CoSMIC: An MDA Generative Tool for Distributed Real-time and Embedded Applications*

Aniruddha Gokhale, Douglas Schmidt
Tao Lu, Balachandran Natarajan
ISIS
Vanderbilt University
Box 1829, Station B
Nashville, TN 37235
{gokhale,schmidt,lut,bala}@dre.vanderbilt.edu

Nanbor Wang
Computer Science
Washington University
Box 1045, One Brookings Drive
St. Louis, MO 63130
nanbor@cse.wustl.edu

Abstract

The Object Management Group (OMG) has adopted the Model Driven Architecture (MDA) to standardize the integration of the modeling and simulation paradigm with middleware technology platforms. The MDA defines platform-independent models (PIMs) and platform-specific models (PSMs) that streamline platform integration issues and protect investments against the uncertainty of changing platform technology. This technology has been most successful to date notably for enterprise and business applications, where modeling techniques using the Unified Modeling Language (UML) have been integrated with component middleware technologies, such as Enterprise Java Beans (EJB), Microsoft's .NET, and the CORBA Component Model (CCM).

The MDA technology is yet to make its impact in the domain of distributed real-time and embedded applications (DRE) in areas such as avionics, telecommunications, industrial process control and defense. Recent efforts, notably within OMG and some DARPA DoD programs, have started addressing these issues.

This paper provides three contributions to the R&D in applying MDA technology to DRE applications. First, we delineate seven points of integration of model driven techniques with DRE component middleware frameworks. Second, we describe our MDA tool suite, called CoSMIC (Component Synthesis using Model Integrated Computing). Finally, we describe the modeling and generative programming approach of CoSMIC used to statically configure and fine tune component middleware tailored to provide the quality of service (QoS) requirements of DRE applications.

*Work supported by DARPA PCES grant number F33615-00-C-1695

1 Introduction

Commercial-off-the-shelf (COTS) distribution middleware technologies, such as OMG CORBA, Sun's J2EE/EJB and Microsoft's COM+/SOAP/.NET, have matured considerably in recent years. They are increasingly used to reduce the time and effort required to develop applications in a broad range of domains. These middleware technologies, however, have historically been applied to enterprise applications.

More recently, middleware has been applied to distributed real-time and embedded (DRE) applications with stringent quality of service (QoS) requirements for predictability, latency, efficiency, scalability, dependability, and security. DRE application developers face similar challenges as enterprise applications developers when dealing with heterogeneity arising out of differences in hardware, operating systems, programming languages, and middleware. In addition, they also need to ensure that applications obtain the levels of QoS they require while also keeping total ownership costs low and maintain a long shelf life.

As is the case with enterprise applications, a promising way to address the DRE software development and integration challenges is to combine OMG's Model Driven Architecture (MDA) with QoS-enabled component middleware, such as CIAO [1] which is a CORBA Component Model (CCM) implementation tailored to the requirements of DRE applications.

In the context of DRE applications, MDA-based tools can be applied to

1. **Analyze** different – but interdependent – characteristics of 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 the static and dynamic QoS provisioning challenges of complex DRE systems. This paper provides the following three contributions to the successful integration of MDA and QoS-enabled component middleware that is essential to address these challenges: First, we illustrate seven points of integration of MDA with DRE component middleware frameworks, such as CIAO; Second, we describe our MDA tool suite, called CoSMIC (Component Synthesis using Model Integrated Computing); Third, we describe the modeling and generative programming approach of CoSMIC used to statically configure and fine tune component middleware tailored to provide the QoS requirements of DRE applications.

2 Integrating MDA with QoS-enabled Component Middleware

Section 1 outlined the key challenges associated with developing DRE applications with multidimensional QoS requirements. Integrating OMG MDA with QoS-enabled component middleware is a promising approach to address these challenges. This integration can provide the following benefits:

- Combining MDA with component middleware helps to overcome problems [2] with earlier-generation CASE tools since it does not require the modeling tools to generate all the code. Instead, large portions of applications can be *composed* from reusable, prevalidated middleware components.
- Combining MDA and component middleware helps address environments where control logic and procedures change at rapid pace, by synthesizing and assembling newer extended components that implement the new procedures and processes.
- Combining component middleware with MDA helps to make middleware more flexible and robust by automating the configuration of many QoS-critical aspects, such as concurrency, distribution, resource reservation, security, and dependability. Moreover, MDA-synthesized code can help bridge the interoperability and portability problems between different middleware for which standard solutions do not yet exist.
- Combining component middleware with MDA helps to model the interfaces among various components in terms of standard middleware, rather than language-specific features or proprietary APIs.
- Changes to the underlying middleware or language mapping for one or many of the components modeled can be handled easily as long as they interoperate with other components. Interfacing with other components can be modeled as constraints that can be validated by model checkers.

