Applying System Execution Modeling Tools to Evaluate Enterprise Distributed Real-time and Embedded System QoS

John M. Slaby
Raytheon
Portsmouth, RI, USA
john_m_slaby@raytheon.com

Steve Baker
Raytheon
Portsmouth, RI, USA
steven_d_baker@raytheon.com

James Hill
Vanderbilt University
Nashville, TN, USA
j.hill@vanderbilt.edu

Douglas C. Schmidt
Vanderbilt University
Nashville, TN, USA
d.schmidt@vanderbilt.edu

Abstract

Component middleware is popular for enterprise distributed systems because it provides effective reuse of the core intellectual property (i.e., the "business logic"). Component-based enterprise distributed real-time and embedded (DRE) systems, however, incur new integration problems associated with component configuration and deployment. New research is therefore needed to minimize the gap between the development and deployment/configuration of components, so that deployment and configuration strategies can be evaluated well before system integration. This paper uses an industrial case study from the domain of shipboard computing to show how system execution modeling tools can provide software and system engineers with quantitative estimates of system bottlenecks and performance characteristics to help evaluate the performance of component-based enterprise DRE systems and reduce time/effort in the integration phase. The results from our case study show the benefits of system execution modeling tools and pinpoint where more work is needed.

1. Introduction

Integration challenges of component-based enterprise DRE systems. Enterprise DRE systems are increasingly developed using applications composed of distributed components running on feature–rich middleware frameworks. The distributed components are designed to provide reusable services to a range of application domains, which are then composed into domain-specific assemblies for application (re)use. Examples of component middleware platforms include Enterprise Java Beans and the CORBA Component Model (CCM).

The transition to component middleware is occurring in *enterprise business systems* to address problems of inflexibility and reinvention of core capabilities associated with prior monolithic, functionally-designed, and "stovepiped" legacy applications. Legacy applications were developed with the precise capabilities required for a specific set of requirements and operating conditions. Component-based systems, however, are designed to have a range of capabilities that enable their reuse in other contexts. Moreover, these systems are developed in layers, e.g., layer(s) of infrastructure middleware services (such as naming and discovery, event and notification, security and fault tolerance) and layer(s) of application components that use these services in different compositions.

Certain types of component middleware, such as Real-time CCM [14], are also being applied to the domain of *enterprise distributed real-time and embedded (DRE) systems*, such as shipboard computing environments and supervisory control and data acquisition systems, to provide users with quality of service (QoS) support to process the right data in the right place at the right time over a grid of computers. Some QoS properties required by enterprise DRE systems include the low latency and jitter as expected in conventional real-time and embedded systems, and high throughput, scalability, and reliability as expected in conventional enterprise distributed systems. Achieving this combination of QoS capabilities in enterprise DRE systems developed using component middleware is hard.

Component middleware can also complicate software lifecycle processes by shifting responsibility from software development engineers to software configuration/deployment engineers and systems engineers. Software development engineers traditionally created entire applications in-house using top-down design methods that could be evaluated throughout the lifecycle. In contrast, software configuration and deployment engineers and system engineers today assemble enterprise DRE systems by composing reusable components, whose combined properties are usually evaluated only during the integration phase. Unfortunately, problems uncovered during integration are much more costly to fix than if they were discovered earlier in the lifecycle. A key research challenge is thus exposing these types of issues (which often have dependencies on components that are not available until late in development) earlier in the lifecycle, e.g., prior to the system integration phase.

Component-based enterprise DRE systems use design- and run-time configuration steps to customize the behavior of reusable components to meet QoS requirements in the context where they execute. Finding the right configurations for components to meet application QoS requirements is hard. For example, tuning the concurrency configuration of a shipboard computing system to support both real-time and fault-tolerant QoS involves tradeoffs that challenge even experienced engineers. Moreover, application functionality is distributed over many components in a DRE system and developers must interconnect their components correctly and efficiently. This process can be tedious and error-prone using conventional handcrafted configuration processes.

The components assembled into an application must also be deployed on the appropriate nodes in an enterprise DRE system. Deployment is hard since host and network characteristics can vary *statically* (e.g., due to different hardware/software platforms used in a product-line architecture) and *dynamically* (e.g., due to damage/faults, change in computing objectives, or differences in the real vs. expected behavior of applications during actual operation). Evaluating the characteristics of system deployments is therefore tedious and error-prone when deployments are performed manually.

Another complexity of evaluating deployments of component-based enterprise DRE systems stems from applications sharing components with differing QoS requirements, such as a system resource manager that processes requests from high-priority tactical applications and low-priority desktop applications. It is hard to assure that a stand-alone application can meet stringent QoS requirements using dedicated resources. It is harder to assure these requirements with components that share resources with other applications.

Solution approach → System execution modeling tools. Despite the flexibility offered by component middleware, there are often surprisingly few configurations and deployments that can satisfy the functional and QoS requirements of an enterprise DRE system. We have therefore developed a *system execution modeling* tool chain called the *Component Workload Emulator (CoWorkEr) Utilization Test Suite (CUTS)*, which combines QoS-enabled component middleware and model-driven engineering (MDE) technologies. Software architects, developers, and systems engineers can use CUTS to explore design alternatives from multiple computational and valuation perspectives at multiple lifecycle phases using multiple quality criteria with multiple stakeholders and suppliers. In addition to validating design rules and checking for design conformance, CUTS facilitates "what if" analysis of alternative designs 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, and the ability of alternative system configurations and deployments to meet end-to-end QoS requirements for a particular workload.

In the context of enterprise DRE systems, our CUTS system execution modeling tool helps developers discover, measure, and rectify performance problems *early* in the lifecycle (e.g., in the architecture and design phases), as opposed to the integration phase, when mistakes are much harder and more costly to fix. This paper shows how we used CUTS to rapidly emulate component-based applications in an shipboard computing enterprise DRE system and then perform experiments that systematically estimated and evaluated the end-to-end QoS for key scenarios in this system.

