The Design and Performance of a Real-Time Notification Service

Pradeep Gore and Irfan Pyarali
OOMWorks, LLC
Metuchen, NJ
{pradeep,irfan}@oomworks.com

Christopher D. Gill
Washington University
St. Louis, MO
cdgill@cse.wustl.edu

Douglas C. Schmidt
Vanderbilt University
Nashville, TN
d.schmidt@vanderbilt.edu

Abstract

Many distributed real-time and embedded (DRE) applications require a scalable event-driven communication model that decouples suppliers from consumers and simultaneously supports advanced quality of service (QoS) properties and event filtering mechanisms. The CORBA Notification Service provides publisher/subscriber capabilities designed to support scalable event-driven communication by routing events efficiently between suppliers and consumers, enforcing QoS properties (such as reliability, priority, ordering, and timeliness), and filtering events at multiple points in a distributed system. The standard CORBA Notification Service is insufficient, however, to enforce predictable communication needed by DRE applications and does not leverage Real-time CORBA capabilities, such as end-to-end priority assignment or scheduling services.

This paper makes three contributions to the study of scalable real-time notification services for DRE applications. First, we describe the requirements of the OMG Request for Proposals (RFP) on Real-time Notification, which seeks solutions to the problem of enforcing real-time properties by enhancing the standard CORBA Notification Service. Second, we explain how we have addressed key design challenges faced when implementing a Real-time Notification Service for TAO, which is our CORBA-compliant real-time Object Request Broker (ORB). We discuss how we integrate Real-time CORBA features (such as thread pools, thread lanes, and priority models) to provide real-time event communication. Finally, we analyze the results of empirical benchmarks of the performance and predictability of TAO's Real-time Notification Service. These results show that the static real-time assurances provided by Real-time CORBA are maintained within the more flexible context of TAO's Real-time Notification Service.

Keywords: Distributed real-time embedded (DRE) systems, Quality of Service (QoS), CORBA, Event / Notification Services.

1. Introduction

Many distributed real-time and embedded (DRE) applications (such as real-time avionics mission computing systems, distributed interactive simulations, and computer-assisted stock trading) require an event-based communication model. Client/server communication via distribution middleware (such as CORBA, Java RMI and COM+) typically support a synchronous method invocation (SMI) model, where a client invokes a two-way operation on a target object implemented by a server and then blocks waiting for the response. This model has limitations, however, stemming from its tight coupling between client and server lifetimes, synchronous communication, and point-to-point communication.

The deficiencies of the SMI model can be resolved by publisher/subscriber services that support the decoupling of event suppliers and consumers, asynchronous communication, and transparent group communication. For example, the CORBA Event Service [5] introduces a standard object model consisting of an event channel that acts as a broker between anonymous event suppliers and consumers connected to the event channel via proxy objects. Likewise, the CORBA Notification Service 0 is an extension to the CORBA Event Service that provides (1) greater scalability via event filters, (2) simplified administration via standard event channel factory support, shared subscription information, and the ability to navigate the event channel object hierarchy, (3) resource management via QoS properties for reliability, event priority, and internal ordering/discarding of events, and (4) improved usability via support for three different types of events (CORBA Anys, structured events, and batch events) and a constraint language (ETCL) for specifying filters.

Although the event-based publisher/subscriber CORBA middleware has been implemented and studied widely [6][7][8], the original CORBA Event and Notification Services have drawbacks that limit their applicability to applications with stringent QoS requirements, such as real-time deadline assurance [9][10]. For example, the existing CORBA publisher/subscriber services do not address the requirements of real-time event communications that requires timeliness and predictability when delivering events from suppliers to event consumers via event channels. Likewise, they do not use the priority and scheduling capabilities defined in the Real-time CORBA 1.0 [2] and 2.0 [3], respectively. Though Real-time CORBA provides end-to-end QoS support for point-to-point communication via prioritized operation calls, it does not address end-to-end QoS guarantees for anonymous event-based communication. To overcome these limitations, therefore, the OMG has issued a Request for Proposals (RFP) [4] to provide enhancements that will support a Real-time Notification Service, which must satisfy the following requirements:

Limit the complexity of filters. Events that do not match the constraints specified in the filter are discarded. A filtering expression can be arbitrarily complex. Submissions are required to provide mechanisms and describe the interfaces by which the filtering expressions could be minimized by limiting the number of different constraints in a filter, specifying the length of a message that might be successfully evaluated by a filter, and/or limiting how many different event types a supplier can produce.

Subset functionality for real-time behavior. Submissions must document which portions of the Real-time Notification Service address predictable resource management and timeliness predictability.

