A Scheduling Service for a Dynamic Real-Time CORBA System[*]

Lisa Cingiser DiPippo, Victor Fay Wolfe
Roman Ginis and Michael Squadrito
Department of Computer Science
University of Rhode Island
Kingston, RI USA 02881
lastname@cs.uri.edu

Thomas Wheeler
The MITRE Corporation
Bedford, Mass USA
twheeler@mitre.org
Russell Johnston
U.S. Navy NRaD
San Diego, CA USA
russ@nosc.mil

Abstract

Distributed real-time applications have presented the need to extend the Object Management Group's (OMG) Common Object Request Broker Architecture (CORBA) standard to support real-time. The OMG has formed a real-time special interest group (RT SIG) to specify requirements for extending CORBA for real-time. One of these requirements involves providing global scheduling of all executions to support end-to-end timing constraints in the real-time CORBA system. This paper describes the design and implementation of a real-time scheduling service for a Dynamic Real-Time CORBA system.

1 Introduction

Distributed object computing is becoming a widely accepted programming paradigm for applications that require seamless interoperability among heterogeneous clients and servers. The Object Management Group (OMG), an organization of over 700 distributed software vendors and users, has developed the Common Object Request Broker Architecture (CORBA) as a standard software specification for such distributed environments. The CORBA specification includes an *Object Request Broker* (ORB), which is the middleware that enables the seamless interaction between distributed client objects and server objects; *Object Services*, which facilitate standard client/server interaction with capabilities such as naming, event-based synchronization, and concurrency control; and the *Interface Definition Language(IDL)*, that defines the object interfaces within the CORBA environment.

Many distributed real-time applications, such as command and control, automated manufacturing, telecommunications, and simulation, are embracing the object-oriented paradigm and have a mandate to use an open systems design. The designers of many of these applications are considering CORBA for their architecture, but are finding it is currently inadequate to support real-time requirements. CORBA contains neither the services, nor the interface facilities to express and enforce end-to-end timing constraints on distributed client/server interactions.

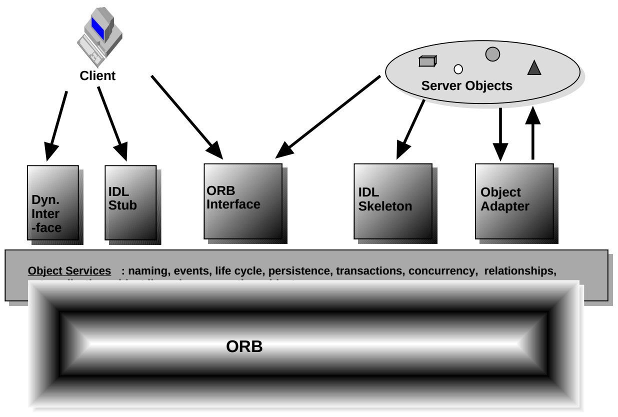
The OMG has formed a Special Interest Group (RT SIG) with the goal of extending the CORBA standard with real-time extentions. One of the requirements that has been established by the RT SIG involves providing global real-time scheduling to support the enforcement of end-to-end timing constraints on client-server interactions.

We have developed a dynamic real-time CORBA system prototype that meets many of the requirements specified by the RT SIG [3], including a scheduling service with that enforces a dynamic global priority ordering across the CORBA system. This paper describes the design and prototype implementation of our scheduling service. Section 2 presents background on CORBA, real-time CORBA and real-time scheduling. It also describes related work in real-time CORBA scheduling. Section 3 briefly describes our real-time CORBA system design, and goes on to describe in detail our scheduling service. Section 4 concludes by summarizing what we have done so far and by indicating areas for future work.

2 Background

This section provides background on the CORBA architecture and on the OMG RT SIG's scheduling requirements. The section also describes other related work in real-time CORBA scheduling.

[*]This work is partially supported by the U.S. Office of Naval Research grant N000149610401.