Paper organization. This paper is organized as follows: Section 2 summaries limitations with prior work on QoS-enabled component middleware and MDE tools in the context of a shipboard computing system case study; Section 3 describes CUTS, shows how it overcomes limitations with prior work, and explains how we resolved key design challenges when developing CUTS; Section 4 shows how we applied CUTS to evaluate the QoS of various deployments in our case study; Section 5 compares our R&D efforts with related work; and Section 6 presents concluding remarks and lessons learned.

2. Background and Case Study

Our work on CUTS has evolved incrementally over the past three years in the context of a multi-phase program that is developing multi-layer resource management (MLRM) services to support product-lines that coordinate a grid of computers to manage many aspects of a ship's power, navigation, command and control, and tactical operations [15]. The MLRM services have hundreds of different types and instances of infrastructure components written in ~500,000 lines of Java and C++ code and ~1,000 files developed by six teams at different geographic locations. This section uses our experience to motivate the need for the CUTS system execution modeling tools.

Our initial approach. To address the configuration and deployment problems common to integrating components in enterprise DRE systems, our initial work combined QoS-enabled component middleware platforms with MDE tools. QoS-enabled component middleware supports the provisioning of key QoS properties, e.g., (pre)allocating CPU resources, reserving network bandwidth/connections, and monitoring/enforcing the proper use of DRE system resources at runtime, to meet end-to-end requirements. MDE tools combine

- *Domain-specific modeling languages* (DSMLs), which provide programming notations that formalize the process of specifying application logic and QoS-related requirements using type systems that precisely express key characteristics and constraints associated with DSMLs for particular application domains and
- *Model transformations and code generation*, which automate and ensure the consistency of software implementations via analysis information associated with functional and QoS requirements captured by models of domain-specific structure and behavior.

In prior work with colleagues at Washington University, St. Louis we developed a QoS-enabled component middleware platform called the *Component-Integrated*

ACE ORB (CIAO) [14] that combines Lightweight CCM [4] capabilities (such as standards for specifying, implementing, packaging, assembling, and deploying components) with Real-time CORBA [12] features (such as thread pools and priority preservation policies) to create a Real-time CCM middleware platform. Likewise, we created an MDE tool suite called *Component Synthesis using Model Integrated Computing* (CoSMIC) [7], which is an integrated set of DSMLs that support the development, deployment, configuration, and evaluation of enterprise DRE systems based on Real-time CCM. CoSMIC is implemented using the *Generic Modeling Environment (GME)* [9], which is an open-source MDE toolkit for creating and using DSMLs. These tools/platforms are open-source and available from www.dre.vanderbilt.edu.

By combining CIAO and CoSMIC, we tackled many integration challenges associated with configuring and deploying enterprise DRE systems by leveraging MDE tools to enforce correct-by-construction design. For example, we used CoSMIC's model interpreters to generate Real-time CCM XML configuration files [1] and CIAO's *Deployment And Configuration Engine* (DAnCE) [5] to -deploy the resulting component assemblies on DRE system nodes, as shown in Figure 1.

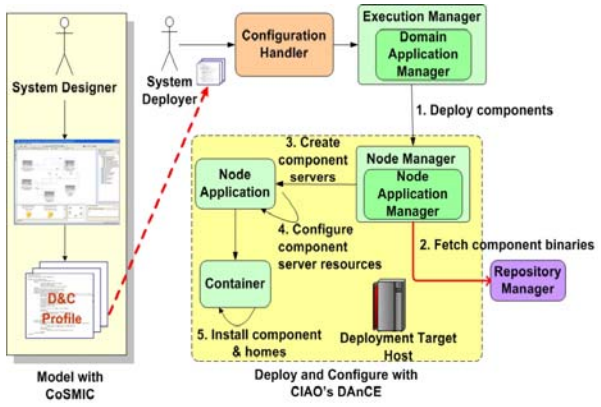

Figure 1. Integrating CIAO, DAnCE, and CoSMIC

Limitations with our initial approach and common alternatives. To evaluate the benefits of combining CIAO, DAnCE, and CoSMIC, we applied them in phase one of our MLRM project [15]. Our experience, however, indicated that CIAO, DAnCE, and CoSMIC were insufficient to evaluate the QoS of applications in enterprise DRE systems due to the following limitations:

- **Insufficient performance evaluation.** In the MLRM environment, many different applications ran concurrently across networks that included both shared and dedicated components. CIAO, DAnCE, and CoSMIC, however, provided insufficient support for evaluating QoS-related characteristics (such as communication delay, temporal phasing, parallel execution, and synchronization).
- **Serialized phase ordering dependencies.** Application components that exercised the MLRM infrastructure middleware services were not developed until later in the system lifecycle. The QoS of the infrastructure services therefore was not evaluated adequately under realistic workloads to validate their architecture and design.

We initially considered evaluating MLRM QoS characteristics via simulation. Due to size, interdependencies, and the sheer number of variables involved it was impractical to develop and evolve realistic models that simulate complex scenarios. Moreover, while pure simulation can provide valuable information about system QoS behavior, it is hard to leverage simulation results *directly* in the production operational environment.

3. The Component Workload Emulator (CoWorkEr) Utilization Test Suite (CUTS)

To overcome the limitations described in Section 2, we needed more effective technologies to evaluate the end-to-end QOS characteristics of MLRM applications in a production-scale environment, even before any actual application components were developed. Our goals were motivated by our experience in phase one of the MLRM project and involved:

- *Not* obtaining 100% precision, but providing systems engineers and architects with rapid, reasonably accurate estimates of system QoS early in the lifecycle.
- Improving the accuracy of our estimates of system QoS incrementally as our understanding of application requirements, implementations, and execution environments increased.
- Automatically transitioning select artifacts used in our evaluations (such as models of deployment plans that met end-to-end QoS requirements) to the component-based application and middleware deployments and configurations we were creating.