Describe the schedulable entities. Submissions must describe the schedulable entities (*e.g.*, event messages) that would participate in the operation of the Real-time Notification Service. Also, both the Real-time CORBA 1.0 CLIENT_PROPAGATED and SERVER_DECLARED priority models must be supported.

Priority aware end-to-end event propagation. Real-time CORBA 1.0 describes the priority levels at which participant objects are executed. Submissions are required to support these priority levels so that when the events are propagated across hosts, the event priority is considered in the processing.

Provide means to set real-time QoS parameters. DRE application QoS parameters (*e.g.*, priorities) may need to be communicated end-to-end. Submissions are required to provide mechanisms for this, along with a means to set real-time QoS parameters at the channel, connection, proxy, and message levels.

Interface for resource management. Real-time CORBA 1.0 provides interfaces to manage resources, such as the number of threads in a thread-pool. Similarly, submissions are therefore required to provide an interface that can help manage real-time notification resources, such as event channels.

Interface to support interactions with a Scheduling Service. Some real-time QoS requirements on a Notification Service might need to be checked via a global Scheduling Service. Submissions are required to define an appropriate interface to support interactions with the Real-time CORBA 1.0 Scheduling Service.

Since the Real-time Notification Service is still undergoing adoption via the OMG standardization process, no commercial products or research prototypes are available for it yet. To facilitate the study of real-time publisher/subscriber services, therefore, we have implemented a prototype of the latest revised submission [2] that enhances on our prior work with The ACE ORB (TAO) [12] and its Real-time Event Service [9][13] and Notification Service [8]. Our new prototype implements the real-time extensions to the existing CORBA Notification Service designated in the revised submission to support predictable end-to-end event communications. This prototype has been integrated with the TAO open-source software release and is available from `deuce.doc.wustl.edu/Download.html`.

The remainder of this paper is organized as follows: Section 2 summarizes the key components in the OMG CORBA Notification Service, which provides the baseline for our work; Section 3 describes the design of TAO's Real-time Notification Service, focusing on how we resolved the key challenges faced in meeting the Real-time Notification RFP requirements outlined above; Section 4 analyzes the results of empirical benchmarks that illustrate the end-to-end predictability and scalability of our solution; and Section 5 presents concluding remarks and describes future work.

2. CORBA Notification Service Overview

This section describes the architecture of the core components in the standard CORBA Notification Service, which forms the basis for the Real-time Notification Service described in this paper. As Figure 1 shows, this architecture is similar to the that of the CORBA Event Service, though some components have a broader range of capabilities in the Notification Service. These components are as follows.

Structured events, which define a standard data structure into which event message can be stored. As shown in Figure 2, the header of a structured event consists of type information and a variable header, which can carry the QoS properties of an event. The event body consists of filterable body fields, followed by the payload data.

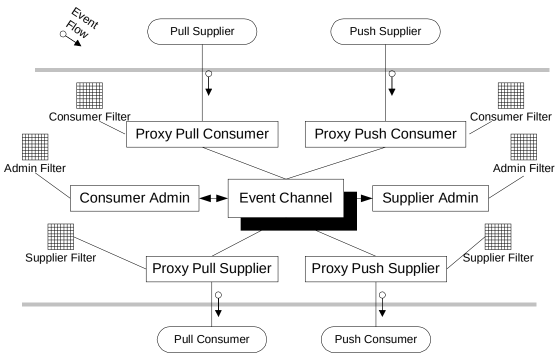


Figure 1: CORBA Notification Service Components

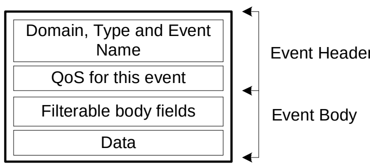


Figure 2: Header of the Structured Event

Proxy objects, which are delegates that provide complementary interfaces to clients, i.e., a consumer obtains and connects to a **ProxySupplier** and a supplier obtains and connects to a **ProxyConsumer**. Hence, a supplier sends events to its **ProxyConsumer**, whereas a consumer receives events from its **ProxySupplier** objects allow anonymous connectivity and communication between consumers and suppliers.

Admin objects, which are factories that create the proxy interfaces to which clients will connect. **ConsumerAdmins** create **ProxySuppliers** to which consumers connect, while **SupplierAdmins** create **ProxyConsumers** to which the suppliers connect.

The CORBA Notification Service treats each **Admin** object as the manager of the group of proxies it has created. **Admin** objects can themselves have QoS properties and **Filter** objects associated with them. The QoS properties associated with an **Admin** object are assigned to the **Proxy** objects that the **Admin** creates, but can be tailored subsequently on a per-proxy basis. The set of **Filter** objects associated with a given **Admin** is treated as a unit, which applies at all times to all **Proxy** objects that the **Admin** creates.

