Developing Distributed Computing Systems with Patterns and Middleware

Douglas C. Schmidt
February 19-21, 2008
UCLA Extension, Los Angeles, California

A distributed system is a computing system in which a number of components cooperate by communicating over a network. The explosive growth of the Internet and the World Wide Web in the mid-1990s moved distributed systems beyond their traditional application areas, such as industrial automation, defense, and telecommunication, and into nearly all domains, including e-commerce, financial services, health care, government, and entertainment. Developing high-quality software for distributed systems is hard; successfully developing and applying high-quality reusable software is even harder. The principles, methods, and skills required to develop and apply reusable distributed systems software cannot be learned by generalities and concepts alone. Instead, researchers, developers, and educators must learn through hands-on experience how reusable software components, frameworks, and applications can be designed, implemented, optimized, validated, maintained, and enhanced by applying good development practices, patterns, tools, and platforms.

Intended for software developers who design and implement software for distributed systems, this course provides an active learning context where participants can significantly improve their skills related to building and applying reusable software assets for distributed systems. Emphasis is on hands-on exercises and analysis of case studies that illustrate by example how to create and apply effective reusable distributed systems software. A theme pervading this course is that knowledge of middleware, patterns, components, and frameworks--together with model-driven engineering (MDE) tools--can significantly reduce the complexity of developing software for distributed systems.

Two types of distributed software complexity are addressed:

Complexity Due to Changes in Functional Requirements
Some distributed system complexities are due to the need to accommodate changing requirements in functionality. Change is inevitable since user requirements, component interfaces, and developers' understanding of their application domain all change. Well-designed software must evolve to support these changes, especially in today's competitive global markets.

Complexity Due to Challenging Quality of Service (QoS) Requirements
Some distributed system complexities arise because software is required to perform its functionality while meeting challenging QoS properties, such as latency, jitter, scalability, dependability, and security. In many distributed systems the right answer delivered too late becomes the wrong answer, e.g., they do not function correctly if too much latency is incurred.

To address these complexities the course illustrates by example how to simplify and enhance the development of software for distributed systems by effective use of:

- Advanced application and middleware design techniques, such as patterns, layered modularity, information hiding, and generative design
- Language features, such as abstract classes; inheritance, dynamic binding, and parameterized types
- Middleware platforms, such as object-oriented network programming frameworks, request brokers, and service-oriented architectures
- Model-driven engineering (MDE) tools, such as Eclipse and the Generic Modeling Environment (GME)
- Advanced operating system mechanisms, such as asynchronous I/O, proactive and reactive event demultiplexing, multi-threading, and dynamic linking

PREREQUISITES
Familiarity with general object-oriented design and programming techniques, interprocess communication, and networking terminology.

COURSE MATERIALS
The text, *Pattern-Oriented Software Architecture: A Pattern Language for Distributed Computing*, F. Buschmann, K. Henney, and D. Schmidt (Wiley & Sons, 2007), and lecture notes are distributed on the first day of the course. The notes are for participants only and are not for sale.

COORDINATOR AND LECTURER
Douglas C. Schmidt, PhD, Professor, Electrical Engineering and Computer Science Department, Vanderbilt University, Nashville, Tennessee. Dr. Schmidt's pioneering research focuses on patterns, implementation, and experimental analysis of object-oriented techniques for developing high-performance distributed applications. In addition to his academic research, he has successfully applied object-oriented analysis, design, and programming techniques in a number of large-scale commercial projects. Dr. Schmidt is the chief architect and developer of the ADAPTIVE Communication Environment (ACE) and The ACE ORB (TAO), which are freely available object-oriented middleware frameworks that implement a rich set of design patterns that recur when building high-performance and real-time distributed systems software. ACE and TAO have been used successfully on many large-scale projects at companies, such as BBN, Boeing, Cisco, Ericsson, Lockheed Martin, Lucent, Motorola, Nokia, Nortel, Raytheon, and Siemens.

Dr. Schmidt has published widely in IEEE, IFIP, ACM, and USENIX technical conference papers and journals on communication software systems, parallel processing for high-performance networking protocols, distributed object computing, and object-oriented design patterns and programming. He is the lead author of the course text; the book *Patterns for Concurrent and Networked Objects*, Volume II (Wiley & Sons, 2000); the two volumes of *C++ Network Programming* books by Addison-Wesley; and has co-edited several other books on patterns and frameworks. In addition, he was editor-in-chief of the *C++ Report* magazine and writes a column on CORBA with Steve Vinoski for the *C/C++ Users Journal*. Dr. Schmidt has presented keynote addresses and tutorials on reusable design patterns, concurrent object-oriented network programming, and distributed object systems at hundreds of conferences.

UCLA FACULTY REPRESENTATIVE
Alfonso F. Cardenas, PhD, Professor, Department of Computer Science, Henry Samueli School of Engineering and Applied Science

COURSE PROGRAM
Overview of Distributed Computing Systems
The key characteristics and challenges of developing distributed computing systems; key software technologies that have emerged to resolve these challenges, including distributed object computing, component middleware, publish/subscribe middleware, service-oriented architectures, and Web services.

Overview of Patterns and Pattern Languages
Introduction to patterns, including their history, and a number of pattern concepts; the anatomy of a pattern--what it offers and what drives it; the relationships often found between patterns; pattern languages--what they are and how they can be presented and used for distributed computing systems.

Presentation of a Pattern Language for Distributed Computing Systems
A pattern language for distributed computing that addresses the following technical topics relevant for building distributed systems: specifying an initial software baseline architecture, understanding communication middleware, event demultiplexing and dispatching, interface partitioning, component partitioning, application control, concurrency, synchronization, object

interaction, adaptation and extension, modal behavior, resource management, and database access.

For more information call the Short Course Program Office at (310) 825-3344; fax (310) 206-2815.

Dates	February 19-21 (Tuesday through Thursday)
Time	8 am-5 pm (subject to adjustment after the first class meeting)
Location	Room 211, UCLA Extension Building, 10995 Le Conte Avenue (adjacent to the UCLA campus), Los Angeles, California
Reg#	
Course No.	Engineering 819.365
Units	1.8 CEU (18 hours of instruction)
Fee	$1,695, includes course materials
	$100 nonrefundable; no refund after February 7, 2008; however, course fee (less $100) may be applied toward another short course enrollment.