To meet our goals and overcome limitations with prior work, we developed the *Component Workload Emulator (CoWorkEr) Utilization Test Suite (CUTS)*. CUTS is a system execution modeling tool chain for creating component-based applications rapidly and performing experiments that systematically evaluate interactions that are hard to simulate. In particular, CUTS provides model-based workload generation, data reduction, and visualization tools to construct experiments rapidly and analyze results from alternate execution architectures. CUTS can also import measured performance data from *faux* application components running over *actual* infrastructure middleware services to estimate enterprise DRE system behavior in a realistic environment.

When combined with our prior work on QoS-enabled component middleware and MDE tools, CUTS allowed more robust and complete solutions for emulating actual application components and evaluating QoS earlier in the enterprise DRE system lifecycle. For example, we used CoSMIC to create models of DRE systems composed of faux application components and actual system infrastructure components. We then used these models with DAnCE to deploy these components into a representative testbed (www.dre.vanderbilt.edu/ISISlab) and conduct systematic experiments that measured how well the system performed relative to QoS specifications from production computing systems. This remainder of this section presents the CUTS architecture and solutions to design challenges we faced when developing it and applying it to the MLRM case study.

3.1 CUTS Architecture

As outlined in Section 2, CUTS is a system execution modeling toolkit that (1) emulates portions of enterprise DRE systems (2) collects performance data provided by the emulation, and (3) analyzes the data to estimate system QoS and pinpoint performance bottlenecks. At the heart of CUTS is an assembly of CCM components, called a *CoWorkEr* (Figure 2). A CoWorkEr is a *faux* component that can be programmed rapidly to emulate the expected behavior and resource consumption of its counterpart in the production application.

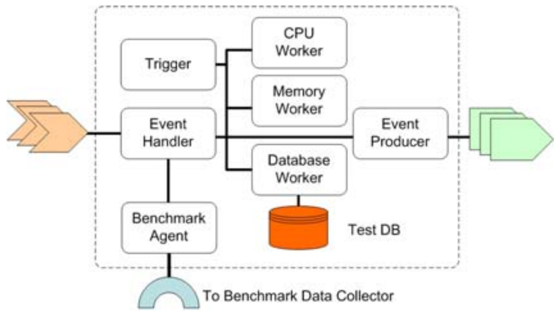

Figure 2. A CoWorkEr Component Assembly

CoWorkErs can be connected together via their exposed ports to create *operational strings*, which are task graphs that capture the partial ordering of a set of executing software components. Figure 2 shows the key elements of the CoWorkErs, which fall into two broad categories: *workload generation* and *test control and analysis*.

3.1.1. Workload generation is implemented in CUTS as an assembly-based CCM component composed of the following monolithic CCM components:

- The *EventHandler* can receive user-defined events. It records the number of events received for each type and performance metrics regarding the delay between original publication and the onset of processing. The *EventHandler* also tracks the time required to process each event it receives. Workloads, which are performed by the *worker* components described next, may also be associated with receiving combinations and numbers of events.
- The *CPUWorker* performs CPU operations. As with all *workers*, the quantity of work to perform is specified as a number of *repetitions*, which represent an abstract unit of work.
- The *MemoryWorker* performs allocation and deallocation of memory.
- The *DatabaseWorker* performs a series of insert, update, and delete operations on a specified database.
- The *EventProducer* (which is also a *worker*) publishes events that carry a data payload of the desired size. Events are time-stamped prior to transmission.
- The *Trigger* is provided to represent external input to a simulated application, or regularly scheduled, time-driven processing not resulting from the receipt of an event. *Triggers* provide both periodic and pseudo-random behavior by inducing *workers* to perform a *workload* at a specified interval and probability of occurrence. A *Trigger* can also perform *startup workload* during activation.

To simplify the programming and configuration of CoWorkErs, we created an MDE-based DSML called the *Workload Modeling Language (WML)* [15]. WML is used to characterize the behavior of individual CoWorkErs by specifying their processor, memory, database, and input/output usage profiles. XML characterization files are then generated from a WML model, and subsequently parsed by *EventHandler* and *Trigger* components to dictate the behavior of their respective CoWorkEr.

3.1.2. Test control and analysis in CUTS includes the following elements:

- The *BenchmarkAgent* completes the CoWorkEr assembly shown in Figure 2. It requests test data collected by *EventHandlers* at a user-defined interval and transmits this data to the *BenchmarkDataCollector*.
- The *BenchmarkDataCollector* (BDC) submits test data to an in-memory *BenchmarkDatabase*.
- The *BenchmarkManagerWeb-interface* (BMW) implements the test control and analysis functionality via an ASP.NET application. This manager processes data captured in the *BenchmarkDatabase* and invokes DAnCE's *ExecutionManager* to start and end the deployment of test assemblies. In addition to the web browser interface, the BMW provides a web-services interface that allows any programming language that supports the Simple Object Access Protocol (SOAP) to automate CUTS tests.

Figure 3 shows how CUTS can evaluate the QoS of enterprise DRE systems. Dedicated hosts, called *test host*, run inside the *test network* and the *BenchmarkDataCollector* and *BenchmarkManagerWeb-interface* exist outside the *test network*. This setup limits outside interference on tests run using CUTS while permitting users to analyze their results either during or after the test run.

