Towards Reducing the Complexity of Adaptive Real-Time Large-Scale Distributed Embedded Systems

Lisa DiPippo, Jiangyin Zhang, Matthew Murphy, and Victor Fay Wolfe
University of Rhode Island
{lastname}@cs.uri.edu

Joseph Loyall, Richard Schantz, and Craig Rodrigues
BBN Technologies
{jloyall,schantz,crodrigu}@bbn.com

Jeff Parsons, Sandeep Neema, Balachandran Natarajan, and Aniruddha Gokhale
Vanderbilt University

This paper describes elements of the approach that we are taking to address the complexity inherent in creating software for large scale distributed real-time embedded (LDRE) applications such as the control of total ship computing on the new US Navy surface ships (DDX), coordinated unmanned vehicles, meteorological measurement and prediction systems, and widely distributed automated financial control. These applications are functionally complex, and their complexity is further amplified due to non-functional considerations such as:

- their large scale
- their potentially wide distribution
- their real-time requirements
- their dynamically changing environment

In general, our approach is to provide more of the solution to the non-functional aspects commonly available off-the-shelf through a combination of advanced middleware services and advanced software engineering approaches, which when combined would provide for developing these systems from a much higher level (and presumably less costly and error prone) basis. We briefly discuss how we have begun to apply four inter-related concepts to the coordinated, real-time scheduling component of our larger QoS management approach. The four concepts are:

- The use of standards-based middleware
- Building in run-time adaptability into scheduling
- "Weaving" scheduling into applications using aspect-oriented programming
- A model-integrated computing approach to scheduling

Standards-Based Middleware. A foundation to our approach to reducing the cost and complexity of constructing LDRE systems is the use of standards-based middleware that supports common non-functional elements typical to LDRE systems. Middleware is reusable software that forms a layer between the application and the underlying operating system, network protocol stack, and hardware. By abstracting away many of the low-level issues of network programming, middleware can:

- Eliminate the necessity for application developers to be aware of proprietary platform details, and enable applications to be more portable.
- Provide a consistent and organized set of capabilities that are closer in appearance and logic to design-level abstractions than to underlying computing and network mechanisms.
- Simplify the management of system resources.
- Amortize development lifecycle costs by leveraging previous expertise and encapsulating essential patterns in reusable frameworks, rather than rebuilding them from scratch for each project.
- Offer a wide variety of higher-level services oriented to the specific needs of developers, such as security or transactional logging.
- Facilitate the integration of heterogeneous legacy subsystems and software artifacts.

Integration of DRE system components via middleware is further facilitated by standards, and mature implementations that comply with them. In addition standards-based middleware makes it possible to replace or update components of DRE systems easily, without requiring costly and time-consuming application design changes.

We base our work on the Common Object Request Broker Architecture (CORBA) standard set forth by the Object Management Group (OMG) [1]. CORBA is the only standards-based commercial off-the-shelf (COTS) middleware implementation that has made substantial progress in enabling middleware to satisfy the quality of service (QoS) requirements of LDRE systems, in part due to the recent adoption of several related specifications including Real-time CORBA [2,3], which exports features that enable applications to reserve and manage memory, CPU, and network resources end-to-end while preserving predictability.

A common characteristic of the application domains of interest is that they operate under real world constraints that can vary significantly. In order address the dynamically changing nature of LDRE systems using middleware, our approach uses an architectural layer focused on supporting adaptable operation. We utilize the Quality Objects (QuO) framework, an adaptive middleware layer developed by BBN Technologies [4] as a building block for managing adaptive runtime behavior. QuO extends middleware with a *contract* feature that represents the QoS contracts between an application and the environment in which it is executing, The contract identifies the levels of QoS in which the application can operate, the tradeoffs and adaptations available when QoS changes, and the environmental conditions that must be measured and controlled to recognize and enforce QoS. QuO also provides off-the-shelf interface objects (based on CORBA IDL) to system resource mechanisms, services, and managers, called *system condition objects-delegates* for inserting QoS awareness and adaptation into the path of object interactions; and an object *gateway* for incorporating transport level QoS control. System condition objects are *wrapper facades* that enable the development of consistent middleware interfaces to low-level infrastructure, mechanisms, services, and managers, which might provide very different special-purpose interfaces and lie at different levels in the system and middleware stack. Delegates are *proxies* that look like remote object stubs or local method interfaces to the application so they can be transparently inserted into the path of object interactions, but with QoS-aware and adaptive code woven in.

