Towards a Reflective Middleware Framework for QoS-enabled CORBA Component Model Applications

Nanbor Wang
Kirthika Parameswaran
{nanbor,kirthika}@cs.wustl.edu
Dept. of Computer Science
Washington University
One Brookings Drive
St. Louis, MO 63130

Michael Kircher
Michael.Kircher@mchp.siemens.de
Siemens ZT
Munich
Germany

Douglas C. Schmidt
schmidt@uci.edu
Dept. of Electrical
and Computer Engineering
University of California
616E Engineering Tower
Irvine, CA 92697

Abstract

Although existing CORBA specifications, such as Real-time CORBA and CORBA Messaging, address many end-to-end quality-of-service (QoS) aspects, they do not define strategies for configuring these QoS aspects into applications. Therefore, application developers must make these configuration decisions manually and explicitly, which is tedious, error-prone, and often sub-optimal. Although the recently adopted CORBA Component Model (CCM) does define a standard configuration framework for packaging and deploying software components, conventional CCM implementations focus on functionality rather than quality-of-service, which makes them unsuitable for applications with stringent QoS requirements.

This paper presents three contributions to the study of reflective middleware for QoS-enabled component-based applications. It outlines strategies for (1) selecting optimal communication mechanisms reflectively, (2) re-factoring QoS aspects from components into their containers to adaptively respond to changing QoS requirements and conditions, and (3) dynamically loading/unloading and activating/deactivating component implementations. Based on our ongoing research on CORBA and the CORBA Component Model, we believe the application of reflective techniques to component middleware will provide an dynamically adaptive and (re)configurable environment for COTS software that meets the stringent QoS demands of next-generation applications.

1 Introduction

Emerging trends and challenges: Distributed applications are increasingly being developed using the standard interfaces, protocols, and services defined by distributed object computing middleware, such as CORBA [1]. CORBA is a distributed object computing middleware standard that allows clients to invoke operations on remote objects without concern for where the object resides or what language the object is written in [2]. In addition, CORBA shields applications from non-portable details related to the OS/hardware platform they run on and the communication protocols and networks used to interconnect distributed objects.

An increasing number of distributed applications require middleware that is both highly flexible and configurable, as well as highly efficient, predictable, and scalable. For instance, the demand for embedded multimedia applications is growing rapidly and hand-held devices, such as PIMs, Web-phones, Web-TVs, and Palm computers, running multimedia applications, such as MIME-enabled email and Web browsing, are becoming ubiquitous [3]. Ideally, these embedded multimedia applications should be configured from standard middleware CORBA components, rather than being programmed from scratch. However, meeting the QoS demands of these applications requires the resolution of many research challenges, including handling low bandwidth, heterogeneity in the network connections, frequent changes and disruptions in the established connections due to migration, maintaining cache consistency, restrictions on physical size, and variable power consumption [4].

Distributed object computing middleware, such as CORBA, should be well-suited to provide the core communication middleware for the distributed applications outlined above. For instance, recent additions to the CORBA specification, such as Real-time CORBA [5] and CORBA Messaging [6], address many end-to-end quality-of-service (QoS) aspects. These specifications standardize interfaces and policies for defining and controlling various types of application QoS aspects.

Historically, however, the standard CORBA specification has not addressed component configuration issues. For example, the CORBA specification has not mandated standard interfaces to (1) initialize and deploy services dynamically or (2) enable different service implementations to interact portably with each other via these predefined interfaces. As a result, many "cross-cutting" [7] service implementation aspects, such as memory and bandwidth management, concurrency, dependability, and security, are tightly coupled into the application structure and behavior of CORBA servants. As a result, programming applications directly using the CORBA specification has yielded (1) brittle servant implementations that are hard to optimize, maintain, and enhance and (2) non-standardized mechanisms for bootstrapping and initializing ORB components and services [8].