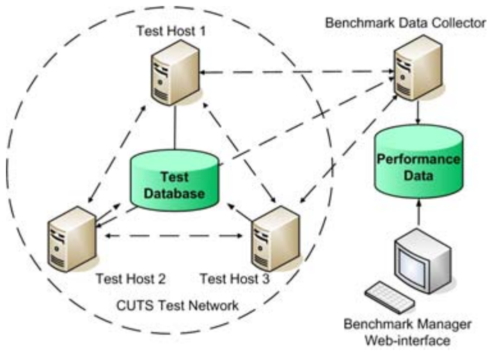

Figure 3. Example Setup of CUTS to Evaluate QoS in an Enterprise DRE System

3.2 CUTS Design Challenges and Solutions

We now describe solutions to key problems encountered when developing and applying CUTS.

Challenge 1. Non-intrusive metrics collection. An *ad hoc* metrics collection system might interfere with the emulation and skew test results. Metric collection should therefore have minimally intrusion and resource usage.

Solution → Decouple metrics collection from emulation, and collect metrics using a 3-phase data acquisition process. The components described in Section 3.1 work together to collect performance metrics in three separate stages. In stage 1, the *EventHandler* maintains for each event type a local in-memory record of the number received, the max/min transmission and processing time, and running totals for transmission and processing time. In state 2, the *BenchmarkAgent* obtains the data from the *EventHandler* at a user-specified interval in a dedicated thread, and resets the *EventHandler's* running totals. The *BenchmarkAgent* transmits the collected data to the *BenchmarkDataCollector*, which immediately queues the data and returns. In stage 3, the *BenchmarkDataCollector* dequeues the data and inserts it into a MySQL database. Each phase of the data acquisition process also uses a dedicated thread to minimize the impact of data collection on the emulation

All data stored and transmitted by the *EventHandler* and the *BenchmarkAgent* is a fixed-size to ensure memory usage is bounded by a constant factor. The aspects of metric collection that cause variable memory usage and delays, e.g., queuing and entry of data into a database, are placed the *BenchmarkDataCollector*, which is deployed on a node not used by a CoWorkEr. Moreover, separate networks can be used to decouple transmission of metric data from the transmission of CoWorkEr operations.

Challenge 2. Simplify characterization of application workload. Some CoWorkEr users will be systems engineers or architects, who may not be familiar with with third-generation languages, such as C++ or Java, or configuration languages, such as XML. It is therefore important for CUTS to offer alternatives to programmatic interfaces and configuration files for these types of users.

Solution → Provide graphical user interfaces for characterizing, deploying and analyzing applications. CUTS allows users to design simulated applications entirely through visual models. In particular, the CoSMIC and WML DSMLs allow users to create structural and behavioral models of their applications without manually editing configuration files or third-generation language code. Deployment and analysis of the application is provided through an intuitive *BenchmarkManagerWeb-interface*. More details and examples of WML appear in [15].

Challenge 3. Simplify Customization. CoWorkErs can emulate four categories of core application work (CPU, memory, database, and network resource utilization), but the need for more customized behavior may arise for particular types of enterprise DRE systems. The design of the CoWorkErs therefore needs to support user-defined extensions to its basic work repertoire.

Solution → Support custom CoWorkEr components. In the spirit of CCM, CoWorkErs employ a modular design where any monolithic components comprising the CoWorkEr assembly shown in Figure 2 can be replaced with a customized component that implements the same interface, without modification or recompilation of other components. For example, it is straightforward to replace the default *CPUWorker* with a *FCPUWorker* that only performs floating-point arithmetic. In addition, GME's convenient inheritance support makes swapping of components straightforward within a CoSMIC model,.

Challenge 4. Descriptive analysis of performance. If an emulation shows that a proposed configuration and deployment of enterprise DRE system components will not meet QoS expectations, CUTS users must be able to pinpoint the source of the problem quickly to correct it.

Solution → Present metrics in layers to support general and detailed analysis. In addition to providing a graphical representation of observed performance vs. deadlines along a critical path, CUTS *BenchmarkManagerWeb-interface* allows users to view statistics for individual CoWorkErs. A tabular display allows users to view summary statistics for operational strings of CoWorkErs simultaneously, whereas detailed graphs support scrutiny of an individual CoWorkEr's performance over time. Statistics for processing time can also be subdivided to reflect the four categories of work, thereby allowing analysts to determine whether QoS target requirements are missed due to reliance upon a sluggish database, paging due to excessive memory allocation, saturation of network bandwidth, etc. Usage and further discussion of these features can be found in Section 4.2.

4. Applying CUTS to Evaluate an Enterprise DRE System

This section describes the design and results of an experiment that uses the CUTS systems execution modeling toolchain to evaluate the QoS of a representative enterprise DRE system from the domain of shipboard computing. This experiment is based upon work conducted in the MLRM project described in Section 2. This project provided a representative case study for evaluating CUTS since it runs on general-purpose operating systems (such as Solaris and Linux) with real-time enhancements. It also uses a component-based architecture developed using the CIAO and DAnCE Real-time CCM middleware and CoSMIC MDE tools, has hundreds of components types/instances and hundreds of thousands of lines of C++ and Java code, and has been developed over the past three years by a group of geographically distributed teams. As a result, the MLRM software base incurs many of the same integration challenges associated with configuration, deployment, and QoS evaluation that occur in other production enterprise DRE systems.

4.1 The MLRM SLICE Experiment using CUTS

4.1.1. Experiment motivation. One of the challenging problems in the second phase of the MLRM project is called the *SLICE scenario*, which consists of 2 sensors, 2 planners, 1 configuration, 1 error recovery, and 2 effector components. The SLICE scenario requires the transmission of information detected by the sensors to each planner in sequence, then to the configuration component, and lastly to both effectors to perform actions that control devices in the physical world. Components in the SLICE scenario are deployed across 3 computing nodes because the workload generated by each component collectively is more than a single node can handle. The main sensor and effector (represented as sensor-1 and effector-1 in Figure 4 and in following sections) are deployed on separate nodes to reflect the placement of physical equipment in the production shipboard system. Figure 4 shows a model of the end-to-end layout of SLICE components, with the critical path specified by the dashed arrows.