Figure 1 illustrates seven points at which OMG MDA can be integrated into component middleware architectures and applied to DRE applications. We describe each

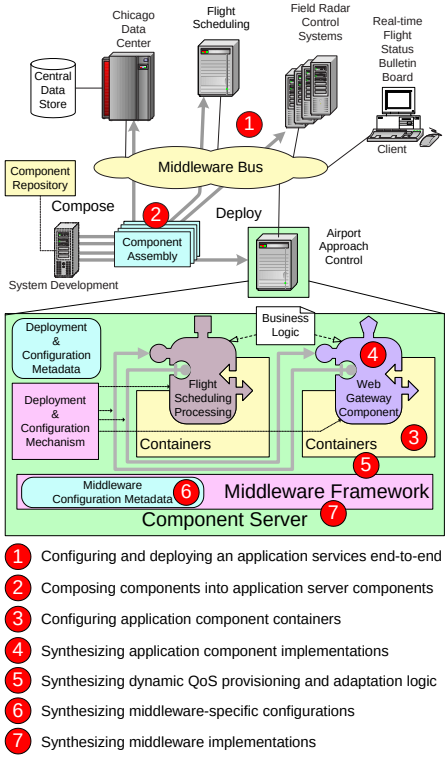

Figure 1: **Integrating MDA with Component Middleware**

of these seven integration points below:

1. Configuring and deploying application services end-to-end. Developing complex DRE applications requires application developers to handle a variety of configuration and deployment challenges, such as

- Locating the appropriate existing services
- Partitioning and distributing application processes among component servers using the same middleware technologies and
- Provisioning the QoS required for each service that comprises an application end-to-end.

It is a daunting task to identify and deploy all these capabilities into an efficient, correct, and scalable end-to-end application configuration. For example, to maintain correctness and efficiency, services may change or migrate when the DRE application requirements change. Careful analysis is therefore required to partition collaborating services on distributed nodes so the information can be processed efficiently, dependably, and securely.

Integrating MDA and component middleware to deploy DRE application services end-to-end can help developers configure the right set of services into the right part of an application in the right way. MDA analysis tools can help determine the appropriate partitioning of functionality that should be deployed into various component servers throughout a network.

2. Composing components into component servers. Integrating MDA with component middleware provides capabilities that help application developers to compose components into application servers by

- Selecting a set of suitable, semantically compatible components from reuse repositories.
- Specifying the functionality required by new components to isolate the details of DRE systems that (1) operate in environments where DRE processes change periodically and/or (2) interface with third-party software associated with external systems.
- Determining the interconnections and interactions between components in metadata.
- Packaging the selected components and metadata into an assembly that can be deployed into the component server.

3. Configuring application component containers. Application components use containers to interact with the component servers in which they are configured. Containers manage many policies that distributed applications can use to fine-tune underlying component middleware behavior, such as its priority model, required service priority level, security, and other quality of service properties. Since DRE applications consist of many interacting components, their containers must be configured with consistent and compatible QoS policies.

Due to the number of policies and the intricate interactions among them, it is tedious and error-prone for a DRE application developer to *manually* specify and maintain component policies and semantic compatibility with policies of other components. MDA tools can help automate the validation and configuration of these container policies by allowing system designers to specify the required system properties as a set of models. Other MDA tools can then analyze the models and generate the necessary policies and ensure their consistency.

4. Synthesizing application component implementations. Developing complex DRE applications today involves programming new components that add application-specific functionality. Likewise, new components must be programmed to interact with external systems and sensors, such as a machine vision module controller, that are not internal to the application. Since these components involve substantial knowledge of application domain concepts, such as mechanical designs, manufacturing process, workflow planning, and hardware characteristics, it would be ideal if they could be developed in conjunction with mechanical engineers or domain experts, rather than programmed manually in isolation by software developers.