Filter objects, which can be associated with all **Admin** and **Proxy** objects and used to encapsulate a set of constraints that affect the event forwarding decisions made by **Proxy** objects. Each constraint consists of (1) a sequence of event types and (2) a string containing a Boolean expression whose syntax conforms to a constraint grammar.

EventChannels, which are factories that create **ConsumerAdmin** and **SupplierAdmin** objects. The **EventChannels** defined by the CORBA Notification Service differ slightly from the CORBA Event Service **EventChannels**, which only have one instance of each **Admin** object. QoS and administrative properties can be set on an **EventChannel** during its creation. These parameters are passed as default values to any **Admin** object created by the **EventChannel**. Consumers and suppliers can change these parameters later to tailor the properties for their specific requirements. **EventChannelFactory**, which creates **EventChannels**.

3. Real-Time Notification Service Design

This section describes the design of TAO's Real-time Notification Service, whose key components are shown in Figure 3. These components and their capabilities are as follows:

- **EventChannel component**, which is the CORBA interface presented to supplier and consumer participants. The **EventChannelFactory**, **EventChannel**, **ConsumerAdmin**, **SupplierAdmin**, **ProxyConsumer** and **ProxySupplier** (for CORBA any, structured, and sequence events) interfaces in the Notification Service give access to this component.
- **Subscription lookup component,** which consists of the subscription map of consumers to event types and the publication map of suppliers to event types. The subscription map is consulted to obtain the list of consumers subscribed to an event. This component also provides subscription/publication update messages to participants when the respective map is modified.
- **Filtering component**, which provides the filtering support to check if an event matches the filtering constraints in its path of event propagation within an event channel. It provides interface support for the **Filter** and **FilterAdmin** interfaces of the Notification Service. The OMG standard ETCL filtering language is supported via a parser component.
- **Dispatching component**, which consists of a buffer queue that provides support for the *Order* and *Discarding Policy* QoS properties. Event buffering is not used by the real-time configuration of TAO's Notification Service, however, because the Real-time CORBA 1.0 specification lacks the necessary means to control the buffer. The **Dispatching** component offers the *Pacing* and *BatchSize* properties to support event batching in sequence consumers.

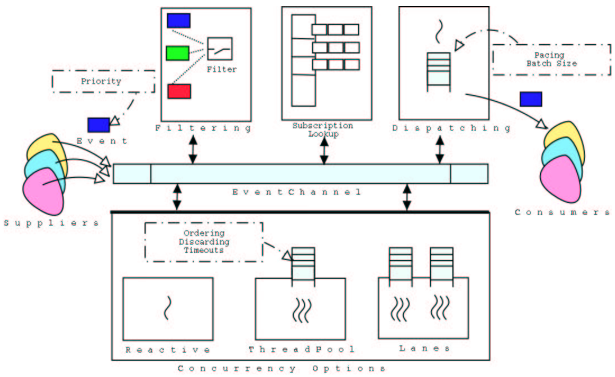


Figure 3: TAO's Real-time Notification Service

In general, the enhancements provided in TAO's Real-time Notification Service support:

- **Priority-aware, end-to-end event propagation.** By leveraging RT-CORBA 1.0 features, event priorities are maintained and respected along the entire path of event propagation from event supplier to event consumers, which provides DRE applications with an end-to-end, priority aware publisher/subscriber service.
- **Administration of Concurrency options.** Extensions to the **Proxy** interfaces provide support for the configuration of concurrency within the Real-time Notification Service. **ThreadPool/Lanes** parameters can be applied at the event channel, admin, and proxy levels of the Real-time Notification Service object hierarchy.

The remainder of this section describes how we resolved the following challenges faced when meeting the Real-time Notification Service requirements described in Section 1:

- Limiting filter complexity
- Ensuring end-to-end priority preservation
- Supporting real-time thread pools
- Optimizing Event Processing
- Minimizing context switching between supplier and consumer proxies, and

For each challenge, we describe the context in which the challenge arises, identify the specific problem that must be addressed, describe our solution for resolving the challenge, and explain how this solution was applied to TAO's Real-time Notification Service. Section 4 then presents the analysis of empirical benchmarks that illustrate the end-to-end predictability and scalability of our solutions.

3.1. Limiting Complexity of Filters

Context: Filter objects can be applied at the proxy and admin levels of the Notification Service hierarchy. The length of the constraints specified in a filter is unbounded. Moreover, filters can be changed dynamically and a filter can be a remote object. The RFP seeks to limit the complexity of such filters so that DRE applications do not incur unbounded filter evaluation overhead. The goal is to allow an application developer to set a useful bound on the time needed to evaluate a filter for an event.

Problem: Without bounds on the number of filters or the complexity of evaluating each filter, filter processing itself could consume an excessive and unpredictable amount of time, leading to deadline failures for delivery and processing of notifications.