Client
Server Objects
Dyn. Inter -face
IDL Stub
ORB Interface
IDL Skeleton
Object Adapter
Object Services : naming, events, life cycle, persistence, transactions, concurrency, relationships,
ORB
Figure 1: CORBA System Components

2.1 CORBA

CORBA is designed to allow a programmer to construct object-oriented programs without regard to traditional object boundaries such as address spaces or location of the object in a distributed system. The CORBA specification includes: an *Interface Definition Language(IDL)*, that defines the object interfaces within the CORBA environment; an *Object Request Broker* (ORB), which is the middleware that enables the seamless interaction between distributed client objects and server objects; and *Object Services*, which facilitate standard client/server interaction with capabilities such as naming, event-based synchronization, and concurrency control.

Figure 1 illustrates the parts of a CORBA system. The client stubs and the server skeletons are produced by the IDL compiler. There is a stub and skeleton for each method on a server's interface. The purpose of these code segments is to hide the details of communication between the client and the server. Using the stubs, the skeletons, the ORB, and a component called the Basic Object Adapter, the CORBA system handles all details of the distributed method invocation so that the distribution is essentially transparent to the client and server application developers.

The CORBA standard contains specifications for Object Services that facilitate client/server interaction. These services include a Naming Service for binding a name to an object; an Event Service for notification of named events; and a Concurrency Control Service for locking of resources to maintain consistency. A more complete list of the Object Services can be found in [7]. In a real-time environment, a new service (scheduling service) is necessary to provide for the enforcement of client timing constraints across the real-time CORBA system.

2.2 Real-Time Scheduling

Modes of Real-Time. The way in which scheduling is performed in a real-time system depends upon the urgency of the timing constraints in the system. A *hard real-time* system involves timing constraints that must be met in order to avoid catastrophic results. In such a system, all timing constraints must be guaranteed to be met. Scheduling in a hard real-time system involves analyzing the timing constraints of all tasks a priori to determine if a feasible schedule of the tasks exists. If so, priorities are assigned to all tasks before execution of the system. This type of real-time system is known as a *static* real-time system because all executions that will occur in the system are known a priori and they don't change. *Soft real-time* systems involve timing constraints that are not as crucial as those in hard real-time systems. In a soft real-time system, a task whose timing constraint has been violated is usually completed anyway because it may still provide some value to the system. Soft real-time systems do not require guarantees about meeting timing constraints. Rather, they typically take a "best-effort" approach towards meeting timing constraints. Because soft real-time systems are more flexible with respect to meeting timing constraints, they tend to be more dynamic in nature. That is, it is not necessary to have full knowledge of all executions that will occur. Scheduling decisions are made dynamically as new tasks enter the system, and as system conditions change.

Real-Time Scheduling Algorithms. Most priority-based scheduling algorithms for real-time systems are priority assignment algorithms. That is, the algorithm takes the given timing constraints and produces a priority for every task. When executing, the highest priority task on a particular node always executes until completion, or until a higher priority task enters the system. In the latter case the higher priority task preempts.

Real-time scheduling algorithms can be divided into two types: fixed-priority and dynamic-priority. In a fixed-priority algorithm, once a task is assigned a priority, it keeps the same priority throughout its execution. An example of a fixed-priority scheduling algorithm is the *rate-monotonic* (RM) [5] which works for periodic tasks. RM assigns highest priority to the task with the shortest period. RM is usually used in a static real-time system with hard timing constraints because it can be fully analyzed for schedulability given a set of tasks with timing constraints. A dynamic-priority scheduling algorithm is one in which the priority of a task may change based on changing system conditions. *Earliest deadline first* (EDF) [5] is an example of a dynamic-priority scheduling algorithm. EDF assigns the highest priority to the task with the closest deadline. Clearly, as time progresses, the closeness of a task's deadline changes, and so priority must be adjusted to reflect this change. For instance, if a task enters the system with a long deadline, it will be assigned a low priority. However, as time progresses, and the task's deadline becomes closer, it may become necessary to increase the priority of the task in order to ensure that it will complete by its deadline.

2.3 Real-Time CORBA Scheduling Requirements

The OMG's RT SIG has produced a whitepaper [6] detailing the requirements that must be met in extending the CORBA standard for real-time. A full description of these requirements and how they might manifest themselves in a system can be found in [10]. The specific requirements for scheduling in real-time CORBA involve the need for a *global priority* that has meaning across the entire distributed system. A scheduling service in a real-time CORBA system must provide this global priority, based on the timing constraints expressed by the client's method invocations, and on the server's own timing requirements. The service must also ensure that the global priority can be mapped to the scheduling environments of all local operating systems involved in the execution. Finally, if the real-time CORBA system is a dynamic system, the scheduling service must ensure that the global priority correctly reflects the requirements of the associated execution for the duration of the execution. That is, it may be necessary to modify the value of the global priority based on changes that occur in the system.

