Towards Developing QoS-enabled Distributed Real-time and Embedded Applications

Nanbor Wang
Christopher D. Gill
{nanbor,cdgill}@cse.wustl.edu
Dept. of Computer Science and Engineering

Washington University
One Brookings Drive
St. Louis, MO 63130, USA

Douglas C. Schmidt, Aniruddha Gokhale,
and Balachandran Natarajan
{schmidt,gokhale,bala}@dre.vanderbilt.edu
Institute for Software
Integrated Systems
Vanderbilt University
Box 1829, Station B
Nashville, TN 37203, USA

Joseph P. Loyall, Richard E. Schantz, and Craig Rodrigues
{jloyall,schantz,crodrigu}@bbn.com
BBN Technologies
10 Moulton Street
Cambridge, MA 02138, USA

Overview

Distributed real-time and embedded (DRE) applications, such as flight avionics systems, process control systems, and financial trading systems, have stringent quality of service (QoS) requirements that must be satisfied simultaneously in real-time. Examples of these QoS requirements include allocation of processing resources and network latency, jitter, and bandwidth. Failure to meet these QoS requirements can lead to catastrophic consequences, such as failure to avoid a collision in an avionics system or failure to reposition an actuator in a process control system.

To ensure DRE applications can achieve their QoS requirements, the following types of QoS *provisioning* are needed to allocate and manage system computing and communication resources end-to-end:

- **Static provisioning**, where the amount of resources required to support a particular degree of QoS is preconfigured into an application. Examples of static QoS provisioning include task prioritization and communication bandwidth reservation.

- **Dynamic provisioning**, where the amount of resources required are determined and adjusted based on the runtime system status. Examples of dynamic QoS provisioning include runtime reallocations to handle bursty CPU load, primary and second storage, and network traffic demands.

Our book chapter begins by describing the evolution of middleware over the past 10+ years to show how we got to where we are today, *e.g.*, from procedures to objects, which sets the stage for where we are headed next in this area, *e.g.*, components and model driven architectures (MDA). It will then focus specifically on how QoS resources can be provisioned statically and dynamically under the control of standards-based commercial off-the-shelf (COTS) middleware, such as Real-time CORBA and Real-time Java. The use of standard COTS middleware is increasingly gaining acceptance in the DRE community due to (1) the effort required to develop and verify complex DRE applications, which precludes developers from implementing DRE applications from scratch and (2) the maturation of implementations of standard COTS middleware specifications, such as Real-time CORBA and Real-time Java.

Extending Object-Oriented Middleware

Although standard COTS middleware provides some mechanisms to configure and control the underlying OS and network to manage application resource requirements end-to-end, they are not yet coordinated, do not yet adapt dynamically to runtime variations, and do not yet provide sufficient abstractions to separate real-time QoS policy configurations from application functionality. DRE application developers therefore must

configure and coordinate QoS policies in an *ad hoc* way today and the code to configure these policies is often scattered throughout a DRE system. As a result, it is hard for developers to configure, validate, modify, and evolve complex DRE systems consistently.

There are three areas of simultaneous improvement necessary to address these needs:

1. Combined static and dynamic end-to-end QoS managed behavior for DRE systems
2. Components for integrating bigger chunks and higher level conformity than objects
3. Model-based software engineering as a means to deal with increasing complexity and detail of these environments.

In recent years, component middleware (such as CORBA Component Model (CCM), J2EE and .NET) has emerged to address the limitations of building large-scale, complex applications with object-oriented middleware. Component middleware addresses these limitations by separating various concerns of building complex applications into different aspects that can be specified and composed at various lifecycle points of application development. Component middleware achieves these separation of concerns by:

- Expanding the object model with a new *component metatype*, which encapsulates individually-deployable implementations of application functionality that interact with each other through well-defined interfaces and
- Defining the mechanisms and standards for composing and executing components in generic component servers.

Conventional component middleware technologies were designed largely for applications with conventional business-oriented QoS requirements, such as data persistence, encryption, and transactional support. Many DRE-relevant QoS properties, such as task priorities, communication bandwidth reservations, and power consumption, are therefore not considered in these component technologies. Moreover, QoS provisioning in large-scale DRE applications cross-cuts multiple system layers and requires end-to-end enforcement. It is therefore insufficient to implement QoS provisioning in isolated component implementations.