To address these problems, therefore, the OMG recently adopted the CORBA Component Model (CCM) specification [9]. In theory, the adoption of CCM should reduce the effort required to portably integrate components used to implement services and applications. Moreover, CCM should simplify the reconfiguration and replacement of existing application services by standardizing the interconnection among components and interfaces.

In practice, however, the CCM standard and implementations are as immature today as the underlying CORBA standard and ORBs were three to four years ago. Moreover, commercial CCM vendors largely address the requirements of e-commerce, workflow, report generation, and other general-purpose business applications. The middleware requirements for these applications focus on functional interoperability, with little emphasis on assurance of or control over mission-critical QoS aspects, such as timeliness, precision, dependability, or minimal footprint [10]. As a result, it is not feasible to use off-the-shelf CCM implementations for QoS-enabled applications.

Solution approach $\rightarrow$ reflective middleware: Reflective middleware is a term that describes a loosely organized collection of technologies designed to manage and control hardware and software system resources based on mounting R&D experience with distributed applications and systems [11]. Reflective middleware techniques enable dynamic changes in application behavior by adapting core software and hardware mechanisms with or without the knowledge of applications or end-users [12]. We are applying the following reflective middleware techniques to our research on QoS-enabled CCM implementation to improve its flexibility, efficiency, predictability, and scalability:

• **Selecting optimal communication mechanisms reflectively:** To present a homogeneous programming model for application developers, CORBA hides the location of objects from client applications. By examining an object location's reflectively, however, a CORBA ORB can select an optimal communication mechanism automatically when it *binds* an object reference [13]. To avoid violating the CORBA object model, however, we believe that this selection should occur without direct application intervention in order to optimize middleware performance and predictability transparently.

• **Reflectively re-factoring QoS aspects from components into their containers:** In the CCM, a *container* encapsulates a *component* implementation by offering a run-time environment that provides certain functionality, such as security, event notification, transaction and persistent state services, for the component it manages. We believe that CCM containers should be extended to strategize QoS properties of containers, as well. This extension allows the ORB endsystem to support dynamic QoS configuration reflectively since it can inspect and adjust a component's QoS properties via its container. By factoring out QoS adaptation mechanisms into containers, components developers can defer the selection of QoS requirements of a component to run-time, which enhances component flexibility and adaptability.

• **Reflectively loading/unloading and activating/deactivating component implementations:** Next-generation mobile applications will increasingly run in *ad hoc* wireless networking configurations where the necessary implementation of a service component may not known *a priori*. Thus, on-demand loading/unloading mechanisms are necessary to (re)configure components dynamically. The lifecycle for loading/unloading of components must be optimized using reflective techniques in order to minimize footprint, maximize extensibility, and meet application QoS requirements more adaptively.

We are applying these reflective middleware techniques at various levels, ranging from the ORB Core up to CORBA Component Model services. The vehicle for this research is TAO [14], which is an open-source[1], CORBA-compliant ORB designed to support applications with stringent QoS requirements. Figure 1 illustrates the components in the TAO real-time ORB endsystem.

2 Concluding Remarks

Recent CORBA specifications define better support for QoS and configurability. In particular, the CORBA Component Model (CCM) [9] defines standard interfaces, policies, and services for structuring, integrating, and deploying CORBA components. Likewise, the Real-time CORBA [5] and CORBA Messaging [6] specifications address many end-to-end quality-of-service (QoS) aspects. We believe, however,

[1]The source code and documentation for TAO can be downloaded from `www.cs.wustl.edu/~schmidt/TAO.html`.

