problems are hard, what the solutions to these problems are, and why these solutions applied to ACE are of high quality. If you want thorough coverage of the patterns and frameworks in ACE that are shaping the next generation of networked application software then read this book. I've learned much from it and I'm sure you will too.

Frank Buschmann
Senior Principal Engineer
Siemens Corporate Technology
Munich, Germany

About This Book

Software for networked applications must possess the following qualities to be successful in today's competitive, fast-paced computing industry:

- **Affordability,** to ensure that the total ownership costs of software acquisition and evolution are not prohibitively high
- **Extensibility,** to support successions of quick updates and additions to address new requirements and take advantage of emerging markets
- **Flexibility,** to support a growing range of multimedia data types, traffic patterns, and end-to-end quality of service (QoS) requirements
- **Portability,** to reduce the effort required to support applications on heterogeneous OS platforms and compilers
- **Predictability** and **efficiency,** to provide low latency to delay-sensitive real-time applications, high performance to bandwidth-intensive applications, and usability over low-bandwidth networks, such as wireless links
- **Reliability,** to ensure that applications are robust, fault tolerant, and highly available
- **Scalability,** to enable applications to handle large numbers of clients simultaneously

Writing high-quality networked applications that exhibit these qualities is hard—it's expensive, complicated, and error prone. The patterns, C++ language features, and object-oriented design principles presented in *C++ Network Programming, Volume 1: Mastering Complexity with ACE and Patterns* (C++NPv1) help to minimize complexity and mistakes in networked applications by refactoring common structure and functionality into reusable *wrapper facade* class libraries. The key benefits of reuse will be lost, however, if large parts of the application software that uses these class libraries—or worse, the class libraries themselves—must be rewritten for each new project.

Historically, many networked application software projects began by

1. Designing and implementing demultiplexing and dispatching infrastructure mechanisms that handle timed events and I/O on multiple socket handles
2. Adding service instantiation and processing mechanisms atop the demultiplexing and dispatching layer, along with message buffering and queueing mechanisms
3. Implementing large amounts of application-specific code using this *ad hoc* host infrastructure middleware

This development process has been applied many times in many companies, by many projects in parallel. Even worse, it's been applied by the same teams in a series of projects. Regrettably, this continuous rediscovery and reinvention of core concepts and code has kept costs unnecessarily high throughout the software development life cycle. This problem is exacerbated by the inherent diversity of today's hardware, operating systems, compilers, and communication platforms, which keep shifting the foundations of networked application software development.

Object-oriented frameworks [FJS99b, FJS99a] are one of the most flexible and powerful techniques that address the problems outlined above. A framework is a reusable, "semi-complete" application that can be specialized to produce custom applications [JF88]. Frameworks help to reduce the cost and improve the quality of networked applications by reifying proven software designs and patterns into concrete source code. By emphasizing the integration and collaboration of application-specific and application-independent classes, frameworks enable larger scale reuse of software than can be achieved by reusing individual classes or stand-alone functions.

In the early 1990s, Doug Schmidt started the open-source ACE project to bring the power and efficiency of patterns and frameworks to networked application development. As with much of Doug's work, ACE addressed many real-world problems faced by professional software developers. Over the following decade, his groups at the University of California, Irvine; Washington University, St. Louis; and Vanderbilt University, along with contributions from the ACE user community and Steve Huston at Riverace, yielded a C++ toolkit containing some of the most powerful and widely used concurrent object-oriented network programming frameworks in the world. By applying reusable software patterns and a lightweight OS portability layer, the frameworks in the ACE toolkit provide synchronous and asynchronous event processing; concurrency and synchronization; connection management; and service configuration, initialization, and hierarchical integration.

The success of ACE has fundamentally altered the way that networked applications and middleware are designed and implemented on the many operating systems outlined in Sidebar 2 (page 16). ACE is being used by thousands of development teams, ranging from large Fortune 500 companies to small startups to advanced research projects at universities and industry labs. Its open-source development model and self-supporting culture is similar in spirit and enthusiasm to that driving Linus Torvalds's popular Linux operating system.