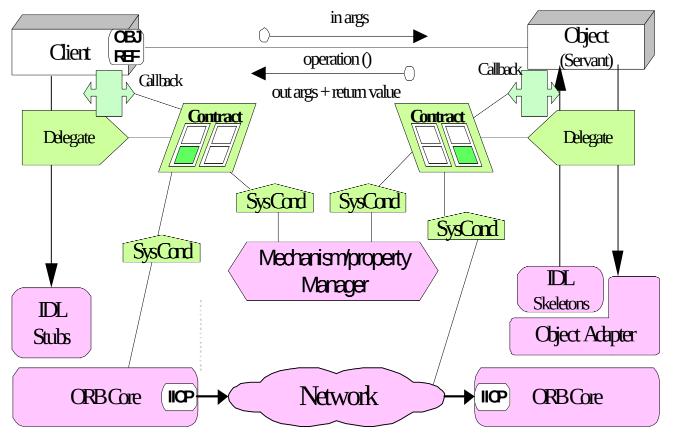

Figure 1: QuO Adaptable Middleware

Dynamically Adaptable Scheduling. To address the complexity introduced by the need to enforce real-time requirements, and the complexity introduced by dynamically changing requirements, our approach involves two steps that inject coordinated scheduling into standards-based middleware. The first step is to integrate the middleware Scheduling Service that is being developed at the University of Rhode Island (URI) [5]

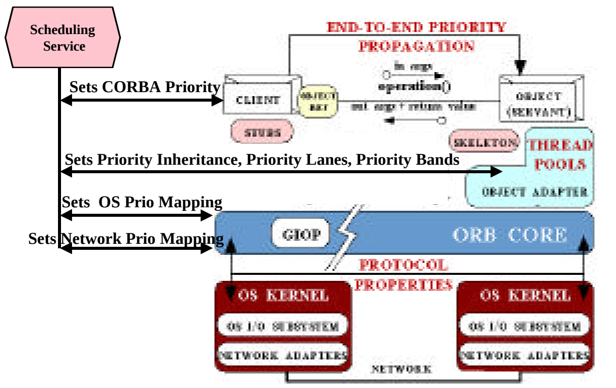

Figure 2: URI Scheduling Service

into the multi-level adaptive reflective QuO framework using Quo contracts. The second step is similar, but involves having the QuO framework directly control the real-time enforcement points in a Real-Time CORBA 2.0 compliant ORB, using similar algorithms in a similar coordinated fashion to what we will use in the Scheduling Service approach.

As depicted in Figure 2, the current URI Scheduling Service uses the Real-Time CORBA 1.0 standard Scheduling Service interface to set global CORBA priorities that are meaningful across the distributed system. The Scheduling Service also sets real-time parameters in the ORB including the ORB's priority mapping (to both the operating systems' and networks' local priorities), and the values for the POA's priority lanes and priority bands. The Scheduling Service does this by coordinating the following four scheduling algorithm categories: 1) priority assignment; 2) priority mapping; 3) overload management; and 4) service dispatch order. These coordinated algorithms ensure that the real-time enforcement in lower layers, such as the ORB and networks, is consistent, and in certain cases, analyzable. We are engineering the Scheduling Service so that the specific algorithm in each of these four algorithm categories is adaptable. Once Real-Time CORBA 2.0 ORBs become available with dynamic scheduling capabilities built into the ORB, which we expect within a year, we will use a similar coordinated algorithm technique, but with the four algorithms and their parameters being set by QuO directly in the ORB rather than necessarily through the external Scheduling Service.

