

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

Douglas C. Schmidt

`schmidt@uci.edu`

Dept. of Electrical
and Computer Engineering
University of California
616E Engineering Tower
Irvine, CA 92697, USA

Aniruddha Gokhale, Balachandran Natarajan

Sandeep Neema, Ted Bapty, Jeff Parsons

{gokhale, bala, neemask, bapty, parsons}

@isis-server.vuse.vanderbilt.edu

Institute for Software Integrated Systems

Vanderbilt University

Box 1829, Station B

Nashville, TN 37203, USA

Jeff Gray

`gray@cis.uab.edu`

Dept. of Computer and
Information Sciences
115A Campbell Hall
1300 University Blvd

Univ of Alabama at Birmingham
Birmingham, AL 35294, USA

Andrey Nechypurenko

`andrey.nechypurenko@siemens.com`

Siemens AG, CT SE2

Otto-Hahn-Ring 6

Munich, 81739, Germany

Nanbor Wang

`nanbor@cse.wustl.edu`

Dept. of Computer Science
and Engineering
Campus Box 1045

One Brookings Drive

Washington University

St. Louis, MO 63130, USA

Abstract

This paper presents three contributions to the challenges of applying the OMG Model Driven Architecture (MDA) to develop and deploy distributed real-time and embedded (DRE) applications. First, we motivate our MDA tool called CoSMIC, which is based on the Model Integrated Computing (MIC) paradigm that provides the intellectual foundation for MDA. Second, we describe how CoSMIC's generative abilities can be used to configure and assemble DRE component middleware required to deploy DRE applications. Third, we delineate the challenges involved in developing CoSMIC. Based on our collective experience developing MIC tools and DRE middleware, we are confident that combining these two paradigms will yield significant advantages in developing model based DRE applications.

1 Introduction

Well over 95 percent of all microprocessors are now used for real-time and embedded systems. These systems are increasingly being networked together to form distributed real-time and embedded (DRE) systems. Many DRE systems are both mission-critical and constrained by the physical world.

We therefore need principled methods for specifying, programming, composing, integrating, and validating software for these systems that can enforce the physical constraints, as well as satisfy stringent quality of service (QoS) and functional requirements.

Due to constraints on weight, power consumption, memory footprint, and performance, development techniques for DRE application software have lagged those used for mainstream desktop and enterprise software. In particular, DRE applications have historically been manually programmed and customized from scratch to implement their required QoS properties, making them expensive to build and maintain. Moreover, they are often so specialized that they cannot adapt readily to meet new functional or QoS requirements, hardware/software technology innovations, or market opportunities.

To address the problems with manually developing and customizing DRE systems from scratch, there is growing interest in composing these types of systems using commercial off-the-shelf (COTS) hardware (such as COTS DSPs and CPUs) and software (such as real-time operating systems and QoS-enabled component middleware services). One of the key challenges in using COTS software in DRE systems is determining, assembling, and deploying a right mix of QoS-enabled COTS middleware components that can satisfy the stringent

QoS requirements of DRE systems. *Ad hoc* techniques, such as manually choosing the right mix of middleware components, do not scale well as the application size and requirements increase. Moreover, *ad hoc* techniques are often tedious, error-prone, and lack a solid verification and validation foundation.

To address these problems, we require tools that allow developers to specify application requirements at higher levels of abstraction than that provided by lower-level mechanisms, such as conventional general-purpose programming languages. These tools must be able to analyze the requirements and generate the required directives that will compose applications from the right set of COTS middleware components. A promising example of such tools are those based on *Model-Integrated Computing* (MIC) [1].

Model-Integrated Computing (MIC) is a development paradigm that applies domain-specific modeling languages systematically to engineer DRE computing systems. Popular examples of MIC toolsuits in use today include the Generic Modeling Environment (GME) [2] and Ptolemy [3]. These toolsuits provide rich, domain-specific modeling environments, including model analysis and model-based program synthesis tools. Work on MIC in the DARPA Model-based Integration of Embedded Systems (MoBIES) program [4] also provides the intellectual foundations of the OMG's MDA approach for DRE systems.

