Panel

The Ultra Challenge: Software Systems Beyond Big

Steven Fraser (Chair)
QUALCOMM

Gregor Kiczales
University of British Columbia

Ricardo Lopez
QUALCOMM

Peter G. Neumann
SRI

Linda Northrop
SEI

Martin Rinard
MIT

Douglas Schmidt
Vanderbilt University

Kevin Sullivan
University of Virginia

Abstract

How can the *ultra large systems (ULS)* of the future be built if they will have the complexity of trillions of lines of code, maintain continuous 24x7 operations with no downtime, and live in a hostile environment with unpredictably changing requirements? This panel will discuss and debate the challenges posed by *ultra large systems* in terms of their design, growth, deployment and dynamics.

Categories & Subject Descriptors:
H.4 Information Technology and Systems
J.9 Computer Applications
K.0 Computing Milieux

General Terms: Design, Management, Reliability, Security

Keywords: Complexity, design, software, systems, ultra, ULS

1. Steven Fraser (chair), sdfraser@acm.org

STEVEN FRASER is a member of QUALCOMM's Learning Center in San Diego, California with responsibilities for technical learning and development. Prior to joining QUALCOMM, Fraser held a variety of diverse software technology program management roles at Nortel/NT/BNR including: Process Architect, Senior Manager (Disruptive Technology and Global External Research), and Process Engineering Advisor. In 1994 he spent a year as a Visiting Scientist at the Software Engineering Institute (SEI) collaborating with the Application of Software Models project on the development of team-based domain analysis techniques. Fraser was the General Chair for XP2006, the Panels Chair for OOPSLA'03 and has organized panels at both OOPSLA and reuse/agile-oriented conferences. Fraser holds a doctorate in Electrical Engineering from McGill University in Montréal – and is a member of the ACM and a senior member of the IEEE.

2. Gregor Kiczales, gregor@cs.ubc.ca

GREGOR KICZALES is a full professor at the University of British Columbia. A primary theme of his work is focused on enabling programmers to write programs that, as much as possible, look like their design. He also seeks to unravel inconsistencies between our field's accounts of computing and the real nature of the beast. While at Xerox PARC, he led the teams that developed aspect-oriented programming and AspectJ. He was the principal designer of the CLOS metaobject protocol, and was one of the designers of the Common Lisp Object System.

Existing programming languages, and theoretical foundations on which they are built, are based on idealizations of software that are tenuous in current practice and that will break down for ULS systems. Our foundations treat software as abstract, isolated, closed-world, "mathematical" programs. But real software does not fit this idealization at all - instead it is a concrete intentional artifact that is richly embedded into an environment of physical and intentional artifacts. One challenge is developing an ability to better cope with the rich semantic relationships between a program and the rest of the world. Identifiers in the code may refer to entities outside the computation the code directly engenders. Intentional artifacts other than code form part of the overall software ecosystem and have semantic references to and from the code (configuration files, build scripts, bug databases, email archives of design discussions and so on). The validity at any moment in time of these intentional relationships is a critical factor in whether the code does as intended. Another challenge is to better cope with the fact that computations are physical processes, running on real computers over real networks. One of our most pervasive memes is that computer scientists don't have to respect the laws of physics. But in fact software is encoded physically, and has classic properties of physical systems, including the fact that its size affects aspects of viability and behavior such as scalability, distribution, latency etc. Many of the problems we struggle most with arise from our inability in fact to ignore physics. These differences in our field between our current foundations and the reality of large complex systems are more than just a typical difference between theory and practice. In important ways our present foundations do not just simplify practice; they fail to account for essential aspects of software in the real world. These limitations cause us problems today, and unless we can make progress in these areas they will become major obstacles to developing ULS systems.

OOPSLA'06 October 22–26, 2006, Portland, Oregon, USA.
ACM 1-59593-491-X/06/0010.

3. Ricardo Lopez, rjlopez@qualcomm.com

RICARDO LOPEZ is a Principal Engineer in the Office of the Chief Scientist at QUALCOMM. He is responsible for software architecture, software process, and sometime *Just Plain Old Software* (JPOS). Architecting and designing software for over thirty years, he has been an evangelist for OO technology for the last twenty years and he has the arrow heads to prove it.

Providing software in today's world requires a deep understanding of the increasing dependence that civilization has placed upon this uniquely human artifact; it begs improved sensitivity to the inherent complexities naturally encountered as we expand the horizons of our software into ubiquitous computing and all the increased leverage that will entail; and it demands a commitment to quality, security and trust not found currently on the campuses of our largest software producers.