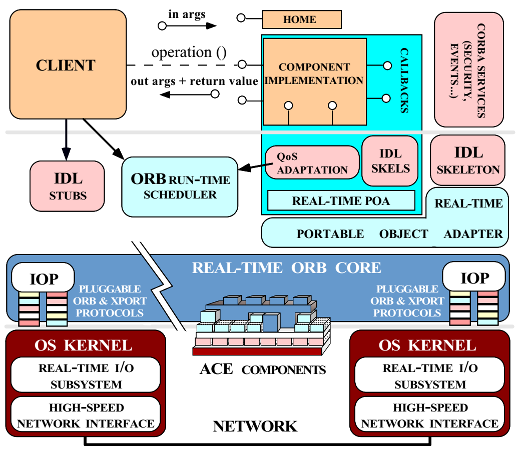

Figure 1: Components in the TAO Real-time ORB Endsystem

that these specifications will be unsuitable for an important class of QoS-enabled applications unless ORB implementations apply *reflective middleware techniques* to automate the selection and adaptation of key QoS aspects.

The reflective middleware techniques we believe are essential to improve the QoS aspects of CORBA services and applications include (1) the ability to select optimal communication mechanisms reflectively, (2) the automatic re-factoring of QoS aspects from components into their containers to adaptively respond to changing QoS requirements and conditions, and (3) on-demand loading/unloading and dynamic activation/deactivation of component implementations to support dynamic reconfiguration of systems. We are incorporating these enhancements into TAO, which is the platform we are using to implement and optimize QoS-enabled CCM.

References

[1] Object Management Group, *The Common Object Request Broker: Architecture and Specification*, 2.3 ed., June 1999.

[2] M. Henning and S. Vinoski, *Advanced CORBA Programming With C++*. Addison-Wesley Longman, 1999.

[3] G. Forman and J. Zahorhan, “The Challenges of Mobile Computing,” *IEEE Computer*, vol. 27, pp. 38–47, April 1994.

[4] A. Gokhale and D. C. Schmidt, “Optimizing a CORBA IIOP Protocol Engine for Minimal Footprint Multimedia Systems,” *Journal on Selected Areas in Communications special issue on Service Enabling Platforms for Networked Multimedia Systems*, vol. 17, Sept. 1999.

[5] D. C. Schmidt and F. Kuhns, “An Overview of the Real-time CORBA Specification,” *Submitted to IEEE Computer Magazine*, 2000.

[6] Object Management Group, *CORBA Messaging Specification*, OMG Document orbos/98-05-05 ed., May 1998.

[7] G. Kiczales, “Aspect-Oriented Programming,” in *Proceedings of the 11th European Conference on Object-Oriented Programming*, June 1997.

[8] N. Wang, D. C. Schmidt, and D. Levine, “Optimizing the CORBA Component Model for High-performance and Real-time Applications,” in *‘Work-in-Progress’ session at the Middleware 2000 Conference*, ACM/IFIP, Apr. 2000.

[9] BEA Systems, *et al.*, *CORBA Component Model Joint Revised Submission*. Object Management Group, OMG Document orbos/99-07-01 ed., July 1999.

[10] C. D. Gill, F. Kuhns, D. L. Levine, D. C. Schmidt, B. S. Doerr, R. E. Schantz, and A. K. Atlas, “Applying Adaptive Real-time Middleware to Address Grand Challenges of COTS-based Mission-Critical Real-Time Systems,” in *Proceedings of the 1st IEEE International Workshop on Real-Time Mission-Critical Systems: Grand Challenge Problems*, Nov. 1999.

[11] F. Kon and R. H. Campbell, “Supporting Automatic Configuration of Component-Based Distributed Systems,” in *Proceedings of the* 5^{th} *Conference on Object-Oriented Technologies and Systems*, (San Diego, CA), USENIX, May 1999.

[12] J. A. Zinky, D. E. Bakken, and R. Schantz, “Architectural Support for Quality of Service for CORBA Objects,” *Theory and Practice of Object Systems*, vol. 3, no. 1, 1997.

[13] M. Henning, “Binding, Migration, and Scalability in CORBA,” *Communications of the ACM special issue on CORBA*, vol. 41, Oct. 1998.

[14] D. C. Schmidt, D. L. Levine, and S. Mungee, “The Design and Performance of Real-Time Object Request Brokers,” *Computer Communications*, vol. 21, pp. 294–324, Apr. 1998.