Feedback–driven Design of Distributed Real-time & Embedded Component Middleware Via Model-Integrated Computing & Distributed Continuous Quality Assurance

Atif Memon, Adam Porter, Douglas Schmidt

1 Motivation

Middleware, increasingly found in most software-intensive systems, is software that resides between applications and the underlying operating systems and networks and is responsible for functionally bridging the gap between applications and the lower-level hardware/software infrastructure. *Component middleware* is a rapidly maturing type of middleware that enables component services to be composed, configured, and deployed to create distributed systems rapidly and robustly. Examples of component middleware include the CORBA Component Model (CCM), Java 2 Enterprise Edition (J2EE), and Microsoft Component Object Model (COM).

Component middleware was originally designed to support enterprise systems, such as airline reservation systems, bank asset management systems, and inventory control systems. It has recently begun to be applied to distributed real-time and embedded (DRE) systems, such as process control systems, on-board diagnostic and entertainment systems in automobiles, mobile devices with strict memory and power consumption requirements, and avionics mission computing systems. By definition, DRE systems are performance-intensive and possess stringent quality of service (QoS) requirements for latency, efficiency, scalability, dependability, and/or security. A key source of DRE system complexity is the presence of many *aspects* (such as distribution, concurrency, synchronization, fault tolerance, persistent data store, security, portability, andmanagement of run-time resources) that cut across the functional component boundaries of DRE software.

The design of software for complex DRE systems is increasingly subject to the following trends:

- **Demand for user-specific customization**. Since performance-intensive DRE software pushes the limits of technology, it must be optimized for particular run-time contexts and application requirements. Thus, general-purpose, one-size-fits-all off-the-shelf software solutions often have unacceptable performance.
- **Severe cost and time-to-market pressures**. Global competition and market deregulation are shrinking budgets for the development and quality assurance (QA) of software in-house, particularly for operating system and middleware infrastructure. Moreover, users of performance-intensive DRE systems are often unable or less willing to pay for specialized proprietary infrastructure software. The net effect is that fewer resources are available to devote to infrastructure software development and QA activities.
- **Distributed and evolution-oriented development processes**. Today's global IT economy and n-tier architectures often involve developers distributed across geographical locations, time zones, and even business organizations. The goal of distributed development is to reduce cycle time by having developers work simultaneously, with minimal direct inter-developer coordination. Such development processes can increase churn rates in the software base, which in turn increases the need to detect, diagnose, and fix faulty changes quickly. The same situation occurs in evolution-oriented processes, where many small increments are routinely added to the base software repository.

As these trends accelerate, they present many challenges to developers of performance-intensive DRE systems. A particularly vexing trend directly related to the design and adaptability of the middleware of these systems, is the explosion of the software *configuration space*. To support customizations demanded by users, performance-intensive DRE software must run on many hardware and OS platforms and typically have many options to configure the system at compile- and/or run-time. For example, performance-intensive DRE middleware, such as real-time object request brokers (e.g., TAO) possess dozen or hundreds of options, as do more general-purpose performance-intensive middleware, such as web servers (e.g., Apache) and databases (e.g., Oracle). While this flexibility promotes customization, it creates many potential system configurations, each of which may need extensive QA to validate.

When the increasing configuration space of performance-intensive DRE software is coupled with shrinking software development resources, it becomes infeasible to handle all design and validation in-house. For instance, developers may not have access to all the hardware, OS, and compiler platforms on which their software will run. Due to time-to-market driven environments, therefore, developers must often release their software in configurations that have not been subjected to extensive QA. Moreover, the combination of an enormous configuration space and severe development constraints mean that developers must make design and optimization decisions without precise knowledge of their consequences in fielded DRE systems. What is needed is a flexible design process for component middleware that can be optimized/evolved as and when feedback is available from fielded systems.

2 Research Challenges and Promising Solutions

To address the challenges outlined in Section 1, we believe that an integrated set of platforms, tools, and software development processes is needed that can (1) select and validate a suitable configuration of QoS-enabled middleware and application components, (2) automatically optimize, generate, and deploy cross-cutting and optimized configurations of QoS-enabled component middleware and application services and (3) validate key qualities of the resulting configurations across the range of platforms on which they execute. The emergence of domain-specific languages, modeling tools, and distributed continuous testing processes is creating an opportunity for increased automation in generating, integrating, and validating applications and QoS-enabled component middleware for DRE systems. **Our research therefore focuses on developing, optimizing, and validating model-integrated generator tools and feedback-driven design of distributed**

continuous QA processes to bridge the gap between specification, implementation, and validation of QoS-enabled component middleware and applications for DRE systems. We are addressing the following research challenges in support of our R&D activities:

