Evaluating Real-time Publish/ Subscribe Service Integration Approaches in QoS-enabled Component Middleware*

Gan Deng, Ming Xiong and Aniruddha Gokhale
Department of Electrical Engineering and Computer Science
Vanderbilt University
Nashville, TN 37235, USA
{dengg,mxiong,gokhale}@dre.vanderbilt.edu

George Edwards†
Department of Computer Science
University of Southern California
Los Angeles, CA 37235
gedwards@usc.edu

Abstract

As quality of service (QoS)-enabled component middleware technologies gain widespread acceptance to build distributed real-time and embedded (DRE) systems, it becomes necessary for these technologies to support real-time publish/subscribe services, which is a key requirement of a large class of DRE systems. To date there have been very limited systematic studies evaluating different approaches to integrating real-time publish/subscribe services in QoS-enabled component middleware. This paper makes two contributions in addressing these key research questions. First, we evaluate the pros and cons of three different design alternatives for integrating publish/subscribe services within QoS-enabled component middleware. Second, we empirically evaluate the performance of our container-based design and compare it with mature object-oriented real-time publish/subscribe implementations. Our studies reveal that both the performance and scalability of our design and implementation are comparable to its object-oriented counterpart, which provides a key guidance to the suitability of component technologies for DRE systems.

Keywords: Real-time Publish/Subscribe services, Component Middleware.

1. Introduction

An increasing number of distributed real-time and embedded (DRE) systems require middleware support for real-time transfer of control and data among large number of heterogeneous entities that coordinate with each other in a loosely coupled fashion. Perhaps the most critical middleware service for the types of DRE systems outlined above are asynchronous, event-based publish/subscribe services [1]. Some widely used real-time publish/subscribe services for DRE systems based on *object-oriented* middleware include CORBA Real-time Event Service (RTES) [2], CORBA Real-time Notification Service (RTNS) [3] and OMG's Data Distribution Service (DDS) [4]. Recent trends indicate that *QoS-enabled component middleware*, such as CIAO [5] and PRiSM [6], are increasingly used to develop and deploy next-generation DRE systems.

The increasing use of QoS-enabled component middleware in DRE systems compounded by the need for real-time publish/subscribe services to support a large class of DRE systems requires the integration of the real-time publish/-subscribe paradigm within QoS-enabled component middleware. Unfortunately standards-based component middleware do not yet specify how publish/subscribe services can be robustly supported within component middleware. Moreover, to date there is a general lack of systematic studies that address these concerns.

This paper systematically evaluates the pros and cons of different design alternatives for integrating real-time publish/subscribe services within QoS-enabled component middleware architectures. Our study shows that the container-managed real-time publish/subscribe services provide predictable and comparable performance when compared to their object-oriented counterparts, which provides key guidance in the suitability of real-time publish/-subscribe services in component technologies for DRE systems.

Paper organization. The remainder of this paper is organized as follows: Section 2 evaluates different architectural choices and analyzes the pros and cons of each approach; Section 3 provides the performance results; and Section 4 presents concluding remarks.

2. Architectural Design Choices for Integrating Real-time Publish/Subscribe Services

In this section we describe three different design choices for integrating real-time publish/subscribe services within QoS-enabled component middleware. To make our discussions concrete, we describe these choices in the context of OMG's Lightweight CORBA Component Model (LwCCM) [7], which is an emerging component middleware standard for DRE systems and implemented by our CIAO QoS-enabled component middleware. It is worth noting that many of our discussions here are also applicable to other component models as well.

Figure 1 illustrates how these architectural choices vary based on where within the component middle-

* This work was sponsored in part by grants from NSF ITR CCR-0312859, Siemens, and DARPA/AFRL contract #F33615-03-C-4112

† Work done by the author while at Vanderbilt University

Proceedings of the 10th IEEE International Symposium on
Object and Component-Oriented Real-Time Distributed Computing (ISORC'07)
0-7695-2765-5/07 $20.00 © 2007 IEEE