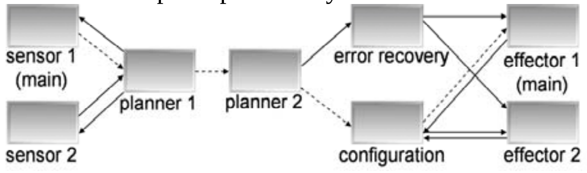

Figure 4. Model of SLICE Showing the Components and Their Interconnections

In phase two of the MLRM project, the multi-layer resource manager infrastructure was re-implemented to use Real-time CCM (via CIAO and DAnCE), and MDE tools (via CoSMIC), instead of Real-time CORBA and *ad hoc* deployment mechanisms used in phase one. Based on the MLRM phase two development schedule, the integration of components that implemented the SLICE scenario atop the new multi-layer resource management infrastructure was not slated to occur until 12 months into the program to provide sufficient time to finish developing, testing, and optimizing the multi-layer resource management infrastructure. The SLICE scenario, however, uses software components similar to product-lines and challenge problems in phase one of the MLRM project. We therefore already understood each component's behavior in SLICE, but did not know how overall performance of the SLICE scenario would be affected by the new MLRM infrastructure.

In phase one, we waited until the integration phase of our schedule to begin benchmarking the system, only to learn none of the QoS requirements were met due to improperly designed multi-layer resource management infrastructure. As a consequence, our schedule slipped and the process of reconfiguring and redeploying MLRM application and middleware components to meet QoS requirements required significant manual effort. To prevent the same problems from happening in phase two of the program, we used CUTS to evaluate the QoS challenges of the SLICE scenario *prior* to the integration phase. Our goal was to determine which configuration and deployment strategies will enable us to meet the QoS critical path deadline and create a pool of selectable deployment strategies that meet the performance requirements. The underlying hypothesis driving the experiment was much of the performance information could be collected prior to the integration phase by *emulating* key properties of the SLICE scenario components using CUTS. As a result, less time would be spent integrating and testing the actual SLICE components after they were completed.

4.1.2. Experiment design. For the SLICE scenario, there is a 350 ms QoS critical path deadline, which is representative of the end-to-end execution time of a similar scenario from phase one of the MLRM project. This deadline corresponds to receiving a command event on sensor-1 up to performing an action with effector-1. Sensor-1 and effector-1 must be deployed on separate nodes to meet the constraints discussed in section 4.1.1. Table 1 describes the predicted behavior for two of the SLICE components (which were defined using the *Workload Modeling Language*) to illustrate the various types of workload and actions for a CoWorkEr.

Planner -1 CoWorkEr	
Workload performed every second	publish command of size 24 bytes
Workload performed after receipt of a track event	alloc 30 KB; 55 dbase ops; 45 CPU ops; publish assessment of size 132 bytes; dealloc 30 KB
Configuration-Optimization CoWorkEr	
Workload performed at startup time	alloc 1 KB; 25 dbase ops; 1 CPU ops; 10 dbase ops; dealloc 1 KB
Workload performed after receipt of an assessment event	alloc 5 KB; 40 dbase ops; 1 CPU op; publish command of size 128 bytes; dealloc 5 KB
Workload performed after receipt of a status event	1 dbase op

Table 1. Expected Behavior for 2 SLICE CoWorkErs

The workload specifications for each component listed in Table 1 is based on the behavior of components implemented in phase one. We obtained these values by estimating the number and types of operations based our understanding gained by implementing and testing the functionality of the workload generators explained in Section 3.1.1. For the predicted behavior for the remaining components, please refer to [13].

Host	Operating System	Database
1	Fedora Core3	YES
2, 3, BDC	Fedora Core3	NO
BMW	Windows XP	YES

Table 2. System Characteristics for Experiment Host

Each host in the CUTS-based experiments was an IBM Blade Type L20, dual-CPU 2.8 GHz processor with 1 GB RAM with the characteristics listed in Table 2. The middleware was version 0.4.7 of CIAO/DAnCE, and the MDE tools used were version 0.4.6 of CoSMIC, which is the target middleware and MDE tool for the SLICE scenario in phase two. Each test was run for 10 minutes.

4.2 Viewing and Interpreting the Results of the SLICE Experiment

This section describes the results of tests that used CUTS to evaluate various deployments of SLICE components onto hosts to (1) test the capability of CUTS, (2) determine which deployment strategies meet the 350 ms critical path deadline when components *Sensor-1* and *Effector-1* are deployed on separate nodes, and (3) prove that workload generated by SLICE is too much when the critical path components are deployed on a single node. The first listing in Table 3 contains the legend for the CoWorkEr symbols used in the second listing.