This book describes how the ACE frameworks are designed and how they can help developers navigate between the limitations of

1. **Low-level native operating system APIs,** which are inflexible and nonportable
2. **High-level middleware,** such as *distribution middleware* and *common middleware services*, which often lacks the efficiency and flexibility to support networked applications with stringent QoS and portability requirements

The skills required to produce and use networked application frameworks have traditionally been locked in the heads of expert developers or buried deep within the source code of numerous projects that are spread throughout an enterprise or an industry. Neither of these locations is ideal, of course, since it's time consuming and error prone to reengineer this knowledge for each new application or project. To address this problem, this book illustrates the key patterns [POSA2, POSA1, GoF] that underlie the structure and functionality of the ACE frameworks. Our coverage of these patterns also makes it easier to understand the design, implementation, and effective use of the open-source ACE toolkit itself.

Intended Audience

This book is intended for "hands on" C++ developers or advanced students interested in understanding how to design object-oriented frameworks and apply them to develop networked applications. It builds upon material from C++NPv1 that shows how developers can apply patterns to master complexities arising from using native OS APIs to program networked applications. It's therefore important to have a solid grasp of the following topics covered in C++NPv1 before reading this book:

- **Networked application design dimensions,** including the alternative communication protocols and data transfer mechanisms discussed in Chapter 1 of C++NPv1
- **Internet programming mechanisms,** such as TCP/IP connection management and data transfer APIs [Ste98] discussed in Chapter 2 of C++NPv1
- **Concurrency design dimensions,** including the use of processes and threads, iterative versus concurrent versus reactive servers, and threading models [Ste99] discussed in Chapters 5 through 9 of C++NPv1
- **Synchronization techniques** necessary to coordinate the interactions of processes and threads on various OS platforms [KSS96, Lew95, Ric97] discussed in Chapter 10 of C++NPv1
- **Object-oriented design and programming techniques** [Boo94, Mey97] that can simplify OS APIs and avoid programming mistakes through the use of patterns, such as Wrapper Facade [POSA2] and Proxy [POSA1, GoF] discussed in Chapter 3 and Appendix A of C++NPv1

The ACE frameworks are highly flexible and powerful, due in large part to their use of C++ language features [Bja00]. You should therefore be familiar with C++ class inheritance and virtual functions (dynamic binding) as well as templates (parameterized types) and the mechanisms your compiler(s) offer to instantiate them. ACE provides a great deal of assistance in overcoming differences between C++ compilers. As always, however, you need to know the capabilities of your development tools and how to use them. Knowing your tools makes it easier to follow the source code examples in this book and to build and run them on your systems. Finally, as you read the examples in this book, keep in mind the points noted in Sidebar 7 (page 46) regarding UML diagrams and C++ code.

Structure and Content

Our C++NPv1 book addressed how to master certain complexities of developing networked applications, focusing on the use of ACE's wrapper facades to avoid problems with operating system APIs written in C. This book (which we call C++NPv2) elevates our focus to motivate and demystify the patterns, design techniques, and C++ features associated with developing and using the ACE frameworks. These frameworks help reduce the cost and improve the quality of networked applications by reifying proven software designs and patterns into frameworks that can be reused systematically across projects and enterprises. The ACE frameworks expand reuse technology far beyond what can be achieved by reusing individual classes or even class libraries.

This book presents numerous C++ applications to reinforce the design discussions by showing concrete examples of how to use the ACE frameworks. These examples provide step-by-step guidance that can help you apply key object-oriented techniques and patterns to your own networked applications. The book also shows how to enhance your design skills, focusing on the key concepts and principles that shape the design of successful object-oriented frameworks for networked applications and middleware.

The chapters in the book are organized as follows:

- Chapter 1 introduces the concept of an object-oriented framework and shows how frameworks differ from other reuse techniques, such as class libraries, components, patterns, and model-integrated computing. We then outline the frameworks in the ACE toolkit that are covered in subsequent chapters.