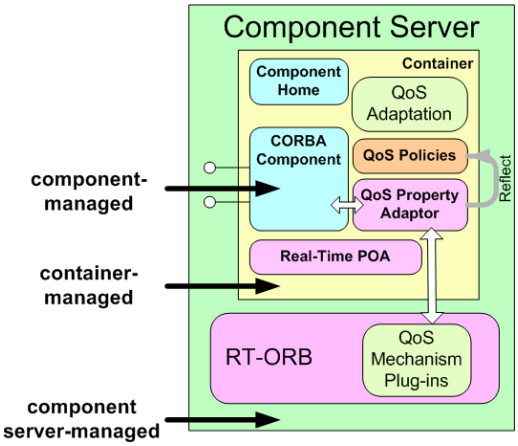

Figure 1. Architectural Choices for Integrating Publish/Subscribe Services in CCM

ware architecture does a publish/subscribe service gets integrated. The three architectural choices include (1) *component-managed* – where the event channel can be represented as an application-level component, (2) *container-managed* – where the event channel can be encapsulated within the container, and (3) *component server-managed* – where the publish/subscribe can reside within the component server. This section describes each architecture choice in detail and analyzes the advantages and disadvantages of each approach.

2.1. Component-Managed Publish/-Subscribe Services

Design: The first architecture choice for providing real-time publish/subscribe services in component middleware is to instantiate them as application-level CCM components as illustrated in Figure 2. In this architecture, the interfaces provided by the publish/subscribe services are exposed as component facet ports. These ports contain methods to connect component event sources/sinks to the event channel, configure event service real-time properties, and push events.

Analysis: The primary advantage of this approach is its simplicity. The complexity needed to implement a publish/-subscribe service component is rudimentary since the encapsulated service already implements the publish/-subscribe service functionalities, making the full set of service features readily available to other components. Instantiating and deploying multiple publish/subscribe service components follows the same rules that apply to standard components.

However, there are a number of disadvantages of using component-managed publish/subscribe mechanisms. Generally speaking, the shortcomings of the *object-oriented* model still manifest in this architecture. First, the component glue-code, or servant, must manipulate publish/-subscribe interfaces directly, which exposes low-level CORBA programmatic details thereby defeating the declarative approaches used by component middleware. Second, the component servant logic must encapsulate QoS and real-time properties, which inhibits the flexibility and reusability of components across different operating contexts and environments. Third, it is impossible to substitute or interchange different real-time publish/subscribe services without recompilation of components because the servant implementation within a component is tightly coupled with a specific type of publish/subscribe service. Finally, application level components must now be responsible for managing the publish/subscribe lifecycles.

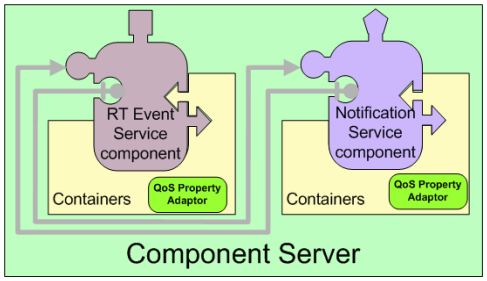

Figure 2. Publish/Subscribe Service as CCM Component

2.2. Container-Managed Publish/-Subscribe Services

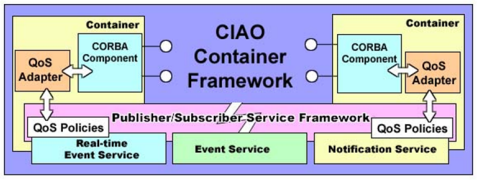

Figure 3. Publish/Subscribe Services Within Container

Design: Figure 3 depicts a second architecture for providing real-time publish/subscribe services in component middleware where a publish/subscribe service is encapsulated within the CCM container. In this architecture, the container is responsible for managing publish/subscribe service lifecycles and their clients, initializing channels and gateways, connecting publishers and subscribers, configuring QoS and real-time properties, managing publisher and subscriber component servants, and setting up the federa-

Proceedings of the 10th IEEE International Symposium on
Object and Component-Oriented Real-Time Distributed Computing (ISORC'07)
0-7695-2765-5/07 $20.00 © 2007 IEEE

tion among multiple event channels across different containers. In this design, the container exposes two distinct interfaces. One interface provides configuration methods and is invoked by the component deployment framework based on the properties specified in XML-based metadata descriptors that describe configuration decisions. The second interface provides a push method and is invoked by application-level components.

Analysis: There are many advantages in this architecture. First, since the publish/subscribe services are managed by the container, the business logic of application components are decoupled from the publish/subscribe service configuration. This decoupling enables real-time publish/subscribe service configurations and specifications to be validated and synthesized via high-level model driven engineering (MDE) tools [8] prior to system deployment time, which increases the level of abstraction and automation of the DRE system development process. This separation of concerns maximizes the flexibility and reusability of components by allowing them to be reconfigured with different QoS properties and/or services as required by new and changing operating contexts without making any changes to the application component logic or glue-code thereby obviating the need for recompilation.