Solution: Timeouts. Timeouts can be used to ensure that filters don't take too long to run, thereby ensuring that other deadlines aren't violated. The CORBA messaging extensions provide a `RelativeRoundtripTimeoutPolicy` that can be applied to a two-way invocation to specify how much time is allowed to deliver a request and its reply.

Applying the solution to TAO's Real-time Notification Service. TAO Real-time Notification Service provides two different mechanisms for applying timeouts to ensure filter processing times are bounded. First, a supplier can map an event deadline to a timeout QoS parameter of a structured event. TAO's Real-time Notification Service can then use the `RelativeRoundtripTimeoutPolicy` from the CORBA `Messaging` interface when making an invocation to a filter object where the timeout value equals the time left for the event to reach its destination. Second, the **Filter** interface can be extended such that the `Filter::match()` operation accepts timeouts, *i.e.*:

```
interface TimeoutFilter : Filter {
 boolean match_with_timeout (in any
filterable_data, TimeBase::UtcT timeout)
    raises  (UnsupportedFilterableData);
 boolean match_structured_with_timeout (
        in CosNotification::StructuredEvent
        data, TimeBase::UtcT timeout)
    raises (UnsupportedFilterableData);
};
```

This alternative pushes responsibility for implementing timeouts to the filter object developer.

3.2. Ensuring End-to-end Priority Preservation

Context: When an event enters a Real-time Notification Service event channel, it carries a priority. Event propagation mechanisms must ensure that the priority at which the event is processed is maintained consistently and correctly as the event traverses the path from supplier to consumer(s).

Problem: Maintaining a per-priority path from supplier to consumer(s) through the standard Notification Service preserves end-to-end priority. This priority preservation is only assured, however, across each instance of the Notification Service implementation, and is not enforced across the remote invocations between these instances. We therefore need to maintain per-priority paths *end-to-end*.

Solution: Real-time CORBA CLIENT_PROPAGATED priority model. Real-time CORBA 1.0 gives the CLIENT_PROPAGATED priority model that an application can use to convey the priority of a supplier thread to the thread processing the event in the Real-time Notification Service. Likewise, Real-time CORBA 1.0 *thread lanes* can be used to configure event paths based on priority.

Applying the solution to TAO's Real-time Notification Service: As shown in Figure 4, consumer and supplier proxies are activated in real-time POAs associated with Real-time CORBA thread pools. The priority of the supplier thread pushing events to the Real-time Notification Service will match the priority of the event sent by the supplier. The proxy consumer in a thread of matching priority will service this event. After being filtered by the Real-time Notification Service, the event will be delivered to the consumer by the proxy supplier in a thread of matching priority. Finally, the consumer in a thread of matching priority will process the event. The proxy consumer executes the consumer admin and proxy level filters and queries a lookup table to retrieve a list of consumers subscribed to receive the event. The proxy supplier executes the supplier admin and proxy level filters and delivers the event to the consumer. A proxy supplier can be configured with its own thread pool.

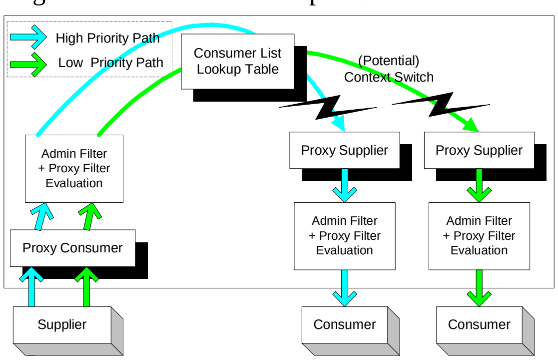


Figure 4: Prioritized Event Propagation Path

3.3. Support for Real-time Thread Pools

Context: The Real-time Notification Service RFP [4] requires submissions to define schedulable entities and to support the CLIENT_PROPAGATED and SERVER_DECLARED priority models defined by Real-time CORBA 1.0 [2].

Problem: The OMG Notification Service does not specify a mechanism for specifying policies to the POA in which the proxy objects are activated. Since the Real-time CORBA 1.0 thread pool, thread pool with lanes, and priority model policies are specified on POA objects, a mechanism is needed to express these policies in the POAs for the proxy objects.

Solution: Use QoS properties to specify POA policies. Specifying new QoS properties for the Real-time Notification Service enables support for Real-time CORBA 1.0 features. These properties can be applied to POAs at multiple levels, i.e., event channel, admin, and proxy.

Figure 5 shows the thread pool policy applied to POA's that exist at these three levels. All admin and proxy objects share the event channel-level thread pool. The admin-level thread pool is only available to proxies that are created by that admin. The proxy-level thread pool is only available exclusively to the proxy with which the thread pool is associated.