In the MIC paradigm, application developers model an integrated, end-to-end view of the entire application, including the interdependencies of its components. Rather than focusing on a single, custom application, MIC models capture the essence of a class of applications, similar to the goals of product-line architectures. MIC also allows the modeling languages and environments themselves to be modeled by so-called *meta-models* [5], which help to synthesize domain-specific modeling languages that can capture the nuances of domains they are designed to model.

Recent advances in QoS-enabled component middleware, such as the Component-Integrated ACE ORB (CIAO) [6] real-time CORBA Component Model (CCM) [7] middleware, make them amenable to composition of DRE applications from COTS component middleware. To use MIC for this composition requires the creation of domain-specific modeling languages that model the behavior and interaction of component middleware, such as CIAO. Moreover, MIC generative tools must be developed that understand these models and automatically configure and customize the middleware for DRE applications.

The remainder of this paper is organized as follows: Section 2 provides an overview of CoSMIC, which is an MDA toolsuite we have designed to integrate MIC and component middleware for DRE systems; Section 3 describes the challenges involved in developing CoSMIC tools that use genera-

tive techniques; and Section 5 presents concluding remarks.

2 Overview of CoSMIC

The *Component Synthesis using Model Integrated Computing* (CoSMIC) project at Vanderbilt University's Institute for Software Integrated Systems (ISIS) is developing domain-specific tools for composing and deploying DRE middleware-based applications. The initial set of CoSMIC tools are targeting a DRE component middleware suite that consists of the following frameworks:

- The Component Integrated ACE ORB (CIAO) [8, 6], which is a QoS-enabled CORBA Component Model (CCM) [7] middleware framework developed at Washington University, St. Louis, and
- The Quality Objects (QuO) [9] framework, which is an adaptive middleware developed by BBN Technologies.

The CoSMIC toolsuite is designed to (1) *model and analyze* DRE application functionality and QoS requirements and (2) *synthesize* CCM-specific deployment metadata for CIAO and QuO required to provision and enforce end-to-end QoS both statically and dynamically [6]. Figure 1 illustrates the key elements in the CoSMIC-based DRE application development process. The CoSMIC tools can be used to model the requirements and adaptation policies needed to manage the QoS of DRE applications. Figure 2 illustrates seven points at which CoSMIC can be integrated into the integrated CIAO and QuO component middleware and applied to DRE applications. The seven points of integration shown in Figure 2 include

1. **Configuring and deploying application services end-to-end**, which involves generating and provisioning the policies for partitioning and distributing application services and resources.
2. **Composing components into component servers**, which involves generating the directives to assemble semantically compatible application components from reuse repositories and determining the interconnections between these selected components.
3. **Configuring application component containers**, which comprises generating QoS policies, such as threading policies or levels of security and fault tolerance, for the containers hosting the components.
4. **Synthesizing application component implementations**, which consists of generating application components tailored to satisfying specific requirements, such as worst-case execution time of tasks.
5. **Synthesizing dynamic QoS provisioning and adaptation logic**, which includes generating the QoS provisioning and adaptation logic understood by adaptive frameworks, such as QuO.

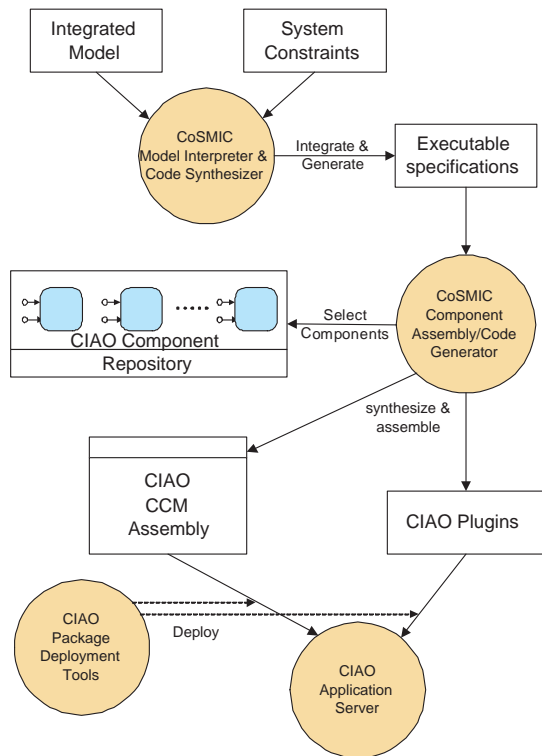


Figure 1: **Developing Component Middleware-based DRE Applications Using the CoSMIC Process**

