# Increasing ULS System Quality with System Execution Modeling Tools

John M. Slaby, Raytheon Integrated Defense Systems
James H. Hill, Vanderbilt University

## 1. Challenges of ULS System Development

A key research area identified in the Ultra-Large Scale (ULS) Systems report published by the Software Engineering Institute is *Adaptable and Predictable System Quality*. This section describes how "managing traditional qualities such as security, performance, reliability, and usability is necessary but not sufficient to meet the challenges of ULS systems" and that research is necessary to understand "how to maintain quality in a ULS system in the face of continuous change, ongoing failures, and attacks"[1]. The report elaborates by noting that "ULS systems will be long running and must operate robustly in environments fraught with failures, overloads, and attacks. … Moreover, ULS systems must maintain robustness in the presence of adaptations that are not centrally controlled or authorized. Some degree of system failures (hardware and software) will be intrinsic to ULS systems… It is inevitable that ULS systems will be tempting targets of attack for capable and motivated adversaries seeking tactical and strategic advantages" [2].

Traditional methods for managing the quality of distributed systems included manually applying software updates to patch security holes, or taking a system completely offline to repair failures. ULS systems, however, have several characteristics, such as scale, geographic location, and continuous availability that impede the application of traditional methods for managing quality. For example, in some cases a ULS system must be available 24x7 because it is offering services to users and other ULS systems, so taking it offline to apply security patches or functional updates is not an option. Likewise, ULS systems that run over extended periods of time can exhibit failures, and must compensate for such failures without affecting to quality of other systems utilizing their services. It is therefore clear that new methods are needed to manage the quality of ULS systems.

## 2. Solution Approach → Applying System Execution Modeling Tools to Evaluate ULS System Architectures and Deployments

To address the challenges in ULS systems, there is a need for a methodology and an associated suite of system execution modeling (SEM) tools [3] that use model-driven engineering (MDE) technologies to simplify the:

- **Emulation of application component behavior** in terms of computational workloads, resource utilizations and requirements, and network communication. This step should be done quickly and precisely using domain-specific modeling languages (DSMLs) that capture the behavior and workload of system components at a high-level of abstraction and then generate code that executes emulated components.
- **Configuration, deployment, and execution of the emulated application components** atop actual infrastructure components to determine their impact on QoS empirically in actual runtime environments. These steps should also be accomplished using DSMLs that specify realistic deployments and configurations and synthesize metadata describing it. The metadata is then processed by the same deployment and configuration tools as the actual system, with SEM tools providing mechanisms to record, consolidate, and collect QoS metrics (such as execution times and resource usage) from the runtime environment.
- **Process of feeding back the results to enhance system architectures and components** to improve QoS. This step would be accomplished by archiving the collected QoS metrics and providing tools that view the overall results of a deployment. The SEM tools would also provide histories of the collected metrics to enable engineers and architects to understand performance and make well-informed decisions to improve QoS.

Over time as the actual application components mature, they can replace the emulated components, thereby providing an ever more realistic evaluation environment for ULS systems. Figure 1 shows the relationships between the steps described above.

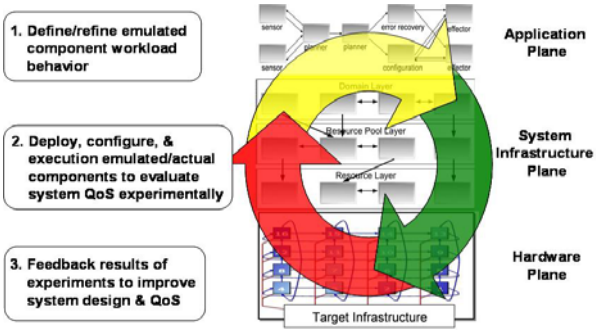


**Figure 1. Evaluating the QoS of ULS Systems via System Execution Modeling Tools**

SEM tools enable system engineers, software architects/developers, and quality assurance (QA) engineers to grapple with the inherent complexities that arise from real world physical properties, such as communication delay, temporal phasing, parallel execution, and synchronization. Typically there are only a few execution designs that actually can satisfy the functional and performance requirements established in ULS system software architectures. SEM tools enable architects and engineers to discover, measure, and rectify incipient integration and performance problems early in a ULS system's lifecycle (e.g., in the analysis and/or design phases), thereby shifting the focus of the software integration resources to productive activities that evaluate and validate system performance and end-user value, rather than serving as the de facto system design debugging activity as is often the case today.