2.4 Related Work

Other research has been done in the area of scheduling for a real-time CORBA system. MITRE has designed a system that provides a static distributed scheduling service supporting rate-monotonic and deadline-monotonic techniques [4, 1]. At the University of Washington in St. Louis, The ACE ORB (TAO) has been developed [9]. TAO is a static real-time CORBA system. The scheduling service uses fixed-priority scheduling, such as rate-monotonic scheduling. It performs off-line feasibility analysis of IDL operations, and it assigns priorities to threads based on the timing constraints expressed on the IDL operations. Static scheduling across the system, such as those provided by the MITRE and TAO scheduling services, are important steps toward supporting hard real-time applications. Our scheduling service is designed for soft real-time applications and is implemented in a dynamic real-time CORBA system, where clients and servers can be added or removed, timing constraints may change, and priorities are not fixed.

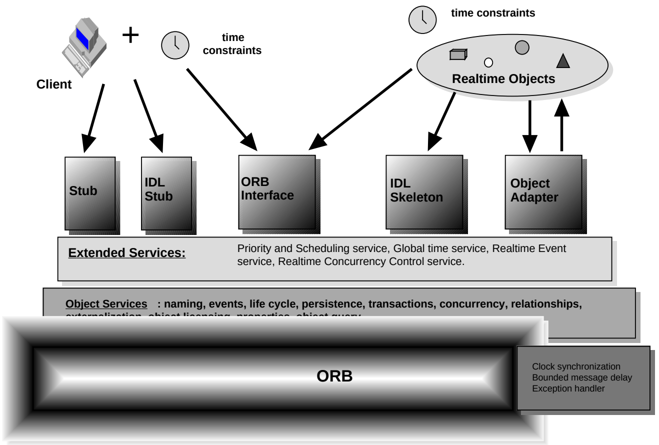

Client
+
time constraints
time constraints
Realtime Objects
Stub
IDL Stub
ORB Interface
IDL Skeleton
Object Adapter
Extended Services:
Priority and Scheduling service, Global time service, Realtime Event service, Realtime Concurrency Control service.
Object Services : naming, events, life cycle, persistence, transactions, concurrency, relationships,
ORB
Clock synchronization
Bounded message delay
Exception handler

Figure 2: Dynamic Real-Time CORBA System

3 Global Scheduling in a Dynamic Real-Time CORBA System

This section gives a brief overview of our Dynamic Real-Time CORBA system. It then describes in detail the scheduling service designed for the system. For a full description of our Dynamic Real-Time CORBA system, see [3, 10].

3.1 Dynamic Real-Time CORBA System

A depiction of our Dynamic Real-Time CORBA system components is shown in Figure 2. The system allows a client to add timing constraints to a server method call through a construct called a *Timed Distributed Method Invocation* (TDMI). In the TDMI, timing information such as importance, and deadline, is packed into a structure called the *RT_Environment* which is examined by the ORB and the object services to enforce timing constraints. As Figure 2 shows, along with the Scheduling Service, we have provided several other extended object services to enforce the timing constraints expressed by the client's TDMI. The *Global Time Service* allows clients and servers to get the current global time. It uses calls to the local operating system that provide a time that is synchronized with the time from all other nodes. The *Real-Time Event Service* prioritizes delivery of events and delivers the time that the event occurred. The *Real-Time Concurrency Control Service* implements priority inheritance [8] so that TDMIs with low priorities do not block TDMIs with higher priority on a server for extended periods of time. We also added clock synchronization, bounded message delay, and a