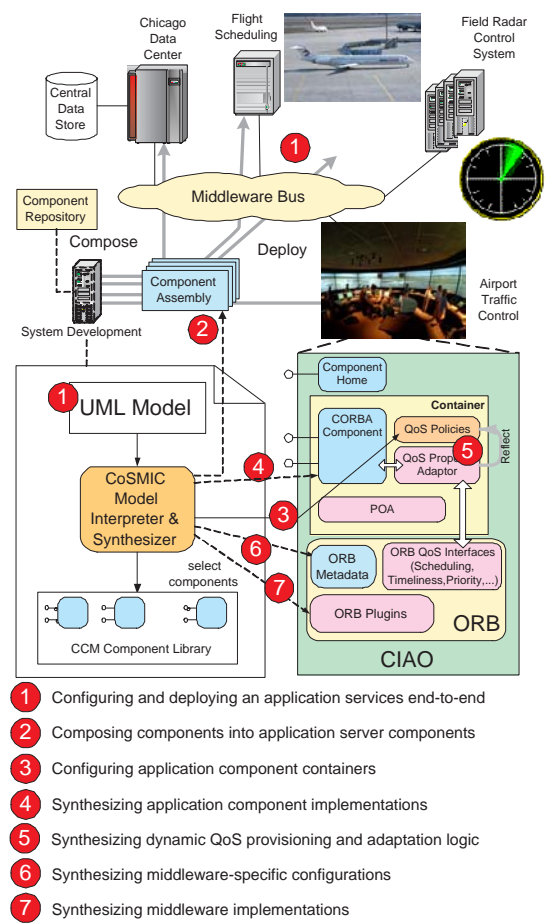
6. **Synthesizing middleware-specific configurations**, which involves generating the directives to configure the middleware, such as choice of transport protocols, threading models, and demultiplexing strategies.
7. **Synthesizing middleware implementations**, which comprises generating custom middleware components, such as components for memory constrained systems.

Additional information on these steps appears in [6].

3 The Design and Implementation Challenges of CoSMIC Generative Tools

This section outlines the challenges we have faced when developing the CoSMIC generative toolset.

Generative techniques for component middleware configuration metadata. The QoS requirements of DRE applications can be assured when the middleware they are based on are highly optimized and tailored to the application's QoS requirements. One way to achieve this is by generating the configuration metadata for parametrizing the middleware from



- 1 Configuring and deploying an application services end-to-end
- 2 Composing components into application server components
- 3 Configuring application component containers
- 4 Synthesizing application component implementations
- 5 Synthesizing dynamic QoS provisioning and adaptation logic
- 6 Synthesizing middleware-specific configurations
- 7 Synthesizing middleware implementations

Figure 2: **Incorporating Model-Integrated Computing with Component Middleware**

high-level models. Figure 3 illustrates the different metadata that needs to be generated to configure QoS-enabled component middleware. In particular, CoSMIC provides a CIAO/QuO metadata modeling language to generate metadata, such as XML descriptors that provide directives on assembling and packaging CORBA components along with the adaptation logic. The challenges also include modeling internal behavior of CIAO/QuO.