- Chapter 2 completes the domain analysis begun in C++NPv1, which covered the communication protocols and mechanisms, and the concurrency architectures used by networked applications. The focus in this book is on the service and configuration design dimensions that address key networked application properties, such as duration and structure, how networked services are identified, and the time at which they are bound together to form complete applications.

- Chapter 3 describes the design and use of the ACE Reactor framework, which implements the Reactor pattern [POSA2] to allow event-driven applications to demultiplex and dispatch service requests that are delivered to an application from one or more clients.
- Chapter 4 then describes the design and use of the most common implementations of the `ACE_Reactor` interface, which support a wide range of OS event demultiplexing mechanisms, including `select()`, `WaitForMultipleObjects()`, `XtAppMainLoop()`, and `/dev/poll`.
- Chapter 5 describes the design and use of the ACE Service Configurator framework. This framework implements the Component Configurator pattern [POSA2] to allow an application to link/unlink its component service implementations at run time without having to modify, recompile, or relink the application statically.
- Chapter 6 describes the design and effective use of the ACE Task framework. This framework can be used to implement key concurrency patterns, such as Active Object and Half-Sync/Half-Async [POSA2].
- Chapter 7 describes the design and effective use of the ACE Acceptor-Connector framework. This framework implements the Acceptor-Connector pattern [POSA2] to decouple the connection and initialization of cooperating peer services in a networked system from the processing they perform once connected and initialized.
- Chapter 8 describes the design and use of the ACE Proactor framework. This framework implements the Proactor and Acceptor-Connector patterns [POSA2] to allow event-driven applications to efficiently demultiplex and dispatch service requests triggered by the completion of asynchronously initiated operations.
- Chapter 9 describes the design and use of the ACE Streams framework. This framework implements the Pipes and Filters pattern [POSA1] to provide a structure for systems that process streams of data.
- The book concludes with a glossary of technical terms, a list of references for further study, and a general subject index.

The chapters are organized to build upon each other and to minimize forward references. We therefore recommend that you read the chapters in order.

Although this book illustrates the key capabilities of ACE's most important frameworks, we don't cover all uses and methods of those frameworks. For additional coverage of ACE, we refer you to *The ACE Programmer's Guide* [HJS] and the online ACE reference documentation, generated by Doxygen [Dim01]. ACE's reference documentation is available at `http://ace.ece.uci.edu/Doxygen/` and `http://www.riverace.com/docs/`.

Related Material

This book is based on ACE version 5.3, released in the fall of 2002. ACE 5.3 and all the sample applications described in our books are open-source software. Sidebar 3 (page 19) explains how you can 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 the book.

To learn more about ACE, or to report 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 a request to `ace-users-request@cs.wustl.edu`. Include the following command in the body of the e-mail (the subject 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. If you use this alternate address method, the list server will require an extra authorization step before allowing you to join the list.

Postings to the `ace-users` list are also forwarded to the `comp.soft-sys.ace` USENET newsgroup, along with postings to several other ACE-related mailing lists. Reading the messages via the newsgroup is a good way to keep up with ACE news and activity if you don't require immediate delivery of the 30 to 50 messages that are posted daily on the mailing lists.

Archives of postings to the `comp.soft-sys.ace` newsgroup are available at `http://groups.google.com/`. Enter `comp.soft-sys.ace` in the search box to go to a list of archived messages. Google has a complete, searchable archive of over 40,000 messages. You can also post a message to the newsgroup from Google's site.

Acknowledgments

Champion reviewing honors go to Alain Decamps, Don Hinton, Alexander Maack, Chris Uzdavinis, and Johnny Willemsen, who reviewed the book multiple times and provided extensive, detailed comments that improved its form and content substantially. Many thanks also to the official reviewers, Timothy Culp, Dennis Mancl, Phil Mesnier, and Jason Pasion, who read the entire book and gave us many helpful comments. Many other ACE users provided feedback on this book, including Marc M. Adkins, Tomer Amiaz, Vi Thuan Banh, Kevin Bailey, Stephane Bastien, John Dilley, Eric Eide, Andrew Finnell, Dave Findlay, Jody Hagins, Jon Harnish, Jim Havlicek, Martin Johnson, Christopher Kohlhoff, Alex Libman, Harald Mitterhofer, Llori Patterson, Nick Pratt, Dieter Quehl, Tim Rozmajzl, Irma Rastegayeva, Eamonn Saunders, Harvinder Sawhney, Christian Schuhegger, Michael Searles, Kalvinder Singh, Henny Sipma, Stephen Sturtevant, Leo Stutzmann, Tommy Svensson, Bruce Trask, Dominic Williams, and Vadim Zaliva.