4. Peter G. Neumann, neumann@csl.sri.com

Peter G. Neumann, (PhD, Dr.rer.nat.) has been at SRI International in Menlo Park, California for 35 years (where he is Principal Scientist in the Computer Science Lab), following 10 years at Bell Labs in Murray Hill, New Jersey. His research has been concerned with high-assurance systems and how they can be developed as predictable compositions of more easily understood subsystems. He was heavily involved in Multics in the 1960s. At SRI, his research has focused on computer systems and networks, including the design of the Provably Secure Operating System PSOS (an object-oriented capability-based architecture project from 1973 to 1980), trustworthiness/dependability, security, reliability, survivability, safety, formal methods, and many risks-related issues such as voting-system integrity, crypto policy, social implications, and human needs including privacy. His 1995 book, Computer-Related Risks, is still timely. He is a Fellow of the ACM, IEEE, and AAAS. He received the National Computer System Security Award in 2002 and the ACM SIGSAC Outstanding Contributions Award in 2005. He is a member of the U.S. Government Accountability Office Executive Council on Information Management and Technology, and the California Office of Privacy Protection advisory council. He has taught courses at Darmstadt, Stanford, U.C. Berkeley, and the University of Maryland. Visit www.csl.sri.com/neumann for further background, Senate and House testimonies, papers, bibliography, and www.csl.sri.com/neumann/chats4.pdf and .html.

Consider two straw-man polarized approaches for ultra-large-scale systems:

1. If we had trustworthy operating systems, predictably composable architectures that are designed to be easily administratable and evolvable, realistic system development and software engineering practices, inherently sound programming languages, pervasive use of assurance techniques, and better education that supported all of that, THEN we might be able to build trustworthy ultra-large-scale systems. This is a little like the joke about "if we had ham, we could have ham and eggs – if we had eggs."

2. If we slap together an unbounded collection of often incompatible architecture-less poorly designed untrustworthy components into an open-ended Internet-accessible unconstrained environment, with neither far-sighted *a priori* requirements nor an engineering-oriented development discipline, and with potentially untrustworthy users and diverse would-be adversaries, THEN we are not likely to have any assurance that the systems would do what is really needed. For realistic ultra-large-scale systems, can we get there from here? Are there any feasible middle grounds? What is most important? Are we shooting a straw herring in the foot?

5. Linda Northrop, lmn@sei.cmu.edu

LINDA NORTHROP has more than 35 years of experience in software development as a practitioner, researcher, manager, consultant, and educator. She currently is director of the Product Line Systems Program at the SEI where she leads the work in software architecture, software product lines, and predictable component engineering. She recently led a year long study including leaders in the software community to define technical and social challenges to the creation of ultra-large-scale systems that will evolve in the next generation. The report, *Ultra-Large-Scale Systems: The Software Challenge of the Future* (ISBN 0-9786956-0-7), has just been published. She is coauthor of *Software Product Lines: Practices and Patterns* and chaired both the first and second international Software Product Line Conferences (SPLC1 and SPLC2). She is a past chair of the OOPSLA Steering Committee and OOPSLA 2001 conference Chair. Before joining the SEI, she was associated with both the United States Air Force Academy and the State University of New York as professor of computer science, and with both Eastman Kodak and IBM as a software engineer. As a private consultant, Linda also worked for an assortment of companies covering a wide range of software systems. She is a recipient of the Carnegie Science Award of Excellence for Information Technology and the New York State Chancellor's Award for Excellence in Teaching.

Today's primary approach to system monitoring and assessment is through the use of measurements. We characterize the quality of a system by a set of measurements captured at critical probe points defined for the system's constituent component and networks. Using measurements we monitor a system's health in terms of its security, availability, performance, reliability, usability, etc. The scale, decentralization, distribution, and heterogeneity of ULS systems will challenge today's measurement practices. It is not clear what we should measure. What system-wide, end-to-end and local quality-of-service indicators are relevant to ULS systems? For example,

does reliability as we know it have a meaning in a ULS system? Moreover, we don't have a clear understanding of why those indicators would change, how they should be prioritized, and how measurement processes will handle continual changes to components, services, usage, and connectivity. People are also an important integral part of ULS systems and so there must be indicators for the human, organizational, economic, and business elements of the system as well as the technical elements. It is likely that at least some ULS system indicators should be statistical, composite measures of a system's overall state, like the gross national product. Current quality measures and measurement process will simply not be adequate for ULS systems.

6. Martin Rinard, rinard@cag.csail.mit.edu

MARTIN RINARD is a Professor in the MIT Department of Electrical Engineering and Computer Science and a member of the MIT Computer Science and Artificial Intelligence Laboratory. His research interests have included parallel and distributed computing, programming languages, program analysis, program verification, and software engineering. Much of his current research focuses on techniques that enable software systems to execute successfully in spite of the presence of errors. His research results include a semantics for concurrent constraint languages, a meta-language for implicitly parallel programs, commutativity analysis for automatically parallelizing object-oriented programs, synchronization optimizations for eliminating locking overhead in parallel programs, new pointer and escape analysis algorithms, the Hob and Jahob program analysis and verification systems, data structure repair (which enables programs to recover from data structure corruption errors), and failure-oblivious computing (a technique for enabling programs to execute successfully through otherwise fatal memory addressing errors).