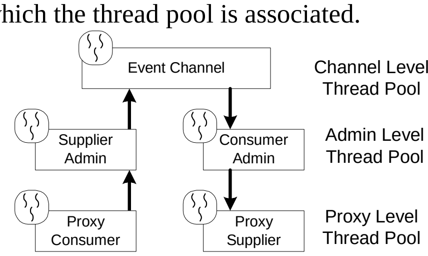


Figure 5: RT Notification Service Thread Pools

Applying the solution to TAO's Real-time Notification Service: When a QoS property specifying the POA policy is set on the on a TAO Real-time Notification Service event channel, admin or proxy a POA is created and the POA policies are applied to this new POA. This POA is used exclusively to activate the proxy objects.

Figure 6 shows several possible thread pool configurations in the Real-time Notification Service. The consumer-side of the Real-time Notification Service is configured as follows: **ProxyConsumer** 1 is associated with **ThreadPool** C. **ThreadPool** C is used only by **ProxyConsumer** 1. **ProxyConsumer** 2 is associated with **ThreadPool** B. **ProxyConsumer** 2 does not have its own exclusive **ThreadPool**: it therefore uses the **ThreadPool** from **SupplierAdmin** 1. **ProxyConsumer** 3 is associated with **ThreadPool** A. **ProxyConsumer** 3 does not have its own exclusive **ThreadPool** and neither does **SupplierAdmin** 2. It therefore uses the **ThreadPool** from the **EventChannel**.

The supplier-side is configured similarly: **ProxySupplier** 1 is associated with **ThreadPool** E. **ThreadPool** E is exclusively used by **ProxySupplier** 1. **ProxySupplier** 2 is associated with **ThreadPool** D. **ProxySupplier** 2 does not have its own exclusive

ThreadPool. It therefore uses the **ThreadPool** from **ConsumerAdmin** 1. **ProxySupplier** 3 is associated with **ThreadPool** A. **ProxySupplier** 3 does not have its own exclusive **ThreadPool** and either does **ConsumerAdmin** 2. It therefore uses the **ThreadPool** from the **EventChannel**.

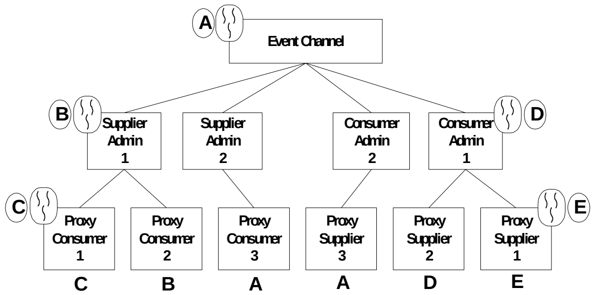


Figure 6: RT Notification Service Thread Pool Configurations

Typically, an administrative application with knowledge of the overall deployment scenario of the Real-time Notification Service would specify QoS properties to control its thread resource usage at initialization. Similarly, the CLIENT_PROPAGATED and SERVER_DECLARED priority models can be specified by another QoS property. The Real-time CORBA 1.0 *thread pool with lanes* feature is also specified in the same manner.

3.4. Optimizing Event Processing

Context: During the processing of events within an event channel, the event type of the incoming event is matched in a *Subscription Lookup Table* to determine the set of consumers that are subscribed to receive that event. The lookup operation simply reads the table. Conversely, administrative methods write to this table to update subscription information.

Problem. Since multiple threads can access the *Subscription Lookup Table*, it is necessary to serialize access to it. A mutex lock causes a *read* operation to wait while other *read/write* operations are in progress. In a typical Real-time Notification Service deployment, however, the number of *push* operations is far greater than the number of subscription change operations. Using a simple mutex can therefore unnecessarily reduce concurrent access to data structures in the critical path of event propagation.

Solution: Readers/writer Lock. A readers/writer locks allow any number of threads to hold a lock for reading as long as no thread holds the lock for writing. A thread can hold the lock for writing only if no thread holds the lock for reading or writing.

Applying the solution to TAO's Real-time Notification Service. The *Event Map* data structure used to maintain subscription information in the *Subscription Lookup Table* has a strategized locking policy that uses a readers/writer lock to give preferred access to the event propagation threads.

3.5. Minimizing Context Switching Between Consumer and Supplier Proxies

Context: Event propagation may need to switch from a proxy consumer thread pool to a proxy supplier thread pool if the two proxies are in different thread pools. Such a configuration may be required to assign a dedicated thread pool to dispatch events to a consumer.

Problem: If the proxy consumer and proxy supplier are activated in separate POA's, there are no interfaces that can be used to transfer the thread of execution from a thread in the proxy consumer to the proxy supplier.