Generative techniques for component middleware container policies. Figure 4 illustrates the need to model and synthesize CCM container policies using aspect model weavers. This task involves refactoring and modeling cross-cutting DRE middleware QoS concerns, such as levels of fault tolerance, security, and persistence. The CoSMIC aspect model weaver tools [10] are designed to synthesize the appropriate CIAO container QoS policies [11]. For example, cross cutting concerns, such as security, are refactored from the models for CCM containers. CoSMIC aspect model

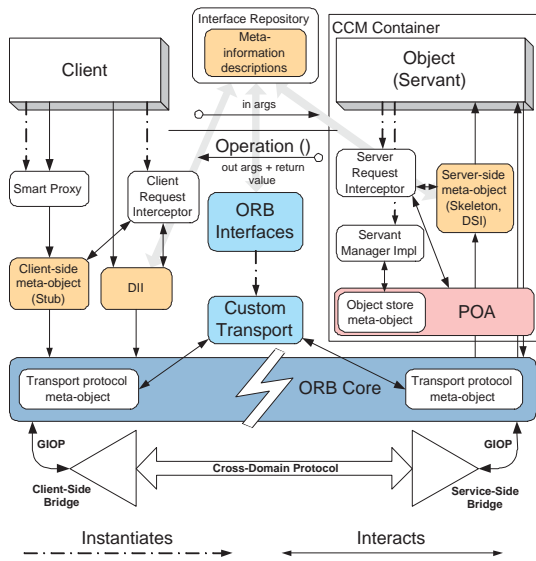


Figure 3: CIAO Metadata Configuration

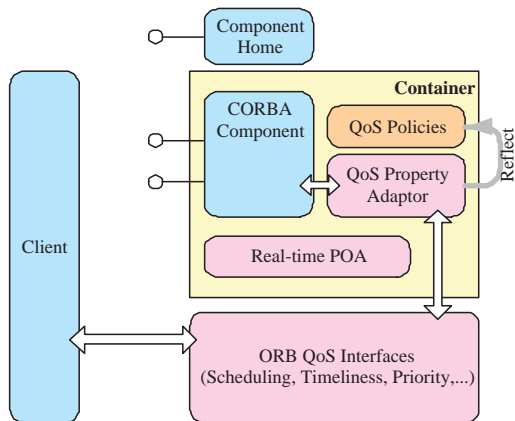


Figure 4: CIAO Container Policy Configuration

weavers generate specialized models of container policies by weaving in the security-related cross-cutting concerns into the original container models.

Generative techniques for synthesizing aspectized component middleware. Figure 5 illustrates the need to model and synthesize application component logic using aspect model weavers. Similar to the previous point, this task involves abstracting out and modeling separately all the cross-cutting DRE application QoS concerns, such as priorities of tasks, worst case execution times, and bandwidth requirements, in addition to application behavior and component interactions. The CoSMIC aspect model weaver tools are used to generate specialized models of application components by weaving in these cross-cutting concerns into the original models. CoSMIC's assembly generator tools then use these specialized models to synthesize components and their assembly, as

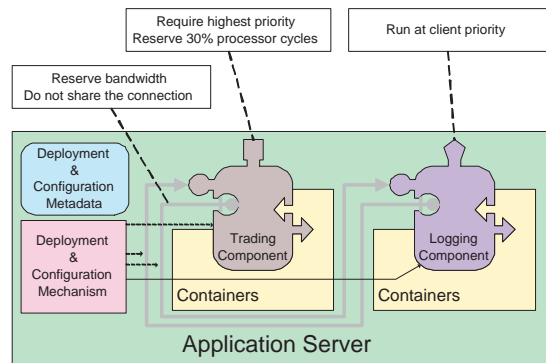


Figure 5: CIAO Component QoS Aspect Weaving

described before.

End-to-end QoS assurance. Assuring end-to-end application QoS involves QoS assurance at different levels of middleware, operating systems, and networks. End-to-end QoS can be attained in the following two ways:

- By aggregating the QoS offered at each layer, since each layer guarantees certain levels of QoS, or
- By each layer adapting their offered QoS based on offered QoS of adjacent layers.