All ultra large systems are built as assemblages of existing systems. The most important challenges that developers of ultra large systems face are therefore already well known to the software development community. The scale and ambition of these systems, however, changes the context in which these challenges play out. The result is that some challenges may become more difficult to deal with successfully, in some cases requiring new perspectives, new expectations, or new system construction, operation, and maintenance techniques. Major development issues will revolve around the difficulty of making multiple different software systems interact successfully. The standard approach is to use interaction mechanisms from middleware packages. The continuing need to make multiple software systems interact will motivate the development of new and more elaborate middleware packages that support an increasing variety of interaction paradigms. In particular, existing software systems are brittle in that they often fail catastrophically when presented with unexpected or missing interactions. One new middleware theme will be the automatic conversion of potentially problematic interactions into interactions that conform to the expectations of the involved software systems. Techniques may include automatically changing potentially problematic values, synthesizing missing interactions, and removing interactions that may cause problems for the receiving system.

A second set of issues will center around appropriately configuring the component software systems as they are transplanted from their original operating environments into their new environment as part of the larger system. Developers will increasingly employ multiple virtual (or, in some cases, even physical) machines as one mechanism to replicate, as closely as possible, the original operating environment so as to maximize the chances that the systems will operate successfully. In some cases the virtual machine may develop to the point that it becomes an elaborate simulation of the environment (both physical and logical) in which the system was originally deployed. Another motivation for these techniques will be the need to support software systems that run only in obsolete operating environments. We expect that developers will usually accept the (often substantial) inefficiencies associated with the extensive use of middleware and virtual machines as a matter of course. In a sufficiently large system, some parts will always be operating sub-optimally or even completely broken. The need to keep the system operating in the face of these defects will motivate the development of new techniques for graceful degradation in the face of problems: techniques that enable software systems to execute through otherwise fatal errors, replace broken components with much simpler components with reduced functionality, and operate in the presence of interaction patterns that the software was originally not designed to support.

One perspective change will be an acceleration of the shift from deductive to inductive reasoning about the operation of the system. In a large system nobody understands the individual components and their interactions well enough to accurately predict its behavior. Instead, the system will be constructed, its behavior observed, and adjustments made as undesirable behaviors become suspected or apparent.

A key development risk associated with large systems is the possibility that the system will not work (or will take an unacceptable amount of time or resources to develop) because of unrealistic expectations about the behavior, functionality, resource consumption, or performance characteristics of one or more of the components when operating within the final system. Competent organizations will minimize, but not eliminate, these risks by designing their systems with the aid of experts who specialize in understanding the software systems that the organization plans to use. It will be especially important to recognize

unrealistic development plans quickly to minimize the amount of wasted development time and resources. Note that the extensive use of existing systems in a new development effort can make the amount of time and effort required to complete the development less predictable.

Because nobody will understand how a sufficiently large system will behave in its full range of potential operating conditions, there is always the risk that a deployed system will unexpectedly fail to provide acceptable service. While society largely accepts substantial (and arguably unnecessary) amounts of mortality and morbidity in endeavors not perceived to involve large software systems (examples include traffic accidents, medical errors, and gunshot wounds), it remains to be seen how society will react to disasters directly attributable to unexpected aspects of the operation of large software systems. It is possible that we will find out the answer to this question over the next several decades as newly deployed software systems directly affect an increasingly large part of society. On the other hand, software systems have been the cause of remarkably few disasters in the past, and while there is obviously some limit to the size and complexity of software systems that we can safely develop and deploy, it is not clear how close we are to that limit.

7. Douglas C. Schmidt, schmidt@dre.vanderbilt.edu

DOUGLAS C. SCHMIDT is a Full Professor in the Electrical Engineering and Computer Science Department, the Associate Chair of Computer Science and Engineering, and a Senior Researcher at the Institute for Software Integrated Systems (ISIS) at Vanderbilt University. For over a decade, his research has focused on patterns, optimization techniques, and empirical analyses of object-oriented and component-based frameworks and model-driven development tools that facilitate the development of distributed real-time and embedded (DRE) middleware and applications on parallel platforms running over high-speed networks and embedded system interconnects. Schmidt is an internationally renowned and widely cited expert on distributed computing middleware patterns, middleware frameworks, and Real-time CORBA. He has published over 300 works in top IEEE, ACM, IFIP, and USENIX technical journals, conferences, and books that cover a range of topics, including high-performance communication software systems, parallel processing for high-speed networking protocols, real-time distributed computing with CORBA, Real-time Java, object-oriented patterns for concurrent and distributed systems, and model-driven development tools. Schmidt received B.S. and M.A. degrees in Sociology from the College of William and Mary in Williamsburg, Virginia, and an M.S. and a Ph.D. in Computer Science from the University of California, Irvine (UCI).