This chapter will show how standard COTS component middleware can be extended to support static and dynamic resource provisioning. Our approach in this chapter will be through weaving together three threads of discussion:

- Describing the need and mechanisms for managing and coordinating end-to-end behavior in a predictable manner, and providing an integrated software engineering approach toward establishing it.
- Describing the state-of-the-art/practice in component middleware to provide coherent "large chunks of integratable functionality" with a common approach to separating aspects of concern, leading to our current focus on QoS-enabled component middleware.
- Describing how even a component middleware solution still requires substantial manual provisioning and configuration, which motivates the focus on model driven architecture approaches.

Extending Component Middleware

The new QoS-enabled CCM middleware provides mechanisms for composing QoS provisioning policies and configuring QoS support mechanisms. This middleware by itself, however, does not effectively resolve the following three key configuration and integration challenges:

- Choosing, customizing, and assembling the appropriate set of semantically compatible QoS-enabled DRE middleware components tailored to application QoS requirements.
- Coordinating among the selected components for appropriate end to end behavior under static and changing conditions, for both single activity and aggregate activities sharing resources.
- The obsolescence of infrastructure technologies and its impact on long-term DRE application lifecycle costs.

An emerging trend for addressing these challenges is to combine Model Driven Architecture (MDA) technologies with QoS-enabled component middleware. MDA is an emerging paradigm for expressing application functionality and QoS requirements at higher levels of abstraction than is possible using third-generation programming languages, such as Visual Basic, Java, C++, or C#. In the context of DRE applications, MDA methods and tools can be applied to

1. Analyze different–but interdependent–characteristics of DRE system behavior, such as scalability, predictability, safety, and security. Tool-specific model interpreters translate the information specified by models into the input format expected by analysis tools. These tools can check whether the requested behavior and properties are feasible given the specified application and resource constraints.
2. Synthesize platform-specific code that is customized for particular component middleware and DRE application properties, such as end-to-end timing deadlines, recovery strategies to handle various runtime failures in real-time,

and authentication and authorization strategies modeled at a higher level of abstraction.

Combining MDA and QoS-enabled component middleware effectively is essential to resolve key configuration and integration challenges of complex DRE applications. Our book chapter will therefore provide the following three contributions to the successful combination of MDA and QoS-enabled component middleware that is essential to address these challenges:

- We illustrate how enhancements to standard component middleware can simplify the development of DRE applications by composing QoS provisioning policies statically with applications. Our discussion focuses on a QoS-enabled enhancement of the standard CORBA Component Model (CCM) called the Component-Integrated ACE ORB (CIAO), which is being developed at Washington University, St. Louis and Vanderbilt University as extensions to the ADAPTIVE Communication Environment (ACE) and The ACE ORB (TAO).

- We describe how dynamic QoS provisioning and adaptation can be addressed using middleware capabilities called Qoskets, which are enhancements of the Quality Objects (QuO) middleware developed by BBN Technologies. Our discussion focuses on how Qoskets can be combined with CIAO to compose adaptive QoS assurance into DRE applications dynamically. In particular, Qoskets manage modular QoS aspects, which can be combined with CIAO and woven to create an integrated QoS-enabled component model.

- We discuss how QoS-enabled component middleware enables MDA tools to rapidly develop, generate, assemble, and deploy flexible DRE applications, yet can be tailored readily to meet the needs of multiple simultaneous QoS requirements. Our discussion focuses on the Component Synthesis with Model-Integrated Computing (CoSMIC) tools being developed by the Institute for Software Integrated Systems (ISIS) at Vanderbilt University.

All the material presented in this book chapter is based on the ACE, TAO, CIAO, and QuO middleware, which is available in open-source format from `deuce.doc.wustl.edu/Download.html` and `quo.bbn.com/quorelease.html`. This middleware has been applied to many production systems worldwide in many domains, including telecommunications, aerospace, financial services, process control, scientific computing, and distributed interactive simulations.