Second, this design reduces the memory footprint of individual components and preserves their lightweight nature. Although the component deployment framework is exposed to the implementation details of the real-time publish/-subscribe services (since the deployment framework must instantiate and configure the channels) rather than component servant glue code, it is not important for the deployment framework to be as lightweight as the CCM components because the deployment framework is not part of the runtime system and does not consume resources after a DRE system is deployed.

Third, this architecture aligns with the CCM container programming model and defers publish/subscribe configuration-related decisions until deployment time, which allows additional optimizations to be incorporated depending on knowledge of the deployment context. For example, it may not be known until deployment time which network links have high latency or low reliability, yet this information is critical to determining the best possible real-time publish/subscribe service configuration.

The disadvantage to the container-managed event channel architecture is the difficulty encountered in actually implementing it effectively and efficiently due to the complexity of CCM container architecture and its programming model. There are a number of design challenges that arise when pursuing this design choice descried in detail in [9].

2.3. Component Server-Managed Publish/Subscribe Services

Design: The third alternative architecture for providing real-time publish/subscribe services in component middleware is to host them within the component server, which is similar to existing approaches of supporting services in object-oriented middleware. In this architecture, publish/-subscribe services are still accessed and manipulated via the container. However, the component server-managed architecture is fundamentally different from the container-managed architecture in that the component server is a lower-level entity which hosts all the components, which in turn end up sharing the same publish/subscribe service.

Analysis: The advantages present in the container-managed architecture are also applicable to the component server-managed architecture: components are still isolated from publish/subscribe services in such a way that they remain configurable after compilation, and push operations result in only local method invocations. However, the component server-managed architecture is more coarse-grained *i.e.*, a large number of components may be required to share a single service thereby affecting differentiated treatment to application components depending on their real-time needs.

For applications that require either multiple publish/-subscribe services on a single host or those who wish to maximize component flexibility to allow for future enhancements or modifications, the component server-managed architecture may be too restrictive. On the other hand, for applications that do not require these capabilities, the component server-managed architecture results in a simpler configuration and deployment process, which reduces development effort. In the case of very large-scale DRE systems, the savings may be substantial if sharing is desired, however, for DRE systems that require partitioning and configuring the system capabilities based on priorities, load balancing and reliability, this coarse-grained approach is not suitable.

Summary: Based on our analysis of the pros and cons of each design choice, we have selected the *container-managed* architecture as our design choice to obtain additional guidance on its applicability and performance.

3. Empirical Performance Evaluation

This section provides empirical results for the container-managed CORBA real-time event service (RTES) integrated within our CIAO QoS-enabled component middleware. We choose RTES as a vehicle to evaluate our design because it is a mature real-time publish/subscribe implementation based on real-time CORBA and has been widely used in many DRE systems [10].

All our benchmarks were conducted on ISISlab (`www.isislab.vanderbilt.edu`). ISISlab consists of 6 Cisco 3750G-24TS switches, 1 Cisco 3750G-48TS

Proceedings of the 10th IEEE International Symposium on
Object and Component-Oriented Real-Time Distributed Computing (ISORC'07)
0-7695-2765-5/07 $20.00 © 2007 IEEE

switch, 4 IBM Blade Centers. Each blade has two 2.8 GHz Xeon CPUs, 1GB of RAM, and runs Fedora Core 4 Linux, version 2.6.16-1.2108_FC4smp. All our benchmark applications were run in the Linux real-time scheduling class to minimize extraneous sources of memory, CPU, and network load. For each test, we run the iteration at least 10,000 times.

The most important metric for publish/subscribe service is the *event latency*, i.e., the time elapsed from when a publisher sends an event until the last subscriber interested in the event receives it.

Latency Results for Process Collocated Event Processing. In this test all the event publishers, subscribers, and the event channel are collocated in the same process, which eliminates effects of ORB remote communication overhead. In the container-managed RTES case, both the publisher and subscriber components are deployed into the same container which in turn is hosted in a single CIAO component server. The end-to-end latency is determined by the publisher sending out the timestamp right before the `push` call and subsequently the subscribers calculating the difference between the timestamp at the publisher side and the subscriber side. We also use variable-sized octet sequence as the payload so that we can easily control the volume of the payload.