The shift toward high-level design languages and modeling tools is creating an opportunity for increased automation in generating and integrating application components. The goal is to bridge the gap between specification and implementation via sophisticated aspect weavers [3] and generator tools [4] that can synthesize platform-specific code customized for specific application properties, such as resilience to equipment failure, prioritized scheduling, and bounded worst-case execution under overload conditions.

5. Synthesizing dynamic QoS provisioning and adaptation logic. Based on the overall system model and constraints, MDA tools may decide to plug in existing dynamic QoS provisioning and adaptation modules, such as QuO [], using appropriate parameters. When none is readily available, the MDA tools can assist in creating the new behavior by synthesizing the logic using the languages provided by the adaptation modules. The generated dynamic QoS behavior can then be used in system simulation dynamically to verify its validity. It can then be composed into the system as described above.

6. Synthesizing middleware-specific configurations. The infrastructure middleware technologies used by component middleware provide a wide range of policies and options to configure and tune their behavior. For example, CORBA ORBs often provide the following options and tuning parameters:

- Various types of transports and protocols
- Various levels of fault tolerance
- Middleware initialization options
- Efficiency of (de)marshaling event parameters
- Efficiency of demultiplexing incoming method calls
- Threading models and thread priority settings and
- Buffer sizes, flow control, and buffer overflow handling

Certain combinations of the options provided by the middleware may be semantically incompatible when used to achieve multiple QoS properties.

For example, a component middleware implementation could offer a range of security levels to the application. In the lowest security level, the middleware exchanges all the messages over an unsecure channel. The highest security level, in contrast, encrypts and decrypts messages exchanged through the channel using a set of dynamic keys. The same middleware could also provide an option to use zero-copy optimizations to minimize latency. A modeling tool could automatically detect the incompatibility of trying to compose the zero-copy optimization with the highest security level (which makes another copy of the data during encryption and decryption).

Advanced meta-programming techniques, such as adaptive and reflective middleware [5, 6, 7, 8] and aspect-oriented programming [3], are being developed to configure middleware options so they can be tailored for particular DRE application use cases.

7. Synthesizing middleware implementations. MDA can also be integrated with component middleware by using generative tools to synthesize custom middleware implementations. This integration is a more aggressive use of modeling and synthesis than integration point 5 described above since it affects middleware *implementations*, rather than their configurations. Application integrators could use these capabilities to generate highly customized implementations of component middleware so that

- It only includes the features actually needed for a particular application and
- It is carefully fine-tuned to the characteristics of particular programming languages, operating systems, and networks.

2.1 CoSMIC: Component Synthesis using Model Integrated Computing

The Component Synthesis using Model Integrated Computing (CoSMIC) project is a MDA toolset being developed by the Institute for Software Integrated Systems (ISIS) at Vanderbilt University to (1) *model and analyze* distributed real-time and embedded application functionality and QoS requirements and (2) *synthesize* CCM-specific deployment metadata required to deliver end-to-end QoS to DRE applications.

The CoSMIC toolsuite provides modeling of DRE systems, their QoS requirements, and QoS adaptation policies used for DRE application QoS management. The component behavior, their interactions, and QoS requirements are modeled using a language similar to the Embedded Systems Modeling Language (ESML) [9]. Whereas ESML enables modeling a proprietary avionics component middleware, CoSMIC enables modeling the standards-based CCM components. Moreover, CoSMIC provides modeling languages to model the adaptive QoS behavior supported by QuO/Qoskets.

The CoSMIC project is developing synthesis tools targeted at the CIAO [1] component middleware, which is a real-time, QoS-enabled enhancement to CCM. CIAO abstracts component QoS requirements into metadata that can be specified in a component assembly after a component has been implemented. Decoupling QoS requirements from component implementations greatly simplifies the conversion and validation of an application model with multiple QoS requirements into CCM deployment of DRE applications.

The remainder of this section describes how we are combining the CoSMIC modeling and generative tools with the CIAO component middleware platform to address key challenges faced by the developers of DRE applications. Figure 2 illustrates the interface between CoSMIC and CIAO.