SLICE CoWorkEr Legend for Test Table			
Symbol	**CoWorkEr**	**Symbol**	**CoWorkEr**
A	Sensor-1 *	E	Config-Op *
B	Sensor-2	F	Error-Recovery
C	Planner-2 *	G	Effector-1 *
D	Planner-1 *	H	Effector-2
** represents CoWorkEr in the critical path*			

Test	Deployment Strategy			Critical Path Execution Time (avg./worse) (ms)
	Host 1	Host 2	Host 3	
1	C,D,E,F	A,B	G,H	411 / 1,028
2	A,B,C,D	F	E,G,H	420 / 1,094
3	A,B,C,D,E	F	G,H	416,/ 1,085
4	A,B,C,D,E,F,G,H			463 / 1,247
5	A,B,C,D,E,G,H	F		467 / 1,219
6	A,C,D,E,G	F	B,H	323 / 844
7	A,G	C,D,E	B,F,H	363 / 887
8	D	A,B,C, F,G,H	E	405 / 975
9	A,D	C,E,G	B,F,H	235 / 387
10	A,D	E,G	B,C,F,H	251 / 395
11	A,D,E	C,G	B,F,H	221 / 343

Table 3: SLICE Results for Experiments using Different Deployment Strategies in CUTS

4.2.1. Discussion of the hypothesis. Test 4 and 5 were two tests that not only missed the 350 ms deadline, they incurred the worst critical path execution time for all 11 tests. The main purpose of test 4 and 5 was to evaluate our hypothesis that the 350 ms deadline could not be met if all components were deployed on the same node. After completing test 4 and 5, we validated this hypothesis – the workload generated by components in the critical path is more than a single node can handle, so they must be deployed across multiple nodes. On the other hand, test 6 deployed only the components in the critical path on the same node, and had an average execution time of 323 ms. CUTS therefore enabled us to learn that we could meet the 350 ms deadline if only the critical path components were deployed on the same node.

Workload	Avg. Samples	Avg./Rep (ms)	Avg. Time (ms)
Transmit Delay	5		6.19
Total Workload	5		169.6
CPU		2.20859	99.39
Memory		0.00727	0.51
Publication		1.40206	1.4

Table 4. Snapshot of Timing Data for Sensor-1 in Test 8 obtained from the BMW Test Results Page

CoWorkEr	Transmission Delay (ms)	Avg. Time of Completion (ms)
Sensor-1	6.19	169.6
Planner-1	12.11	54.03
Planner-2	10.69	110.66
Config-Op	17.04	23.84
Effector-1		0.34

Table 5. Snapshot of the Critical Path Timing Data for Test 8 from the BMW Analysis Page

4.2.2. Interpreting the CUTS benchmark data results. Running 11 tests with various deployment strategies provided key information about the current MLRM infrastructure. Of the 11 tests, only 3 deployed the critical path components across multiple nodes and completed their end-to-end execution in 350 ms. Of these 3 tests, 2 deployed the critical path across all three nodes and completed it within an average time of 350 ms, and 1 test (test 11) completed it within a worse time of 350 ms. Although we did not exhaust all possible deployment strategies in this experiment, we learned that only 18% (2 out of 11) of the current test passed on their planned infrastructure while meeting the deployment requirements and test 11 yield the best performance.

After running test 1 through 8, only 1 test met the 350 ms end-to-end deadline, and 7 of the tests had faults in their deployment specification, e.g., placing a CoWorkEr on a host with insufficient resources to handle its workload without missing deadlines. We used CUTS graphical analysis features to investigate why these deployment strategies did not meet their QoS requirements.

Tables 4 and 5 show the results provided via the *BenchmarkManagerWeb-interface* (BMW) for test 8, which measures the behavior when two components in the critical path handling the most workload are deployed on their own node. Table 4 shows the time to transmit a message between two CoWorkErs and how long it took to complete each type of workload – CPU, database, or memory – for *Sensor-1*. For the CoWorkErs in the critical path in test 8, it took 169.6 ms for *Sensor-1* to process its workload after receipt of a *command* event from *Planner-1*; 54.0 ms for *Planner-1 to* perform its workload after receipt of a *track* event from *Sensor-1* or *Sensor-2*; and 110.6 ms for *Planner-2* to perform its workload after receipt of a *command* event from *Planner-1*.

For test 8, *Sensor-1* and *Planner-2* have the longest completion times. Based on the quantitative analysis provided by CUTS, we realized that the *Sensor-1* and *Planner-2* CoWorkEr components had a heavier workload than expected, and must be deployed on separate nodes. We then used CoSMIC and DAnCE to place the *Sensor-1* and *Planner-2* CoWokErs on different hosts, which created the deployment strategies used in test 9, 10 and 11, all of which met the 350 ms deadline. Of those 3 tests, test 11 was the best test case and was the only test to have a worse execution time that meets the 350 ms deadline. In addition, these deployment strategies meet the deployment requirements of placing *Sensor-1* and *Effector-1* on different nodes, as discussed in Section 4.1.4.

More detailed examples of the types of visualizations and analysis provided by CUTS is presented in [13].

5. Related Work

Distrubuted system emulation environments. Various environments can be used to emulate and evaluate distributed system behavior. A popular environment is Emulab [6], which provides tools that can be used to configure the topology of experiments, e.g., by modeling the underlying communication links. This topology is mapped to ~250 physical nodes that can be accessed via the Internet. CUTS enhances the Emulab network-centric focus via the WML and CoSMIC DSMLs that create tests and deployment/configuration specifications at a high-level of abstraction that is more suitable for emulating component-based DRE systems than the NS scripts provided by Emulab to provision communication links.

ModelNet [17] is another environment for evaluating large-scaled distributed systems. In ModelNet, developers emulate multiple clients and hosts using one host. For example, 100 Gnutella clients each with a 1 Mbps bottleneck bandwidth can be emulated on one dual processor-1 GHz machine. ModelNet also facilitates the emulation of faux and real applications. The CUTS emulation environment is similar to the ModelNet environment in that both address large-scaled distributed systems. CUTS, however, focuses on DRE systems and uses the target architecture to facilitate emulation and performance accuracy. Whereas, ModelNet seeks to provide scalable and accurate solutions using as few hosts as possible.

System execution modeling tools. KLAPER [8] is a modeling language that specifies system behavior for component-based systems. Similar to WML in CUTS, KLAPER specifies workload, such as resource utilization, but does not capture handling of events. WML extends KLAPER by allowing sequential specification of resource utilization, transmission and receipt of events, and workload types, e.g. event, periodic, or startup.