Figure 6 illustrates the adaptive approach taken by QuO. Figure 6 illustrates one way of decoupling (1) the application's

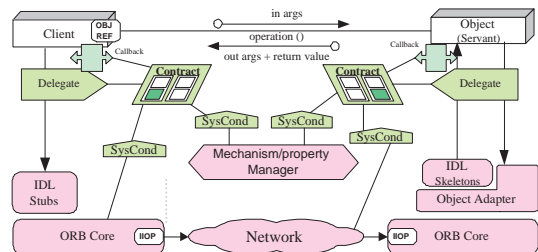


Figure 6: QoS Assurance via Adaptation in QuO

functional path, which consists of information flows between client and remote server applications, from (2) the QoS systemic paths, which are responsible for determining how well the functional interactions behave end-to-end with respect to key DRE QoS properties. In QuO, the QoS systemic properties are specified using the Quality Description Language (QDL) and the negotiated QoS contract between the application and middleware is specified in the Contract Description Language (CDL). The QuO framework monitors the resource usage whose values are stored in the *syscond* objects. Depending on the current values of the *syscond* objects, the QuO framework consults the CDL contracts to achieve runtime QoS adaptation.

CoSMIC modeling and generative tools are addressing both scenarios by modeling the offered QoS of individual layers and/or modeling the adaptive policies required by each layer.

4 Future Directions in Generative Programming and the Role of MDA

In generative programming, the set of requirements that describes the interface and semantic behavior is referred to as a *concept* [12]. A concept is a set of features that one software component expects from another. The implementation (or a model) of a particular concept makes it possible to plug the component to an existing infrastructure that knows how to deal with a specific concept, *e.g.*, the set of STL algorithms and the *iterators* concepts.

The idea of component containers could be also interpreted in a similar way, where a component must satisfy requirements exposed by its container to be plugged into a container framework, *e.g.*, the POA/ServantBase relationships in CORBA. The same argument is valid for component-based applications where a component can be substituted by another component that models the same concept. As an example of such a substitution, the CCM component description has *expected* and *provided* sections to enable automatic composability checking.

The *concept* technique alone, however, is not sufficient to achieve the goals of MDA, where platform independence, component composability, and interoperability issues are paramount. The problem lies in the existence of various mechanisms provided by containers and operating systems to achieve the same goals. For example, although both CCM and Enterprise Java Beans (EJB) containers provide access to the current execution context of the component they use different interfaces. The same is true for concurrency, IPC, and other mechanisms provided by different operating systems [13, 14].

A promising way to solve this problem is *on-demand remodularization* [15], which is the ability to identify and encapsulate new dimensions of concern at any time without invasive changes, thereby allowing manual or automatic (adaptive) selection of the best modularization based on any or all of the concerns of the development task. On-demand remodularization could be used to convert functionality provided by infrastructural software to support concepts expected by the collaborative parties. After such a conversion, for example, the collaborative components in CCM and EJB can use generative techniques for adapting the component callback interfaces to satisfy the container requirements. Moreover, on-demand remodularization enables high degree of adaptivity.

Our future work involves (1) adding remodularization support to CoSMIC MDA generators and (2) support for on-demand adaptive remodularization to CIAO containers. For

example, the CoSMIC modeling languages can be extended to allow modeling of on-demand remodularization. Based on these models, the CoSMIC generative tools can synthesize the appropriate adapters within collaborative components.

5 Concluding Remarks

The Model Driven Architecture (MDA) is an ambitious standards-based effort that aims to codify the patterns and development techniques evolved over years of R&D efforts on component middleware, Model-Integrated Computing (MIC), and related generative software technologies. Achieving the MDA vision is essential to reduce the lifecycle costs of complex DRE applications that encompass a wide range of application domains, including defense, telecommunications, medicine, process control, automotive, and manufacturing.

This paper describes our work on the CoSMIC project, which is an MDA toolsuite we are developing to integrate MIC and component middleware for DRE systems. The initial focus of CoSMIC is modeling and generating code that can statically provision key QoS properties of CIAO and QuO. In the future, we will focus on analyzing and generating model driven solutions for systems that operate in dynamic environments and hence need runtime QoS provisioning via adaptation.