Solution: Use an internal interface to exploit the ORB's collocation mechanism. We exploit the fact that when a CORBA interface operation is invoked, the thread pool configured at the POA in which the interface is activated processes the request. As the **ProxySupplier** interface does not support any operation to accept an event, it must be extended so that the **ProxyConsumer** can forward the event to the extended interface.

Applying the solution to TAO's Real-time Notification Service: We define an internal interface, **Event_Forwarder** that extends the **Structured ProxyPushSupplier** interface. The **Event_Forwarder** interface supports the operation signature **void forward (in CosNotification::StructuredEvent event)**. The **ProxySupplier** interface implements the **Event_Forwarder** interface. When a proxy consumer needs to switch execution context to the proxy supplier, it simply calls the **Event_Forwarder::forward()** operation, passing it the event. The ORB collocation mechanism ensures that execution switches to the thread configured in the proxy supplier. If the proxy supplier is configured in the same thread pool as the proxy consumer, however, no context switch occurs.

4. Analysis of End-to-end Predictability

This section describes experiments we conducted to validate TAO's Real-time Notification Service prototype described in Section 3. We first describe the testbed environment and experimental benchmarks used for our experiments. We then present and analyze the results of experiments conducted to compare the throughput of TAO's Real-time Notification Service implementation with TAO's standard Notification Service implementation as the load is increased and the number of supplier-to-consumer paths is increased.

We also assess the overhead of TAO's Real-time Notification Service implementation by comparing its jitter to that of (1) the standard Notification Service, (2) a direct Real-time CORBA 1.0 priority connection, and (3) a standard CORBA connection that does not use Real-time CORBA priority features.

4.1. Testbed Environment and Benchmarks

All the experiments were conducted in the Emulab [14] testbed at the University of Utah (http://www.emulab.net). The results presented in the following subsections were obtained on 1 PC with all suppliers, all consumers, and the Notification Service each in separate processes. Similar results were observed by running each supplier and consumer on a separate PC. Each PC was a 800 MHz Pentium3 processor, with 256KB on-chip cache and 239MB of free RAM memory, running RedHat Linux 7.1 in the real-time scheduling class. The benchmarking programs were compiled using the GCC compiler version 2.96, with all optimizations enabled. TAO version 1.3.4 was used for all tests.

4.2. Load vs. Throughput

Overview. In this experiment, an exclusive high priority *path* is set up between a supplier/consumer pair, as shown in the Figure 7.

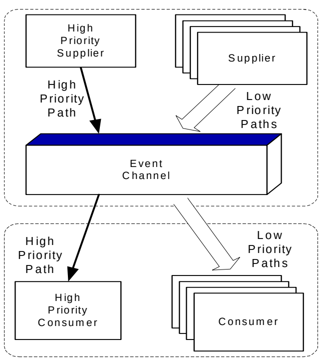


Figure 7: Exclusive High-priority Path Setup

To achieve this, the consumer subscribes to a type that is supplied only by the high-priority supplier. The proxy consumer for the high-priority supplier is activated in a real-time POA with 1 high-priority thread lane. The supplier uses the CLIENT_PROPAGATED priority model to send events to the consumer. Likewise, medium-priority and low-priority paths are also established. Each supplier sends events every 10 ms, i.e., at 100 Hz. The experiment was performed using both the RT-Notification and CosNotification services by increasing the amount of CPU intensive work performed by each consumer and measuring the throughput obtained. The "load" is a positive count that is supplied with the event payload. A prime number calculation is performed proportional to the supplied load value.

Results. Figure 8 shows that with CosNotification, the throughput of the high-, medium-, and low-priority paths decreased proportionally with increasing load.

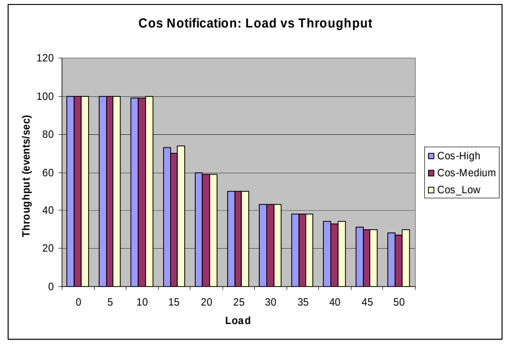


Figure 8: CosNotification: Load vs Throughput

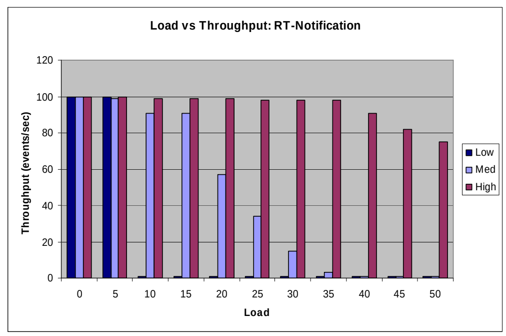


Figure 9: RT-Notification: Load vs Throughput

Figure 9 shows that with RT-Notification, under increasing load the throughputs of the high-, medium-, and low-priority paths are maintained preferentially according to their specified priorities, i.e., low falls off first, followed by medium, and then high. These results indicate how RT-Notification is more effective at enforcing end-to-end priorities than CosNotification.