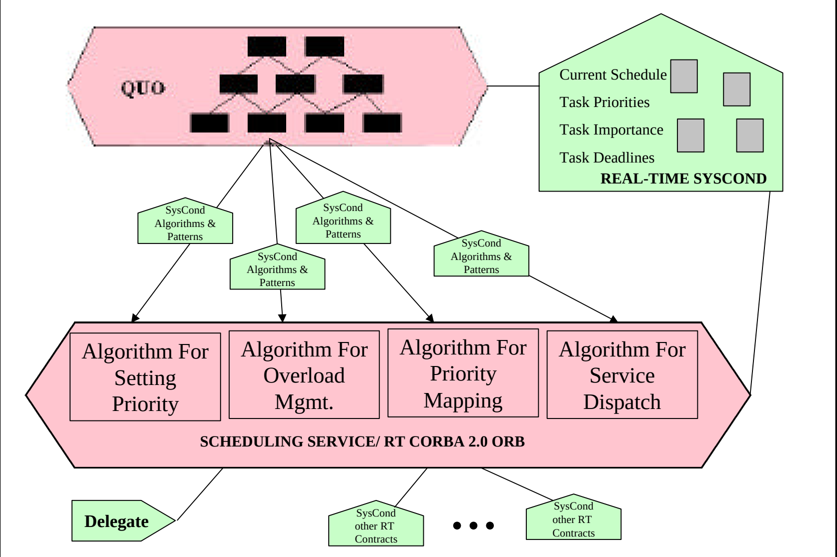
QUO
Current Schedule
Task Priorities
Task Importance
Task Deadlines
REAL-TIME SYSCOND
SysCond Algorithms & Patterns
SysCond Algorithms & Patterns
SysCond Algorithms & Patterns
SysCond Algorithms & Patterns
Algorithm For Setting Priority
Algorithm For Overload Mgmt.
Algorithm For Priority Mapping
Algorithm For Service Dispatch
SCHEDULING SERVICE/ RT CORBA 2.0 ORB
Delegate
SysCond other RT Contracts
• • •
SysCond other RT Contracts

Figure 3: QuO Control of Scheduling

To help manage the complexity that dynamic environments introduce to LDRE system, we will use the QuO middleware framework in several ways depicted in Figure 3. The multi-level QoS management will configure the adaptable algorithms described above based on reflective system information that will be gathered by monitoring services and reported using QuO's System Conditions (shown above the Scheduling Service in Figure 3). In turn, the real-time services will feed back analyzed system state information, such as the current planned schedule for resources and analysis-detected potential overload. This analysis would, for instance, enable an overload management technique to re-negotiate QoS contracts based on analyzed overload conditions before the overload occurs. Much of the real-time scheduling information, such as deadlines, periods, expected and measured execution times, measured utilizations of resources, the current schedule, the designation of importance of tasks, etc, will be kept in QuO system condition objects, as shown in Figure 3. This makes the real-time information available to all managers, mechanisms, and services through the middleware.

The use of an adaptable standards-based middleware scheduling mitigates some of the complexity due to real-time and adaptive requirements faced by LDRE developers. If developers can use off-the-shelf middleware components that are configured to provide comprehensive, coordinated real-time support, then their software simply has to interface to those adaptable scheduling components instead of developers having to try to inject adaptive real-time enforcement into LDRE software themselves.

Aspect-Oriented Scheduling.
Although standards based middleware and scheduling can remove some of the complexity from LDRE development, interacting with the scheduling elements correctly within the functional application can still be complicated. To ease this burden on LDRE developers, we use the QuO delegates to allow them to employ *aspect-oriented programming* (AOP) [6], to insert interactions with the standard middleware schedulers as an *aspect*. The AOP methodology and associated tools allow programmers to decouple the functional development of programs from aspects that cross-cut all or many components and levels in the system. These programming of these aspects are then done separately and "woven" into the program with tools. Typical AOP techniques allow the aspects to be "turned on" or "turned off" easily in the resulting application software. Real-time is such an aspect – its enforcement requires a coordinated approach at all levels of the system and typically spans multiple components. The application interface to the Scheduling Service and/or real-time ORB employs QuO delegates (shown at the bottom of Figure 3) to send the required parameters, such as deadlines and Importances, to the real-time enforcement mechanisms and services and handle the resulting priority and other return values appropriately. For instance, the delegate can invoke the underlying ORB to set in a client the CORBA priority that was received from the Scheduling Service.

Thus, in our approach, not only are the details of complex adaptive real-time enforcement embodied in the off-the-shelf middleware, the interfaces to those middleware components, which themselves can be somewhat complex, can be designed and programmed separately from the functional development of the LDRE application.

Model Driven Scheduling.
To bring several of the concepts of our approach together and allow them to work in larger scales, we use a *Model-Integrated Computing* (MIC) approach. MIC creates a unified software architecture and framework for creating LDRE software. The core components of the MIC infrastructure are: a customizable *Generic Model Editor* [7] for creation of multiple-view, domain-specific models; *Model Databases* for storage of the created models; and, a *Model Interpretation* technology that assists in the creation of domain-and application-specific model interpreters for transformation of models into executable/analyzable artifacts. The new environment is domain-specific and includes tools to support the creation and storage of system models, in addition to generation of executable/analyzable artifacts from these models.