UPPAAL [3] is a system execution modeling tool that verifies properties of DRE systems via a modeling-language and environment for verifying a system's specified behavior early in the development stage. by dynamically validates all possible behaviors with its model-checking simulator. CUTS focuses on complementary areas, such as (1) emulating system behavior on the target platforms, (2) benchmarking DRE systems as a whole and as individual components, and (3) monitoring system flows to verify QoS requirements are met.

RT-UML [11] models and evaluates the performance of component-based systems by defining services and QoS policies for components, though modeling system behavior is future work. RT-UML is also designed to be supported by external *simulation* tools, which are still under development. WML enhances RT-UML by providing a working DSML tool that allows developers to specify a component-based system behavior, which is then *emulated* by CUTS.

Evaluation techniques for component architectures. [16] discusses a technique called *trace-based analysis* for Enterprise Java Bean (EJB) components. In trace-based analysis, different execution traces in a component are monitored and dumped to a trace file contained on the host. After the emulation, the trace files are parsed and combined with the deployment descriptors, which define the structure of the system, to determine the different paths of execution in the system. CUTS is similar to trace-based analysis since it collects traces of execution times, but these traces are logged to a central database. CUTS also monitors predetermined execution paths in real-time, whereas [16] uses methods to reconstruct every path autonomously, but does not monitor performance in real-time over the duration of the emulation. [16] also focuses on service calls, whereas CUTS performance metrics use *events* sent between components.

[18] and [19] discuss *vertical profiling* evaluation techniques in the context of EJB. In vertical profiling, performance metrics based on the types of operations and actions (e.g., cache misses and CPU cycles) are collected in trace files across multiple executions of the same tests. The trace files are then fused through a process called

trace-alignment using a common metric that occurs in the source traces. After the traces are aligned, *correlation analysis* is applied to the traces to help determine what other metrics collected in the trace may influence its behavior. [19] also discusses how to automate this process. CUTS provides a similar approach in the context of CCM that allows analysis of individual actions and operations in a component. CUTS, however, goes further and allows the analysis to happen at real-time with the emulation.

Architectures for deployment and configuration of components. Proactive [2] is a framework for component deployment and configuration designed for conventional Java applications running on JVMs. In contrast, CUTS leverages DAnCE [5], which is targeted for deploying and configuration components in DRE systems.

The Globus Toolkit [10], which is part of the Open Grid Standard Architecture (OGSA), is another framework that handles deployment and configuration of components for Grid computing. Unlike DAnCE, however, Globus does not provide DSMLs for modeling various concerns of enterprise DRE systems and validating systems before deploying them. Lastly, the DAnCE framework conforms to the OMG D&C standards, which allows it to leverage other efforts based on the OMG D&C specifications, such as OpenCCM and MICO-CCM.

6. Concluding Remarks

This paper described the *Component Workload Emulator (CoWorkEr) Utilization Test Suite* (CUTS). CUTS is a system execution modeling toolchain that simplifies the creation of – and experimentation with –emulations of applications that help evaluate the QoS of component-based enterprise DRE systems. We also described the design and implementation of CUTS, along with the challenges we encountered and solutions we applied.

Our experience applying CUTS to the SLICE scenario in phase two showed how systems execution modeling tools can decrease the time spent resolving integration problems. Instead of waiting until full system integration, CUTS allowed us to test deployments of the MLRM infrastructure in the actual target environment using emulated application components. When combined with other QoS-enabled component middleware and MDE tools, alternative deployment plans could be evaluated rapidly earlier in the lifecycle, thereby reducing the time and effort spent in integration. Although phase two is still ongoing, CUTS has already saved significant amounts of time and effort compared to phase one.

The following summarizes the benefits of applying CUTS based on our experience thus far:

- CUTS allowed us to emulate system components using the target hardware and software infrastructure, instead of waiting until completely implementing the real components and trying to resolve all issues during integration phase, as we had attempted to do (rather unsuccessfully) in phase one.
- CUTS allowed us to rapidly create and quantitatively evaluate a range of deployment plans to see how they impacted end-to-end QoS behavior. Much more time and effort would have been required if these tests were conducted manually, i.e., without the visual MDE functionality and automation provided by CUTS and the underlying CoSMIC MDE tools and CIAO/DAnCE middleware.
- CUTS provided qualitative performance analysis to assist in locating deficiencies in current deployments so we can determine alternative deployments that meet end-to-end QoS requirements more effectively.
- The use of MDE tools enabled CUTS to substitute real components for the emulated ones quickly, so we can incrementally evaluate QoS performance with more realistic workloads as knowledge of the application and system infrastructure evolves.

Although using CUTS in phase two provided the benefits outlined above, we also discovered that the following work is needed to improve the evaluation of QoS in component-based enterprise DRE systems:

- There were test cases in the empirical results in Section 4.2 where the critical path deadline was missed significantly. After further analyzing these results, specifically after test 8, we realized that messages not on the critical path were handled at the same priority as arbitrary messages in the system. We therefore need to extend CUTS to allow QoS specifications for the various components of a CoWorkEr.
- CoWorkErs currently generate a pre-defined set of events, which is representative of a certain class of statically provisioned DRE systems. Enterprise DRE systems, however, often must adapt to changes in the environment. We therefore need to extend CUTS and WML to permit specification and enforcement of adaptive behavior for QoS evaluation.
- CUTS uses XML specifications to configure the behavior of generic CoWorkErs, whose internals and -interfaces do not resemble the components they emulate. We are therefore extending WML to generate proxy CoWorkErs that simplify the interchange of -emulated with production application components. This enhancement will also enable the collection of performance metrics from actual and emulated components to evaluate their similarities and differences.
- Enterprise DRE systems can share resources either locally or remotely, which affects QoS performance of the system. Further work is therefore needed to extend CUTS to allow CoWorkErs to share resources both remotely and locally for QoS performance evaluation.
- QoS does not always depend on behavior at the application level. In many instances, QoS can depend-

ent on performance metrics at the different layers of middleware below the application, and the machine, e.g., CPU operations and cache misses. CUTS therefore needs to be extended to monitor performance metrics at all levels in an application and apply QoS requirements to these metrics.