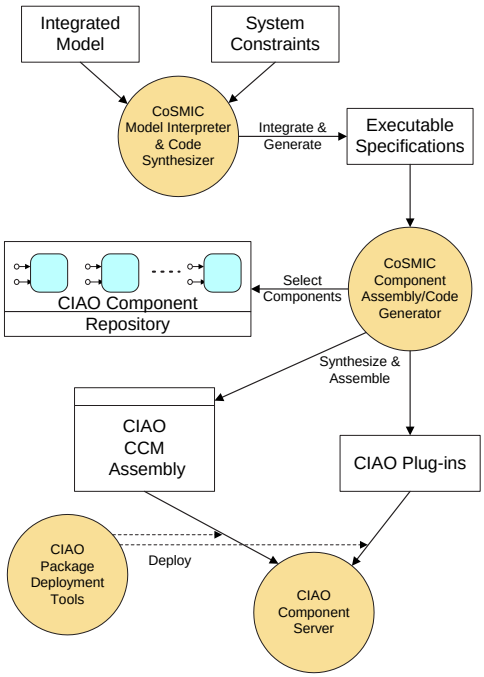

Figure 2: **Interaction between CoSMIC and CIAO**

Challenge 1: Satisfying Multiple Quality of Service (QoS) Requirements Simultaneously

Problem. DRE applications demand stringent QoS support from their middleware. For example, DRE applications such as controller for high-speed surface mount component pick-and-place machines require real-time predictability and performance guarantees. Due to (1) the complexity of these QoS requirements, (2) the heterogeneity of the environments in which they are deployed, and (3) the existing legacy systems and data, it is infeasible to develop a single-vendor, end-to-end solution that can address all these challenges. Instead, integrating highly configurable, flexible, and optimized COTS components from several different providers based on standard component middleware enables developers to assemble and deploy these systems rapidly and robustly. Ensuring application QoS requirements end-to-end, however, can be complicated.

Solution. A benefit of MDA is its ability to employ complex modeling tools that can check for certain properties of the implementation, *e.g.*, check the correctness of an algorithm or ensure that a series of constraints are enforced.

Although the OMG MDA standard has adopted the UML-based PIM and PSM for CORBA, it does not yet adequately address a broad spectrum of DRE application QoS issues. In particular, it does not address the integration of static and dynamic QoS provisioning mechanisms, such as priority propagation, resource allocations, dependability, predictability, and adaptation that are crucial to DRE applications.

The tools we are developing in CoSMIC are therefore designed to model and analyze both the application functionality and its end-to-end QoS requirements. With CIAO's support for QoS-enabled, reusable CCM components, it is possible to

- Model the QoS requirements of applications using UML
- Associate the model with different static and dynamic QoS profiles
- Simulate and analyze dynamic behaviors and
- Synthesize the QoS-enabled application functionality in component assemblies.

Figures 3, 4 and 5 , respectively, illustrate CoSMIC meta models for describing CCM container POA policies, QoS parameters and ORB configuration options.

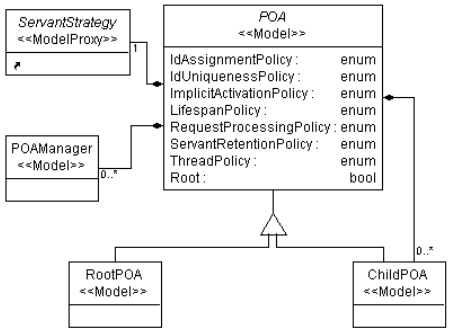

Figure 3: **CoSMIC Meta Model for Container POA Policies**

Application developers can use these meta models to model their application requirements. CoSMIC generative tools are then used to synthesize and assemble QoS-enabled, CCM middleware for DRE applications. This synthesis uses the following iterative process to assemble and deploy QoS-enabled distributed applications:

1. **Model the overall application** using CoSMIC visual modeling tools and specify the application's QoS requirements as constraints. This step defines and partitions the functionality and QoS requirements demanded by each application module based on the overall model of the application, as described by integration point 1 of Figure 1