One important fact concerns the event data type is that since all the event source and event sink ports of CCM components are defined as `Eventtype`, which is a specialized `valuetype`, it is unavoidable to eliminate the additional overhead incurred due to marshaling/demarshaling of such a data type. To ensure a fair comparison between the performance of TAO's RTES and CIAO's container-managed RTES, we send `valuetype` data in both test cases, and make the octet sequence payload as the member of this data type.

Figure 4 shows the latency results for the *one-to-many* case. In this test, we increase both the number of subscribers and the payload size to see how the end-to-end latency is affected.

Analysis. Our collocation experiment results indicate that the event dispatching overhead in CIAO's container-managed RTES incurs about 20∼25% latency performance overhead consistently over the TAO's RTES implementation. This overhead is primarily due to two reasons. First, in contrast to programming model of CORBA RTES, where the publisher *CORBA objects* and the subscriber are all plain *CORBA objects* that can directly interact with the event channel as RTES clients, the OMG CCM standard defines a much more sophisticated architecture that introduces multiple levels of indirections. For example, when a component publishes an event, the component implementation, *i.e.,* the *component executors*, must create a CCM event and make a call to its *context* object. A component context object is used by a component instance to interact with its execution environment. The component context object, in turn, delegates the call to the *component servant object*, which is boiler plate like code automatically generated by the CIDL compiler [7] to implement the component interfaces. The component servant object will then identify the appropriate event channels to push events. Second, the performance overhead is also partially attributable to the the additional levels of indirection introduced with the higher-level container mediation interface, which implements the adapter pattern and wrapper facade pattern to allow different publish/subscribe services to be plugged into the container seamlessly. More information about our design approach can be found in [9].

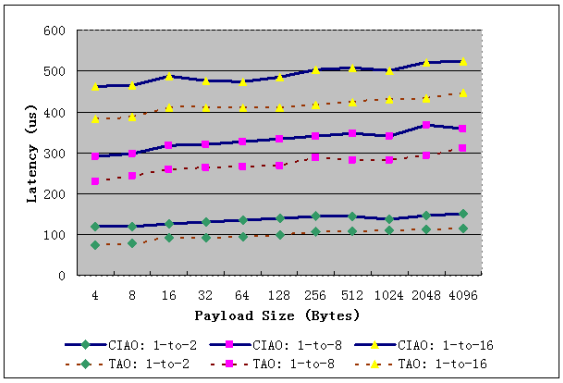

Figure 4. Collocated Event Latency Results

While it is inevitable to avoid the overhead caused by the indirection defined in the OMG CCM standard without breaking standards-based interfaces, we applied process-collocation optimization to CIAO's implementation to improve the performance and predictability of collocated component communication. The process-collocation optimization we conducted improves the performance and predictability for objects that reside in the same address space as the servant implementation, while maintaining locality transparency.

Our process-collocated experiment results demonstrate that the performance optimizations of object-oriented real-time event dispatching are preserved within a container-based solution. It is also interesting to observe that as the number of subscribers increase, the increase in latency is less than linear in both TAO's CORBA RTES implementation and CIAO's container-managed RTES implementation, due in large part to the Handle/Body idiom used to optimize the processing of CORBA `Any` data types in TAO's RTES implementation [11], which is inherited by CIAO RTES. This idiom presents multiple logical copies of the same data while sharing the same physical copy.

Proceedings of the 10th IEEE International Symposium on
Object and Component-Oriented Real-Time Distributed Computing (ISORC'07)
0-7695-2765-5/07 $20.00 © 2007 IEEE

Latency Results for Remote Event Processing. [1] In this test, we run the experiment by creating two different processes within a single node to allow the events to be sent remotely. The event publisher and the event channel are collocated in one process, while the subscribers are in the other process. Figure 5 show the results.

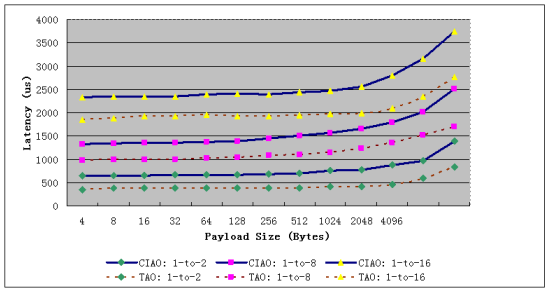

Figure 5. Two Process One-to-Many Latency

Analysis. The results indicate that the remote event processing latency in both CIAO's container-managed RTES and TAO's RTES are much higher than that of process-collocated cases (both incurring about about $6 \sim 10$ times overhead) due to the process boundary crossing, though they are close to the performance of a remote operation invocation. With increasing number of event subscribers and payload size, the results still show that the event dispatching performance in container-managed RTES in CIAO consistently has about 75∼80% performance of TAO's RTES in all test cases. In conjunction with the results of process-collocated tests, these results further confirm that the CIAO container-based RTES solution has predictable performance which is comparable with TAO's RTES, and is thus suitable for DRE systems.