## 3. Overview of the Component Workload Emulator Utilization Test Suite (CUTS)

To evaluate the benefits of SEM tools in the context of distributed real-time and embedded (DRE) systems we have developed the *Component Workload Emulator (CoWorkEr) Utilization Test Suite (CUTS)* [4] and applied it to large-scale shipboard computing systems. CUTS combines quality-of-service (QoS)-enabled component middleware and MDE technologies. CUTS allows software architects, developers, and systems engineers to explore design alternatives from multiple computational and valuation perspectives during all lifecycle phases. Unlike pure simulation tools, CUTS runs in environments representative of the target architecture and collects realistic measurements of application behavior and resource usage. It also facilitates analytical methods to quantify the costs of certain design choices on end-to-end system performance. For example, CUTS can help determine the maximum number of components a host can handle before performance degrades; the average and worse response time for various workloads; or the ability of alternative system configurations and deployments to meet end-to-end QoS requirements for a particular workload.

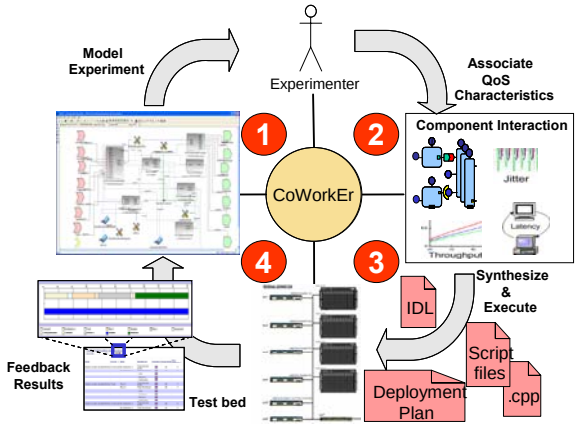


**Figure 2. The CUTS Architecture and Workflow**

Figure 2 shows how software architects, developers, and systems engineers specify the structure of the system, e.g., the component and their interconnections using CUTS DSMLs (step 1), and associate the necessary QoS characteristics (step 2) with individual components (e.g., CPU utilization) or the system as a whole (e.g., execution deadline of a critical path through the system). The information captured is then syn-

thesized into executable code and configuration metadata (step 3), which middleware uses to deploy the emulated/actual application/system components onto the target architecture. System developers and engineers then analyze the collected metrics (step 4) and explore design alternatives from multiple computational and valuation perspectives to quantify the costs of certain design choices on end-to-end system performance.

## 4. Scaling CUTS and SEM Tools to Meet the Needs of ULS Systems

Although CUTS provides techniques that apply well in the context of today's DRE systems, tomorrow's ULS systems have key differences, such as the scale and degree of ownership and control over system elements. We posit, however, that the core CUTS technologies can be enhanced to address many of the challenges with ULS systems identified above. In particular, we envision typical scenarios using the enhanced CUTS tool chain for ULS systems as follows:

1. **Modeling system behavior and quality.** Using the CUTS DSMLs [5], we could build models that capture the providers and users of component services and their respective interconnections with other components. We could then model and record the normal interactions of the users and providers, and add to the model abstract elements to represent unexpected behavior, such as request overload, denial-of-service attacks, network failures or unacceptable jitter that would cause components to fail in production environments. The code that emulates this behavior could be generated from the model as CoWorkErs and used as a basis for validating various ULS system component properties, such as best/average/worst availability and quality metrics under best/worst case expected/unexpected scenarios, or ability to gracefully handle unexpected user and system behavior.