Software engineers have traditionally created, integrated, and tested entire systems internally in-house before shipping them to end-users. In contrast, ULS systems will increasingly be developed in situ in the deployment environment due to the blurring between design time and runtime. This trend creates the need for synergistic model-driven engineering environments and runtime platforms for deploying, configuring, and validating the behavior of reusable components and integrated applications to ensure they meet the quality-of-service (QoS) requirements in the context in which they execute.

The scale of ULS systems will also require in situ systems, processes, and techniques for measuring, analyzing, and modeling the interactions between configuration choices and the achievement of desired functional and QoS qualities. Unlike traditional small-scale software systems that could be analyzed and validated using precise quality assurance (QA) techniques, ULS systems will need advances in statistical, usage-based QA techniques to provide confidence that their implementations can be depended on to meet end-to-end requirements in harsh operational environments. ULS systems will therefore need distributed continuous QA environments that can divide complex QA processes into multiple test subtasks, intelligently disseminate and execute these subtasks to a distributed grid of in-house and in-the-field platforms, and fused together the results to inform and guide the QA process.

8. Kevin Sullivan, sullivan@cs.virginia.edu

KEVIN SULLIVAN is an Associate Professor and Virginia Engineering Foundation Faculty Fellow in Computer Science at the University of Virginia. He received his undergraduate degree from Tufts University in 1987 and the PhD in Computer Science and Engineering from the University of Washington in 1994. His research interests are in software-intensive systems, in general, and in software engineering and languages, in particular. He has long been interested in issues and models of modularity in software design, and is currently working on the technical and economic aspects of modularity in software system design. Sullivan also has broad interests in the dependability characteristics of software and software-intensive systems. He has served as associate editor for the ACM Transactions on Software Engineering and Methodology and the Journal of Empirical Software Engineering and on the program and executive committees of many top conferences including the International Conference on Software Engineering (ICSE), the ACM SIGSOFT Symposium on the Foundations of Software Engineering (FSE), the ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL) and the Aspect-Oriented Software Development (AOSD) conference. Sullivan was also a member of the core team that wrote the Ultra-Large-Scale Systems report recently issued by the Software Engineering Institute.

The theory and practice of software-intensive system design has traditionally focused on interesting and impor-

tant technical aspects of the activity: e.g., formalizing architecture, program synthesis, the semantics of programming languages, notations and mechanisms for specification and verification, compositional components, etc. Although such work is elegant and of great value, it is not enough, by itself, to enable adequately the reliable development of successful ULS systems.

One underlying problem is that we lack a theory and practice connecting technical aspects of design to what Studenmaier calls the ambience of design: the "economic, military, social, personal and environmental needs and constraints" that surround designs and designed artifacts, and to which they respond (Vincente, 1990). One quick glance at the state of the art in software economics makes the point: research has focused mainly on cost and schedule estimation, but little progress has been made on modeling and analyzing the ways in which design strategies and structures satisfy ambient economic goals. Simply put, there has been little work on reliable estimation of economic benefits. For example, even though modularity in software design has been studied for decades as a technical issue, testable theories of the economics of modularity are just emerging — and not primarily from within the software engineering and languages field.

On this panel Sullivan will take the position that what we need is a significantly increased emphasis on fundamental and applied research leading to testable scientific theories and useful mechanisms connecting technical aspects of software-intensive system design to ambient objectives and constraints. Our capitalist economic system in particular dictates that most designers see and seek economic value in new designs (Baldwin and Clark, 1999). It is thus particularly important that we develop a rich scientific understanding of design economics. This task is not one that likely can be accomplished by researchers in computer science and engineering alone, but will require close collaboration between us and design-focused researchers in other disciplines, including but not limited to financial economics.

Success in this endeavor is especially needed to enable the development of ULS systems. Their complexity will be so great that tight control of design, development, deployment, operation and evolution will in many cases be untenable. Rather, our main viable point of influence at the whole-system level will be to engineer the ambience so as to create conditions within which decentralized design activities will nevertheless produce satisfactory problem definitions, design architectures, components and systems. Rather than focusing just on improving our abilities to tightly manage technical aspects of designs, we thus also need to develop new approaches to what might be called design environmentalism and stewardship. Heaping on another metaphor, we need to shift our attention from engineering the plants, which is untenable, to engineering the gardens so that the plants will grow (borrowed from Dick Gabriel). A scientific discipline of design economics, as described here, has the potential to deliver the concepts, methods and tools we need to reliably establish and maintain fertile ambient conditions — gardens — in which successful ULS system designs and artifacts might emerge and grow healthy over time.