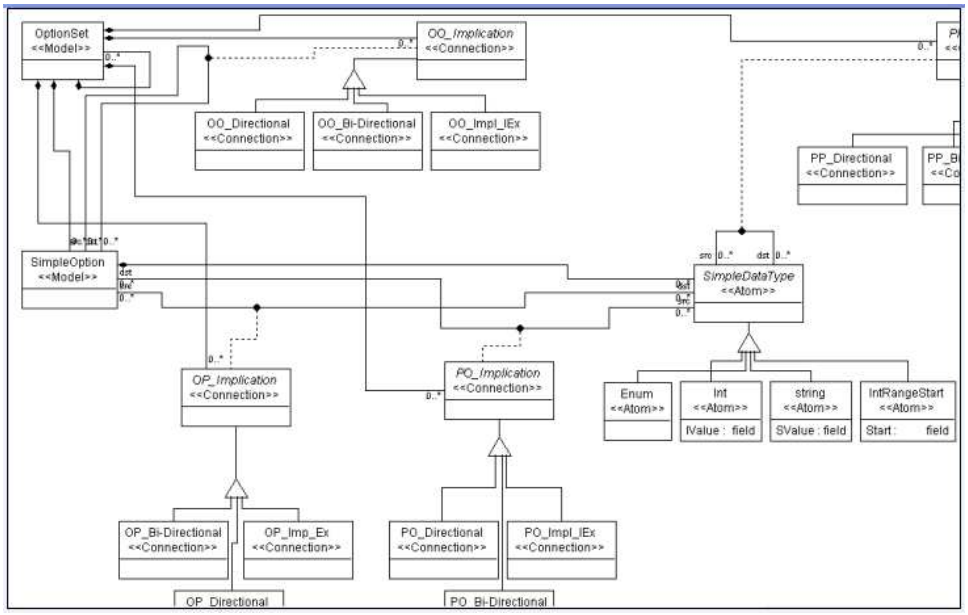

Figure 5: **CoSMIC Meta Model for ORB Configuration**

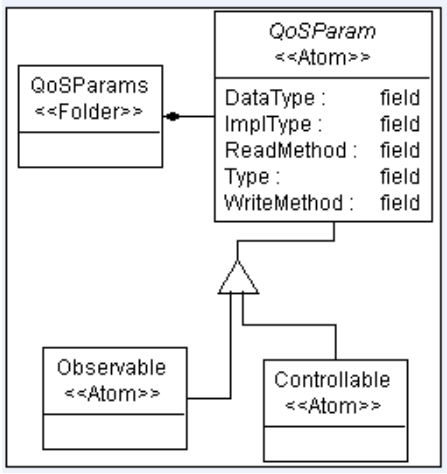

Figure 4: **CoSMIC Meta Model for QoS**

2. **Compose component servers** using CoSMIC component server composition tools to combine component assemblies by mixing and matching existing off-the-shelf components and partitioning or defining the functionality of new components, as needed, as shown in Point 2 of Figure 1. The metadata in a component assembly also contain QoS requirements for each components that the composition tools derived from the model.
3. **Model and synthesize components**—If new component implementations are needed from the previous step, each can be modeled by using CoSMIC's modeling tool. CoSMIC's component implementation synthesizer will generate the actual implementations based on the models, as indicated by integration point 4 of Figure 1.
4. **Validate and simulate applications** via CoSMIC tools that check whether an application composition implements its model definitions correctly.
5. **Deploy the resulting system for testing and tuning** via tools that fine-tune CIAO's QoS requirements for assemblies. Later iterations of this process can use these adjustments as feedback to improve the overall system model.

Challenge 2: Addressing Accidental Complexities in Integrating Software Systems

Problem. QoS-enabled component middleware, such as CIAO, provides libraries of reusable, configurable components that can be used to assemble and deploy QoS-aware DRE applications. However, a naive approach to assemble and configure these components can yield components with incompatible, non-interoperable QoS requirements, thereby increasing accidental complexities. Manual assembling components and configuring their QoS requirements are tedious and error-prone, which adversely affects application lifecycle costs and time-to-market. Moreover, to ensure these requirements are met end-to-end across a DRE application, component servers often explicitly require complex policies

and customized middleware plugins. Manually specifying and configuring these policies makes the development process even more vexing.