- Derivation of workloads in the SLICE scenario required us to estimate each component's workload based on our understanding of performance characteristics of similar components from phase one. This process is labor intensive and faulty if the characteristics are misinterpreted. Since CUTS relies on "trial and error" methods we are developing heuristics that will automatically derive workloads using the workload heuristics and performance characteristics.

CUTS is currently being transitioned from the MLRM project to a production shipbuilding program to assist system engineers and architects in evaluating QoS performance metrics of DRE systems. Our future R&D efforts will therefore focus on adding the capabilities listed above to further enhance CUTS and provide system architects and engineers with a stronger tool suite. An open-source version of CUTS and the other MDE tools and middleware platforms described in this paper can be downloaded from www.dre.vanderbilt.edu/CoSMIC.

References

[1] Balasubramanian, K., Balasubramanian, J., Parsons, J., Gokhale, A. and Schmidt, D., "A Platform-independent Component Modeling Language for Distributed Real-time and Embedded Systems," *Proceedings of the 11th IEEE Real-Time and Embedded Technology and Applications Sym.* San Francisco, CA, Mar 2005.

[2] Baude, F., Caromel, D., Huet, F., Mestre, L., and Vayssiere, J."Interactive and Descriptor-based Deployment of Object-Oriented Grid Applications," *Proceedings of the 11th International Symposium on High Performance Distributed Computing*, Edinburgh, UK, Jul 2002.

[3] Bengtsson, J. Larsen, K., Larsson, F., Pettersson, P., and Yi, W., "UPPAAL: A Tool Suite for Automatic Verification of Real-time Systems," *Proceedings of Workshop on Verification and Control of Hybrid Systems III, 1066*, 232 – 243, Oct 1995.

[4] Object Group Management, "Light Weight CORBA Component Model Revised Submission," Ed. OMG Document realtime/03-05-05, May 2003.

[5] Deng, G., Balasubramanian, J., and Otte, W., Schmidt, D. and Gokhale, A., "DAnCE: A QoS-enabled Component Deployment and Conguration Engine," *Proceedings of the 3rd Working Conference on Component Deployment*. Grenoble, France, Nov 2005.

[6] Ricci, R., Alfred, C., and Lepreau , J., "A Solver for the Network Testbed Mapping Problem," *SIGCOMM Computer Communications Review, 33,* Apr 2003.

[7] Gokhale, A., Balasubramanian, K., Balasubramanian, J., Krishna, A., Edwards, G., Deng, G., Turkay, E., Parsons, J. , and Schmidt, D. "Model Driven Middleware: A New Paradigm for Deploying and Provisioning Distributed Real-time and Embedded Applications," *The Journal of Science of Computer Programming: Special Issue on Model Driven Architecture,* 2006 (to appear).

[8] Grassi, V., Mirandola, R., and Sabetta, A., "From Design to Analysis Models: A Kernel Language for Performance and Reliability Analysis of Component-based Systems," *Fifth International Workshop on Software and Performance,* Palma de Mallorca, Spain, Jul 2005.

[9] Karsai, G., Sztipanovits, J., Ledeczi, A. and Bapty, T. "Model-Integrated Development of Embedded Software," *Proceedings of the IEEE,* 145-164, Jan 2003.

[10] Lacour, S., Perez, C., and Priol, T., "Deploying CORBA Components on a Computational Grid: General Principles and Early Experiments using the Globus Toolkit," *Proceedings of the 2nd International Working Conference on Component Deployment (CD 2004)*. Edinburgh, UK, May 2004.

[11] Bertolino, A. and Mirandola, R., "Software Performance Engineering of Component-based Systems," *Proceedings of the 4th International Workshop on Software and Performance*, Jan 2004.

[12] Object Management Group, "Real-time CORBA Specification," OMG Document formal/02-08-02, Jul 2002.

[13] Slaby J., Baker, S. Hill, J., and Schmidt, D. "Defining Behavior and Evaluating QoS Performance of the SLICE Scenario," ISIS Technical Report (ISIS-05-608), Vanderbilt University, Nashville, TN, Dec 2005. www.dre.vanderbilt.edu/~schmidt/SLICE-TR.pdf.

[14] Wang, N. and Gill, C., "Improving Real-time System Configuration via a QoS-aware CORBA Component Model," *Hawaii International Conference on System Sciences, Software Technology Track, Distributed Object and Component-based Software Systems,* Jan 2003.

[15] 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.

[16] Mania, D., Murphy, J. and McManis, J., *"Developing Performance Models from Non-intrusive Monitoring Traces,"* Proceeding of Information Technology and Telecommunications (IT&T), Oct 2002.

[17] Vahdat, A., Yocum, K., Walsh, K., Mahadevan, P., Kostic, K., Chase, J., and Becker, D., "Scalability and Accuracy in a Large-Scale Network Emulator," *Proceedings of 5th Symposium on Operating Systems Design and Implementation (OSDI),* Dec 2002.

[18] Hauswirth, M., Sweeney, P., Diwan, A., and Hind, M., *"Vertical Profiling: Understanding the Behavior of Object-Oriented Applications,"* 18th Conference of Object Oriented Programming, Systems, Languages and Applications, Oct 2004.

[19] Hauswirth, M., Diwan, A., Sweeney, P and Mozer, M., *"Automating Vertical Profiling,"* 19th Conference of Object Oriented Programming, Systems, Languages and Applications, Oct 2005.