We are deeply indebted to all the members, past and present, of the DOC groups at Washington University in St. Louis and the University of California, Irvine, as well as the team members at Riverace Corporation and Object Computing Inc., 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, Luther Baker, Jaiganesh Balasubramanian, Darrell Brunsch, 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 50 countries who've contributed to ACE for over a decade. ACE's excellence and success is a testament to the skills and generosity of many talented developers and the forward-looking companies that had the vision to contribute their work to ACE's open-source code base. Without their support, constant feedback, and encouragement, we would never have written this book. In recognition of the efforts of the ACE open-source community, we maintain a list of all contributors 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), Frank Buschmann (Siemens), Dave Busigo (DARPA), John Buttitto (Sun), Becky Callison (Boeing), Wei Chiang (Nokia Inc.), Joe Cross (Lockheed Martin), Lou DiPalma (Raytheon), Bryan Doerr (Savvis), Karlheinz Dorn (Siemens), Scott Ellard (Madison), Matt Emerson (Escient Convergence Group, Inc.), Sylvester Fernandez (Lockheed Martin), Nikki Ford (DARPA), Andreas Geisler (Siemens), Helen Gill (NSF), 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), Joe Loyall (BBN), Kent Madsen (EO Thorpe), Ed Margand (DARPA), Mike Masters (NSWC), Major Ed Mays (U.S. Marine Corps), John Mellby (Raytheon), Jeanette Milos (DARPA), Stan Moyer (Telcordia), Ivan Murphy (Siemens), Russ Noseworthy (Object Sciences), Adam Porter (U. of Maryland), Dieter Quehl (Siemens), Vijay Raghavan (Vanderbilt U.), Lucie Robillard (U.S. Air Force), Craig Rodrigues (BBN), Rick Schantz (BBN), Andreas Schulke (Siemens), Steve Shaffer (Kodak), Tom Shields (Raytheon), Dave Sharp (Boeing), Naval Sodha (Ericsson), Paul Stephenson (Ericsson), Tatsuya Suda (UCI), Umar Syyid (Storetrax, Inc.), 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 acknowledge our gratitude and indebtedness to the late W. Richard Stevens, the father of network programming literature. The following poem by Samuel Butler sums up our view of Richard's enduring influence:

> Not on sad Stygian shore, nor in clear sheen
> Of far Elysian plain, shall we meet those
> Among the dead whose pupils we have been ...
> Yet meet we shall, and part, and meet again,
> Where dead men meet, on lips of living men.

Steve's Acknowledgments

Wow... C++NPv1 took almost 3 years to complete—this volume took roughly nine months. Thank you to my wife Jane who cheerfully endured this process. Your persistent exhortation to keep life in balance and "be the tortoise" really helped me stay the course, and without your infinite patience through many long days and nights, I would not have completed this—thank you! Thanks to Doug Schmidt for getting the bulk of this book down and organized in world-class time amidst a full-time job and his usual, amazing amount of work on ACE. Finally, thank you to Riverace's customers who supported this work so enthusiastically. It's a privilege to serve you.

Doug's Acknowledgments

I'd like to thank my wife Sonja and my parents for their love and support during the writing of this book. Now that it's done we'll have lots more time to have fun! Thanks also to Steve Huston, who time-shared his overloaded schedule to wrap up the book. I'd also like to thank my friends and colleagues at the College of William and Mary; Washington University, St. Louis; University of California, Irvine; Vanderbilt University; 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. I look forward to working with all of you in the future.

Contents