Using the MIC technology, we are developing a domain-specific modeling environment called the Adaptive Quality Modeling Environment (AQME) for modeling of key aspects of a QoS Adaptive LDRE system [8] (see Figure 4):

- *QoS Adaptation Modeling* – In this category, the adaptation of QoS properties of the LDRE system is modeled. The designer can specify the different state configurations of the QoS properties, the legal transitions between the different state configurations, the conditions that enable these transitions (and the actions that must be performed to enact the change in state configuration), the data variables that receive and update QoS information, and the events that trigger the transitions. These properties are modeled using an extended finite-state machine (FSM) modeling formalism. AQME can then generate QuO contracts.

- *Computation Modeling* – In this category, the computational aspect of a LDRE system is modeled. A dataflow model is created in order to specify the various computational components and their interaction. This model takes the form of hierarchical task graphs. Our approach involves using AQME to address the complexities of real-time scheduling, by plugging the RapidRMA scheduling analysis tool developed by Tri-Pacific Software [9].
- *Middleware Modeling* – In this category, the middleware services, the system monitors, and the tunable "knobs" (i.e., the parameters being provided by the middleware) are modeled. AQME can then direct the generation of code points for aspect weaving, where middleware configuration can be either monitored or adapted to changing QoS requirements.

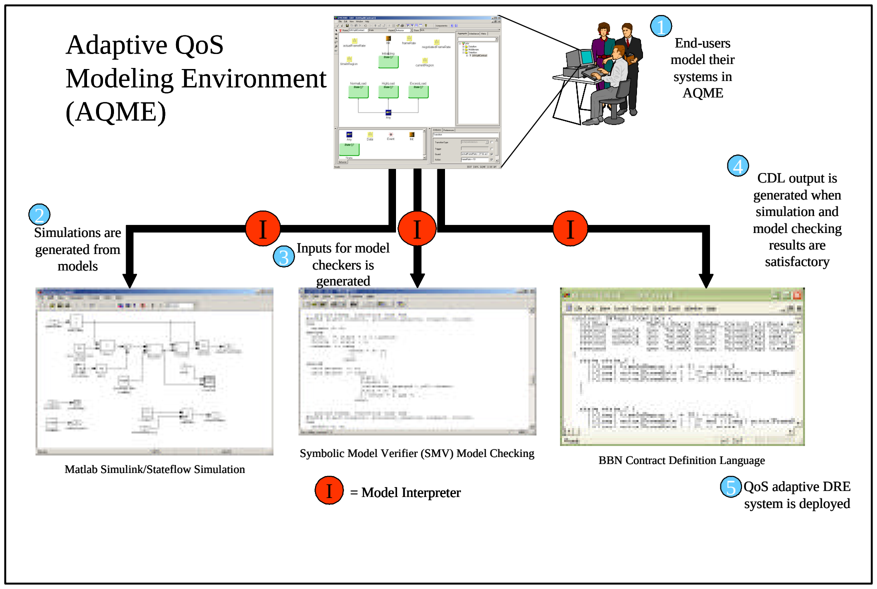
Adaptive QoS Modeling Environment (AQME)
1 End-users model their systems in AQME
2 Simulations are generated from models
3 Inputs for model checkers is generated
4 CDL output is generated when simulation and model checking results are satisfactory
5 QoS adaptive DRE system is deployed
I
Matlab Simulink/Stateflow Simulation
Symbolic Model Verifier (SMV) Model Checking
BBN Contract Definition Language
I = Model Interpreter

Figure 4: GME Modeling and Generation of QuO Middleware

Several generators have been developed using the model interpretation technology to perform various domain-specific analyses and simulations of the modeled system. For example, RapidRMA, as mentioned above, has been integrated with AQME, and can perform real-time schedulability analysis on a system using its own meta-modeling paradigm, which includes task locations, resource locations, task dependencies, and task-specific resource usage.

When the real-time analysis results are satisfactory, the modeled system can be synthesized for deployment on a platform, using a separate generator. Figure 5 (next page) shows our envisioned approach of model generators providing the scheduling delegates used by the application and the QuO contracts used to control the middleware scheduling (see Figure 3). The generated delegates would express real-time requirements, such as deadlines. The generated contracts would ensure that the middleware provides appropriate adaptable real-time scheduling.