In the context of the DARPA MoBIES and PCES programs, we are applying CoSMIC to several application domains, including aerospace, telecommunications, medicine, process control, automotive, and manufacturing. For example, CoSMIC is being used to model and provision adaptive QoS for a Unmanned Aerial Vehicle (UAV) application [11]. Over time, our CoSMIC toolsuite will evolve to target existing and emerging QoS-enabled middleware, such as Real-time Java and XML/HTTP-based web services.

References

- [1] Janos Sztipanovits and Gabor Karsai, "Model-Integrated Computing," *IEEE Computer*, vol. 30, no. 4, pp. 110–112, Apr. 1997.
- [2] Akos Ledeczki, Arpad Bakay, Miklos Maroti, Peter Volgysei, Greg Nordstrom, Jonathan Sprinkle, and Gabor Karsai, "Composing Domain-Specific Design Environments," *IEEE Computer*, Nov. 2001.
- [3] J. T. Buck, S. Ha, E. A. Lee, and D. G. Messerschmitt, "Ptolemy: A Framework for Simulating and Prototyping Heterogeneous Systems," *International Journal of Computer Simulation, Special Issue on Simulation Software Development Component Development Strategies*, vol. 4, Apr. 1994.
- [4] John Bay, "Recent Advances in the Design of Distributed Embedded Systems," in *Proceedings of SPIE, Volume 47: Battlespace Digitization and Network Centric Warfare*, Apr. 2002.
- [5] Jonathan M. Sprinkle, Gabor Karsai, Akos Ledeczki, and Greg G. Nordstrom, "The New Metamodeling Generation," in *IEEE Engineering of Computer Based Systems*, Washington, DC, Apr. 2001, IEEE.

- [6] 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," *Microprocessors and Microsystems*, vol. 26, no. 9-10, jan 2003.
- [7] BEA Systems, et al., *CORBA Component Model Joint Revised Submission*, Object Management Group, OMG Document orbos/99-07-01 edition, July 1999.
- [8] Nanbor Wang, Krishnakumar Balasubramanian, and Chris Gill, "Towards a real-time corba component model," in *OMG Workshop On Embedded & Real-Time Distributed Object Systems*, Washington, D.C., July 2002, Object Management Group.
- [9] Rodrigo Vanegas, John A. Zinky, Joseph P. Loyall, David Karr, Richard E. Schantz, and David E. Bakken, "QuO's Runtime Support for Quality of Service in Distributed Objects," *Proceedings of Middleware 98, the IFIP International Conference on Distributed Systems Platform and Open Distributed Processing*, September 1998.
- [10] Jeffery Gray, Ted Bapty, and Sandeep Neema, "Handling Crosscutting Constraints in Domain-Specific Modeling," *Communications of the ACM*, pp. 87–93, Oct. 2001.
- [11] Sandeep Neema, Ted Bapty, Jeff Gray, and Aniruddha Gokhale, "Generators for Synthesis of QoS Adaptation in Distributed Real-Time Embedded Systems," in *Proceedings of the ACM SIGPLAN/SIGSOFT Conference on Generative Programming and Component Engineering (GPCE'02)*, Pittsburgh, PA, Oct. 2002.
- [12] Krzysztof Czarnecki and Ulrich W. Eisenecker, "Components and Generative Programming," in *Proceedings of the 7th European Engineering Conference held jointly with the 7th ACM SIGSOFT Symposium on Foundations of Software Engineering*, Toulouse, France, 1999, ACM, pp. 2–19.
- [13] Douglas C. Schmidt and Stephen D. Huston, *C++ Network Programming, Volume 1: Mastering Complexity with ACE and Patterns*, Addison-Wesley, Boston, 2002.
- [14] Douglas C. Schmidt and Stephen D. Huston, *C++ Network Programming, Volume 2: Systematic Reuse with ACE and Frameworks*, Addison-Wesley, Reading, Massachusetts, 2002.
- [15] Mira Mezini and Klaus Ostermann, "Integrating Independent Components with On-demand Remodularization," in *To appear in the Proceedings of the 17th ACM Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA'02)*, Seattle, Washington, USA, November 2002, ACM.