1. ***Satisfying multiple QoS requirements in real-time***. An increasing number of applications for DRE systems require stringent QoS demands (such as latency, jitter, throughput, security, reliability, and scalability) that must be satisfied simultaneously and in real-time. Our initial research focuses on ensuring latency, jitter, and scalability requirements for DRE avionics mission computing systems (such as a GPS-based autopilot navigation system).
2. ***Alleviating complexities in integrating and validating DRE software applications***. To reduce lifecycle costs and decrease time-to-market, application developers for DRE systems are attempting to assemble and deploy their applications by selecting the right set of semantically compatible components, which is a difficult task. Our research focuses on model-integrated techniques for designing, implementing/generating, packaging and deploying QoS-enabled component middleware to leverage environment characteristics, such as special-purpose hardware, high-speed network interconnects, advanced OS features, and large-scale network topologies.
3. ***Integrating different middleware technologies***. To provide sufficient QoS support and take advantage of new technologies as they arise, the middleware we are developing and validating works with heterogeneous OS platforms (such as Windows, UNIX, and VxWorks), interface with applications written in different languages (such as Java, C++, and C), and interoperate with multiple component middleware technologies (such as CCM and J2EE).
4. ***Validating runtime properties with distributed, continuous quality assurance processes***. The combination of numerous software aspects and heterogeneous platform support creates an explosion in the configuration space of these software systems. We are therefore developing and refining techniques for intelligently partitioning QA tasks across members of user communities who are geographically distributed throughout the world.

3 Synopsis of Our Ongoing Research Efforts

To address the research challenges described in Section 1, we are combining and enhancing the following technologies, platforms, and tools:

1. **QoS-enabled component middleware** tools (such as ***The ACE ORB (TAO)*** <www.cs.wustl.edu/~schmidt/TAO.html> and the ***Component-Integrated ACE ORB (CIAO)*** <www.dre.vanderbilt.edu/CIAO>) that we have developed. TAO is an open-source, widely used, and highly configurable Real-time CORBA Object Request Broker (ORB) that implements key patterns to meet the stringent QoS requirements of DRE applications. CIAO is our CORBA Component Model (CCM) ORB based on TAO that is targeted for component-based DRE applications.
2. **Model-Integrated Computing** (MIC) technologies we are developing in the ***Component Synthesis with MIC (CoSMIC)*** <www.isis.vanderbilt.edu/CoSMIC> project. CoSMIC is a tool-suite that integrates and extends MIC tools (such as the ***Generic Modeling Environment (GME)*** <www.isis.vanderbilt.edu/Projects/gme/default.html>) to model and analyze different–but interdependent–characteristics of middleware behavior to check automatically whether the requested behavior and properties are feasible given the constraints**.**
3. **Distributed continuous quality assurance (QA)** processes we are developing as part of the multi-site Skoll project <www.cs.umd.edu/projects/Skoll.> Skoll provides an Intelligent Steering Agent (ISA) that controls and automates the QA process across large configuration spaces on a wide range of platforms by leveraging the extensive computing resources of worldwide open-source user communities in order to provide greater insight into the behavior and performance of fielded systems.

Building upon our successful R&D infrastructure outlined above, we are exploring techniques for (1) enhancing CoSMIC to create a new MIC tool that can ***model*** the distribution, concurrency, synchronization aspects of our CIAO QoS-enabled component middleware and then automatically ***optimize*** and ***generate*** platform-specific code that is customized for specific DRE application functional and QoS constraints on a range of platforms and then (2) ***validating*** the generated middleware configurations using the Skoll feedback-driven distributed continuous QA process guided by in-the-field data. We are applying and validating our model-integrated middleware using an avionics navigation application that we have developed in conjunction with Boeing Phantom Works in St. Louis, MO. This DRE application is based on CIAO and contains QoS-enabled software components that manage GPS-based location information to maintain a direction heading through a designated of waypoints. The components and the CIAO middleware are responsible for adapting to changes in the initially calculated route due to minor deviations caused by external factors, such as wind and sensor errors.

4 Concluding Remarks

The fundamental hypothesis behind our research is that changes in today's computing environment (e.g., high connectivity, substantial computing power for the average user, higher demand for and expectation of frequent software updates) are changing the software we are asked to design, the processes we use to design it, and the economic and technical environment for which we design. We believe that the characteristics of these changes can be leveraged to improve software quality and performance by providing rich and rapid feedback about design choices. In particular, we are showing how software engineers can shift substantial portions of their design, analysis, and measurement activities to actual user environments, so as to leverage in-the-field computational power, human resources, and actual user data to investigate the behavior of their systems after deployment and to improve their design, quality and performance. Crucial to the success of our approach is seamless integration with other advanced software design paradigms, such as QoS-enabled component middleware and model-integrated computing.