2. **Emulating and analyzing system quality.** The generated CoWorkErs and real components could then be deployed in a testing environment representative of the production environment, and scenarios could be run that allow generated CoWorkErs to evaluate different interactions with real components. These test scenarios would stress the real components by exposing interactions that would not result in normal operations and presumably violate their design intents and requirements. Lightweight instrumentation [6] of CoWorkErs could provide feedback to the users, e.g., unhandled events or failures, to help developers understand component vulnerabilities and evaluate which strategies could be effective in mitigating these vulnerabilities. We expect that patterns will emerge which maximize different QoS aspects, e.g., availability and reliability, for ULS system components in various environments. These patterns can be captured in MDE tools and applied in subsequent development iterations to improve ULS system quality.

3. **Maintaining system quality.** The CUTS DSMLs that capture unexpected behavior are extensible and can therefore be expanded to include new problems, attacks, and abnormalities of ULS systems as they are gleaned from production system experience. Likewise, these behaviors can be included in models, generated into CoWorkErs, and used to continuously stress different quality aspects of a component. While an actual ULS system environment is too large to replicate entirely in a test environment, enhancements to CUTS could allow us to use lightweight partial analysis techniques such as statistical, incremental, usage-based distributed continuous quality assurance techniques [7] to capture meaningful results. Rather than modeling every possible configuration that components may have in a ULS system, we are building an environment that requires only enough resources to host the components being tested, along with the set of emulated users. We then generate broad test coverage representative of ULS systems through the rich and varied behavior of these emulated services. It should, therefore, be possible to create multiple test environments at reasonable costs that contain different platforms, operating systems and middleware, and types of network connectivity to provide the necessary execution environment coverage that is representative of a ULS system.

## References


[1] Software Engineering Institute, Ultra-Large-Scale Systems: The Software Challenge of the Future, June 2006, Pittsburgh, PA, p 30.

[2] Ibid, p 38.

[3] Smith, C. and Williams, L. *"Performance Solutions: A Practical Guide to Creating Responsive, Scalable Software,"* Addison-Wesley, 2002.

[4] Slaby, J., Baker, S., Hill, J. and Schmidt, D. “Applying System Execution Modeling Tools to Evaluate Enterprise Distributed Real-time and Embedded System QoS,” *In Proceedings of the 12th International Conference on Embedded and Real-Time Computing Systems and Applications*, Sydney, Australia, August 2006.

[5] Paunov, S., Hill, J., Schmidt, D., Slaby, J., and Baker, S., “Domain-Specific Modeling Languages for Configuring and Evaluating Enterprise DRE System Quality of Service,” *Proceedings of the 13th IEEE International Conference and Workshop on the Engineering of Computer Based Systems*, Potsdam, Germany, Mar 2006.

[6] Karr, A. and Porter, A., “Distributed Performance Testing using Statistical Modeling”, *Proceedings of the ICSE 2005 Workshop on Advances in Model-Based Software Testing*, St. Louis, MO, May 15-16, 2005.

[7] Yilmaz, C., Krishna, A., Memon, A., Porter, A., Schmidt, D., Gokhale, A., and Natarajan, B., “Main Effects Screening: A Distributed Continuous Quality Assurance Process for Monitoring Performance Degradation in Evolving Software Systems,” in Proceedings of the 27th International Conference on Software Engineering, St. Louis, MO, May 15-21, 2005.

## Author Bios

**John M. Slaby** is currently a Sr. Principle S/W Engineer with Raytheon Integrated Defense Systems. He has spent over 25 years in software engineering working for high technology start-ups focused on data networking. He is currently involved in research focused on dynamic resource management and model-based engineering using domain-specific modeling languages. His background includes software engineering, engineering management, architecture, product management, product marketing, training, and customer support.

**James H. Hill** is a Ph.D. student at Vanderbilt University. He received his B.S. in Component Science from Morehouse College and M.S. in Computer Science from Vanderbilt University. Over the past 5 years he has interned with CalTech, NASA’s JPL, and Raytheon Integrated Defense Systems, where at each he designed and implemented emulation and evaluation frameworks for small- and large-scale software systems. His Ph.D. research focuses on quality-of-service enabled middleware and model-driven engineering (MDE). He is currently applying MDE techniques to address the development challenges of large-scale distributed real-time and embedded systems using next generation component-based technologies.