Solution. The iterative process described in the solution for Challenge 1 above helps DRE application developers manage the accidental complexity of assembling components by providing rich semantics in models and automatically propagating these semantics into assemblies through metadata. There is, however, a need to ensure that the component servers and the underlying middleware are configured properly to satisfy the QoS requirements demanded by the installed components.

The CCM specification does not yet address how to associate component QoS requirements with a component deployment. Our CCM implementation (CIAO) therefore supports the configuration of certain component QoS properties via the component deployment metadata shown by integration point 2 of Figure 1. Since we provide component QoS management services through containers in our CCM implementation [10], the synthesizing tools will also generate container configurations in a component assembly, as depicted in Point 3 of Figure 1.

To support QoS requirements that were not foreseen by the component middleware implementation, CoSMIC can also synthesize middleware modules that CIAO uses to customize its behavior to support non-native QoS supports required by other systems. CIAO's deployment framework then uses these customized modules to configure component servers before deploying the components, as shown by integration point 6 of Figure 1. The automation of semantic propagation described here ensures that all component servers consisting an integrated DRE application perform their work as specified in the overall model, without undue programmer intervention.

3 Concluding Remarks

This paper describes our R&D on a MDA tool called CoSMIC suitable for DRE applications. The paper describes our approach in using CoSMIC for a QoS-enabled CCM implementation, called CIAO. Currently, CoSMIC only provides CCM/CORBA platform specific metamodels to describe ORB configuration and POA policies. In future CoSMIC will provide platform independent metamodels to describe DRE application QoS requirements. Moreover, currently CoSMIC addresses static QoS provisioning. Later versions of CoSMIC will enable dynamic QoS provisioning so that it can be used with adaptive and reflective middleware.

References

[1] Nanbor Wang, Douglas C. Schmidt, Aniruddha Gokhale, Christopher D. Gill, Balachandran Natarajan, Craig Rodrigues, Joseph P. Loyall, and Richard E. Schantz. Total Quality of Service Provisioning in Middleware and Applications. *The Journal of Microprocessors and Microsystems*, 27(2):45–54, mar 2003.

[2] Paul Allen. Model Driven Architecture. *Component Development Strategies*, 12(1), January 2002.

[3] Gregor Kiczales, John Lamping, Anurag Mendhekar, Chris Maeda, Cristina Videira Lopes, Jean-Marc Loingtier, and John Irwin. Aspect-Oriented Programming. In *Proceedings of the 11th European Conference on Object-Oriented Programming*, June 1997.

[4] Akos Ledeczi, Arpad Bakay, Miklos Maroti, Peter Volgysei, Greg Nordstrom, Jonathan Sprinkle, and Gabor Karsai. Composing Domain-Specific Design Environments. *IEEE Computer*, November 2001.

[5] Fabio Kon, Fabio Costa, Gordon Blair, and Roy H. Campbell. The Case for Reflective Middleware. *Communications of the ACM*, 45(6):33–38, June 2002.

[6] Gordon S. Blair and G. Coulson and P. Robin and M. Papathomas. An Architecture for Next Generation Middleware. In *Proceedings of the IFIP International Conference on Distributed Systems Platforms and Open Distributed Processing*, pages 191–206, London, 1998. Springer-Verlag.

[7] Fábio M. Costa and Gordon S. Blair. A Reflective Architecture for Middleware: Design and Implementation. In *ECOOP'99, Workshop for PhD Students in Object Oriented Systems*, June 1999.

[8] Joseph K. Cross and Douglas C. Schmidt. Applying the Quality Connector Pattern to Optimize Distributed Real-time and Embedded Middleware. In Fethi Rabhi and Sergei Gorlatch, editors, *Patterns and Skeletons for Distributed and Parallel Computing*. Springer Verlag, 2002.

[9] Gabor Karsai, Sandeep Neema, Arpad Bakay, Akos Ledeczi, Feng Shi, and Aniruddha Gokhale. A Model-based Front-end to ACE/TAO: The Embedded System Modeling Language. In *Proceedings of the Second Annual TAO Workshop*, Arlington, VA, July 2002.

[10] Nanbor Wang, Douglas C. Schmidt, Michael Kircher, and Kirthika Parameswaran. Towards a Reflective Middleware Framework for QoS-enabled CORBA Component Model Applications. *IEEE Distributed Systems Online*, 2(5), July 2001.