4. Concluding Remarks

Component middleware has already received widespread acceptance in the enterprise business and desktop application domains. However, developers of DRE systems have encountered limitations with the available component middleware platforms, such as the CCM and J2EE. In particular, component middleware platforms lack standards-based real-time publish/subscribe communication mechanisms that support key QoS requirements of DRE systems, such as low latency, bounded jitter, and end-to-end event propagation. This paper provides guidance to establish the feasibility of integrating mature object-oriented real-time publish/subscribe services in QoS-enabled component middleware architectures. The lessons we learned in this study indicate that component middleware standards often introduce some event dispatching overhead compared to the object-oriented middleware. For example, the CCM standard inevitably introduces some performance overheads for event dispatching caused mainly by valuetype based CCM Eventtype marshaling/demarshaling costs and some levels of indirection between different entities including component executors, servants, contexts and containers. It is worth noting that discussion of these overheads is orthogonal to the focus of this paper.

1 In this test configuration, a "remote" event is one intended for a subscriber located in the other process.

References

[1] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal, *Pattern-Oriented Software Architecture—A System of Patterns*. New York: Wiley & Sons, 1996.

[2] T. H. Harrison, D. L. Levine, and D. C. Schmidt, "The Design and Performance of a Real-time CORBA Event Service," in *Proceedings of OOPSLA '97*, (Atlanta, GA), pp. 184–199, ACM, Oct. 1997.

[3] P. Gore, D. C. Schmidt, C. Gill, and I. Pyarali, "The Design and Performance of a Real-time Notification Service," in *Proceedings of the 10th Real-time Technology and Application Symposium (RTAS '04)*, (Toronto, CA), IEEE, May 2004.

[4] Object Management Group, *Data Distribution Service for Real-time Systems Specification*, 1.0 ed., Mar. 2003.

[5] N. Wang, C. Gill, D. C. Schmidt, and V. Subramonian, "Configuring Real-time Aspects in Component Middleware," in *Lecture Notes in Computer Science: Proc. of the International Symposium on Distributed Objects and Applications (DOA'04)*, vol. 3291, (Agia Napa, Cyprus), pp. 1520–1537, Springer-Verlag, Oct. 2004.

[6] W. Roll, "Towards Model-Based and CCM-Based Applications for Real-time Systems," in *Proceedings of the International Symposium on Object-Oriented Real-time Distributed Computing (ISORC)*, IEEE/IFIP, May 2003.

[7] Object Management Group, *Lightweight CCM FTF Convenience Document*, ptc/04-06-10 ed., June 2004.

[8] G. Edwards, G. Deng, D. C. Schmidt, A. Gokhale, and B. Natarajan, "Model-driven Configuration and Deployment of Component Middleware Publisher/Subscriber Services," in *Proceedings of the Third International Conference on Generative Programming and Component Engineering (GPCE)*, (Vancouver, CA), ACM, Oct. 2004.

[9] G. Deng, M. Xiong, A. Gokhale, and G. Edwards, "Evaluating Real-time Publish/Subscribe Service Integration Approaches in QoS-enabled Component Middleware," Tech. Rep. ISIS-07-804, Vanderbilt University, February 2007.

[10] C. O'Ryan and D. C. Schmidt, "Applying a Real-time CORBA Event Service to Large-scale Distributed Interactive Simulation," in 5^{th} *International Workshop on Object-oriented Real-time Dependable Systems*, (Monterey, CA), IEEE, Nov. 1999.

[11] D. C. Schmidt and C. O'Ryan, "Patterns and Performance of Real-time Publisher/Subscriber Architectures," *Journal of Systems and Software, Special Issue on Software Architecture - Engineering Quality Attributes*, 2002.

Proceedings of the 10th IEEE International Symposium on
Object and Component-Oriented Real-Time Distributed Computing (ISORC'07)
0-7695-2765-5/07 $20.00 © 2007 IEEE