Summary. This paper has presented some elements of our approach to handling the complexity of LDRE systems. In particular it focused on several concepts for support of real-time enforcement: adaptive middleware scheduling, providing scheduling interactions through aspect-oriented programming (AOP), and bringing these concepts together for large scale developments through model-integrated computing (MIC). These elements are part of our more comprehensive approach to providing distributed QoS using adaptive reflexive middleware, AOP, and MIC.

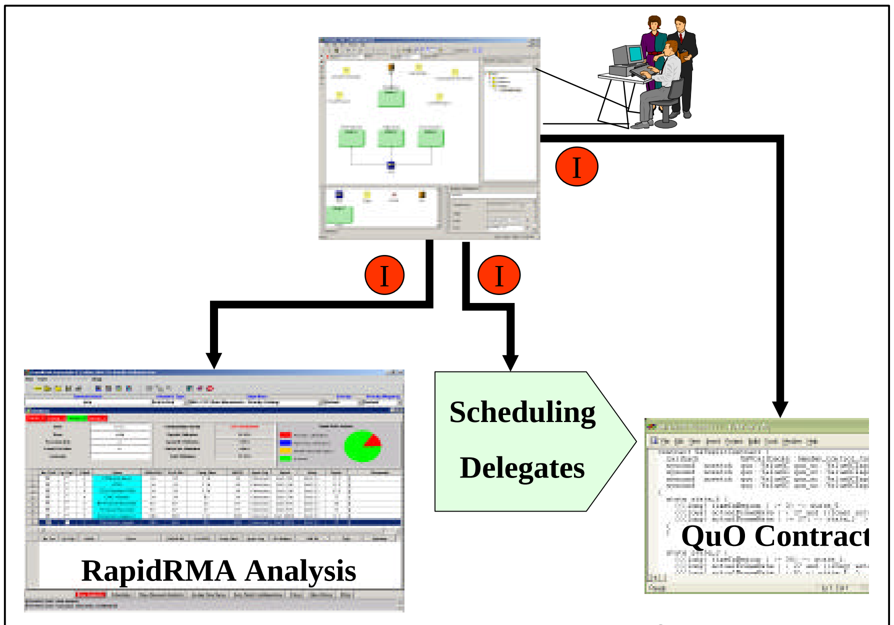

Figure 5: GME Modeling and Generation of Scheduling

.

References

[1] Object Management Group, *The Common Object Request Broker: Architecture and Specificatio*n, 3.0 edition, June 2002. at http://www.omg.org/cgi-bin/doc?formal/02-06-33

[2[Object Management Group, *Real-time CORBA Specification*, 1.1 edition, August 2002 at *http://www.omg.org/cgi-bin/doc?formal/02-08-02*

[3] Object Management Group, *Real-time CORBA 2.0: Dynamic Scheduling Specification*, ptc/01-08-34 edition, September 2002 at http://www.omg.org/cgi-bin/doc?ptc/2001-08-34

[4] Quality Objects at http://quo.bbn.com

[5] L. DiPippo, V. F. Wolfe, L. Esibov, G. Cooper, R. Johnston, B. Thuraisingham, J. Mauer, Scheduling and Priority Mapping for Static Real-Time Middleware, *Real-Time Systems*, 20, 155-182, 2001, Kluwer Academic Publishers. Similar version available at http://homepage.cs.uri.edu/research/rtsorac/pubs/rtsysj2.pdf

[6] Aspect-Oriented Programming at http://aosd.net

[7] GME at http://www.isis.vanderbilt.edu/Projects/gme/default.html

[8] Nanbor Wang, Douglas C. Schmidt, Aniruddha Gokhale, Christopher D. Gill, Balachandran Natarajan, Craig Rodrigues, Joseph P. Loyall, Richard E. Schantz and Richard Shapiro, Applying Model-Integrated Computing to Provision Middleware and Application Quality of Service," Submitted to *the 23rd International Conference on Distributed Computing Systems (ICDCS-23)*, (Providence, RI), IEEE, May 2003. available at http://www.cs.wustl.edu/~nanbor/papers/icdcs03.pdf

[9] RapidRMA at http://www.tripac.com/html/prod-fact-rrm.htm