Jitter analysis. Figure 10 shows the standard deviation of the latency measured for the high-priority path as load is increased. Note that the jitter obtained with CosNotification increases linearly with the load. In contrast, the jitter for RT-Notification remains low and fairly constant. These results show that the throughput of the high-priority path is maintained while there is available capacity in the system. The latency of high-priority invocations is thus the lowest among the competing paths.

4.3. Number of Paths vs. Throughput

Overview. This benchmark measures the effects of increasing the number of low-priority paths, similar to the one described in Section 4.2. In this case, however, the load is held constant at 30 units and the number of low-priority paths is increased. All low-priority tasks are at the same priority. We ran the experiment with 1, 5, 10, and 20 low-priority paths.

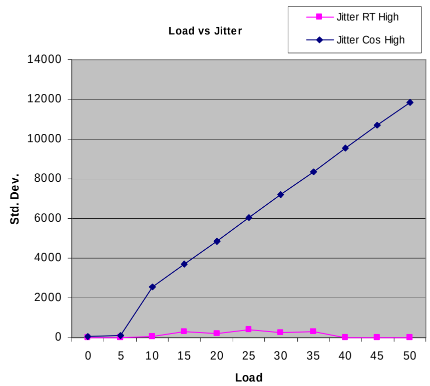


Figure 10: Load vs Jitter

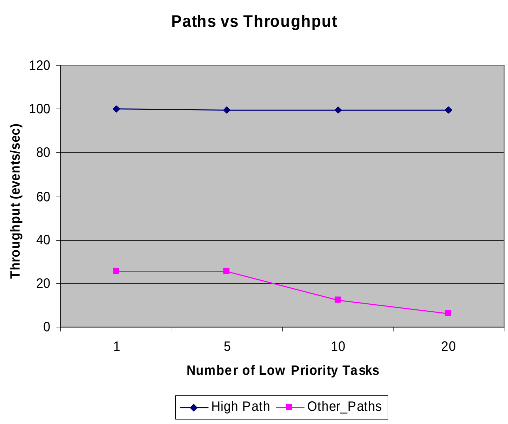
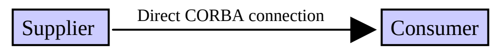


Figure 11: Increasing the Number of Paths

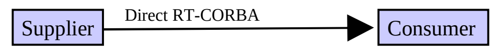
Results. Figure 11 shows that the throughput of the high priority path was maintained consistently at 100 events/sec, despite the increase in the number of low-priority paths. This result shows that TAO's Real-time Notification Service protects the performance of higher priority paths, even as resource demands of lower-priority paths are increased.

4.4. Overhead of RT- Notification

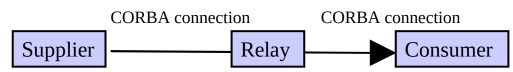
Overview. This experiment measured the maximum throughput obtained between a single supplier and consumer path for the following cases where the payload is a structured event that contains 64 bytes of payload representing the time at which the event was sent.



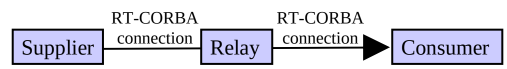
- 2-Hops: In this configuration a supplier sends an event directly to a consumer without an event channel.



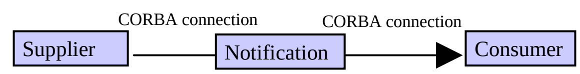
- 2-Hops-RT: This configuration modifies the 2-Hops test to use Real-time CORBA - the consumer is activated in a real-time POA with a single lane.



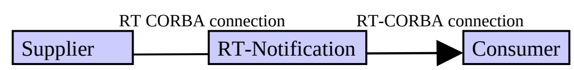
- 3-Hops: In this configuration, the supplier sends events to a *relay* consumer, which forwards the event to the final consumer. Hence, the relay consumer behaves as a minimal event channel.



- 3-Hops-RT: The 3-Hops configuration is modified to use Real-time CORBA. The relay consumer and final consumer are activated in a real-time POA with a single lane.



- CosNotify: This configuration is the standard CosNotification with 1 supplier and 1 consumer.



- RT-Notify: This configuration uses the RT-Notification Service.

Results. Figure 12 shows that the RT-Notification does not add significant overhead to the CosNotification Service. This overhead that is observed is due to the additional processing performed, per request, by the ORB due to the extra Real-time CORBA information in the IIOP messages.

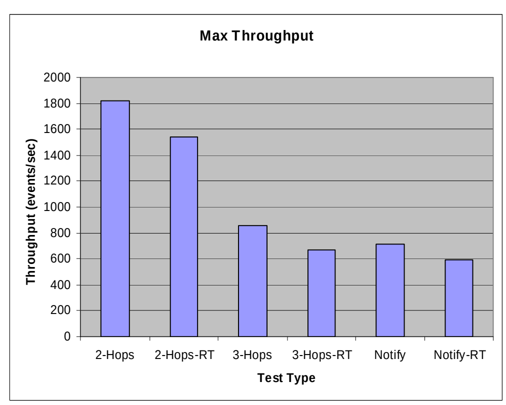


Figure 12: Maximum Throughput

5. Concluding Remarks

TAO's Real-time Notification Service provides end-to-end QoS support for anonymous event communication, improved timeliness and predictability in the transmission and delivery of events to event consumers via event channels, and integration with Real-time CORBA features (particularly in the areas of configuration of priorities and scheduling). The empirical results of Section 4 show that our prototype exhibits priority preservation, low jitter, and low overhead.

TAO's Real-time Notification Service currently provides real-time distributed event communication for statically scheduled applications and does not address dynamic scheduling issues. Our future work will therefore integrate the dynamic scheduling features of the Kokyu scheduling framework [15] that have already been integrated with TAO's Real-time Event Service [11] with TAO's Real-time Notification Service. In addition, we will enhance TAO's Real-time CORBA Notification Service so that it conforms to the OMG Real-time Notification Service specification when it is finalized.

References

[1] Object Mgmt. Group, *Notification Service Specification*, OMG Doc. telecom/99-07-01 ed., July 1999.

[2] Object Mgmt. Group, *Real-time CORBA Joint Revised Submission*, OMG Doc. orbos/99-02-12 ed., Mar 1999.

[3] Object Mgmt. Group, *Real-time CORBA 2.0: Dynamic Scheduling Joint Final Submission*, OMG Doc. orbos/2001-06-09, June 2001.

[4] Object Mgmt. Group, *Real-time Notification: Request For Proposals*, OMG Doc. orbos/00-06-10, June, 2000.

[5] Object Mgmt. Group, *Event Service Specification Version 1.1*, OMG Doc. formal/01-03-01 edition, 2001.

[6] Y. Aahlad, B. Martin, M. Marathe, and C. Lee, *Asynchronous Notification Among Distributed Objects*, Proc. 2nd conference on Object-Oriented Technologies and Systems, USENIX, Toronto, Canada, 1996.

[7] C. Ma and J. Bacon, *COBEA: A CORBA-Based Event Architecture*, Proc. 4th Conference on Object-Oriented Technologies and Systems, USENIX, Apr. 1998.

[8] P. Gore, D. Schmidt, C. O'Ryan, and R. Cytron, *Designing and Optimizing a Scalable CORBA Notification Service*, Proc. ACM SIGPLAN Workshop on Optimization of Middleware and Distributed Systems (OM 2001), Snowbird, Utah, June 18, 2001.

[9] T. Harrison, D. Levine, and D. Schmidt, *The Design and Performance of a Real-time CORBA Event Service*, Proc. OOPSLA '97, ACM, Atlanta, GA, 1997.

[10] C. O'Ryan, D. Schmidt, and J.R. Noseworthy, *Patterns and Performance of a CORBA Event Service for Large-scale Distributed Interactive Simulations*, Intl. Journal of Computer Systems Science and Engineering, vol. 17, num. 2, CRL Publishing, 2002.

[11] C. Gill, D. Schmidt, and R. Cytron, *Multi-Paradigm Scheduling for Distributed Real-time Embedded Computing*, IEEE Proceedings 91(1), Jan 2003.

[12] D. Schmidt, D. Levine, and S. Mungee, *The Design and Performance of Real-time Object Request Brokers*, Computer Communications, vol. 21, num. 4, pp. 294--324, Elsevier, 1998.

[13] D. 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, Lars Lundberg and Jan Bosch (Ed), Elsevier, 2002.

[14] White, Lepreau, Stoller, Ricci, Guruprasad, Newbold, Hibler, Barb, and Joglekar, *An Integrated Experimental Environment for Distributed Systems and Networks*, OSDI, December 2002.

[15] C. Gill, D. Levine, and D. Schmidt, "The Design and Performance of a Real-time CORBA Scheduling Service," *The International Journal of Time-Critical Computing Systems 20(2)*, Kluwer, March 2001.