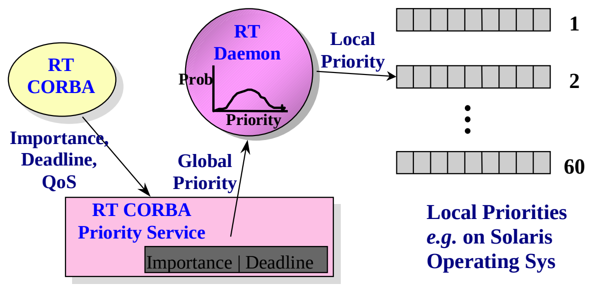

RT CORBA
Importance, Deadline, QoS
RT Daemon
Prob
Priority
Global Priority
Local Priority
RT CORBA Priority Service
Importance | Deadline
1
2
60
Local Priorities e.g. on Solaris Operating Sys

Figure 3: Real-Time CORBA Scheduling Service

real-time exception handler to the CORBA system to enforce real-time constraints.

The components of our Dynamic Real-Time CORBA system are implemented as a *Real-Time Daemon* process (RT Daemon) that executes on each real-time POSIX operating system in the system, and as a real-time library that provides type definitions, IDL definitions, and code that is used to link in with client and server code. The RT Daemon coordinates dynamic aspects of the system including changing global priorities, time synchronization, and supporting real-time events. The library code performs tasks such as initial priority assignment, handling of real-time information that is associated with all execution in the system, and handling of real-time exceptions.

3.2 Dynamic Scheduling Service

The Dynamic Scheduling Service performs three major tasks: assignment of a global priority, mapping of the global priority to local operating systems, and global priority adjustment due to changes in the real-time CORBA system. Figure 3 shows how the Dynamic Scheduling Service works. A client expresses timing constraints such as importance and deadline. The Priority Service (a part of the Dynamic Scheduling Service) assigns a global priority to the client call. And the RT Daemon (also part of the Dynamic Scheduling Service) maps the priority to the priorities available on the local operating system.

3.2.1 Global Priority Assignment

Dynamic real-time scheduling is done by establishing a global priority assignment for all execution in the Dynamic Real-Time CORBA system. Each client communicates its scheduling parameters to a *Global Priority Service*, and in turn receives a global priority for its execution. These priorities are dynamic and may change over the lifetime of the execution. Figure 3 shows how the Global Scheduling Service works.

We call an execution's priority at an instant in time its *transient priority*. A transient priority is an integer that is derived by the Global Priority Service based on the information in the *RT_Environment* for the execution. The Global Priority Service ensures that the transient priority is meaningful relative to all other transient priorities in our Dynamic Real-Time CORBA system. That is, much like a single real-time operating system assigning priorities within its local domain, the Global Priority Service assigns priorities that are meaningful across the real-time CORBA domain.

The Global Priority Service uses a uniform function for all clients and servers in the system to compute transient priority using the attributes in the *RT_Environment* that is associated with the execution. Our prototype uses a global *earliest-deadline-first within importance* priority assignment scheme. That is, the prototype's transient priority function orders priorities based on the *importance* attribute first, and then based on the *deadline* attribute. A transient priority is a seven digit value, where the millions digit represents importance, and the lower order digits represent a time difference (multiplied by 100,000) between the maximum allowable deadline and the deadline specified in the *RT_Environment* for the execution. For instance, if the maximum deadline is 10 seconds, then execution with importance level 2 and a deadline of 3 seconds has a transient priority of 2,700,000. Changing the calculation of transient priorities based on other scheduling policies, such as global rate-monotonic priority assignment, is facilitated by the function's central implementation in the Global Priority Service.

The implementation of the Global Priority Service in our prototype is accomplished through code from the RT Library. The library code calculates the initial transient priority.

3.2.2 Priority Mapping

The Dynamic Scheduling Service maps the transient priority to the priorities available on the local real-time operating system, through the RT Daemon on each node. The function that performs the mapping must be written for each operating system individually because of the variability in ranges of real-time priorities present on different systems (e.g., Solaris has 60 local priorities, and LynxOS has 256). In our prototype, which uses RT Solaris operating systems, the RT Daemon must map the (wide) range of transient priorities into the 60 local priorities. The mapping is done by using a statistical model of the likely deadlines and calculating transient priorities such that TDMIs are probabalistically evenly distributed among the local priorities. For example, if there were 60 executions to be scheduled on a Solaris node, the mapping would reduce the probability that two executions would be at the same priority. Unfortunately mapping of a large range of transient priority values into a smaller range of priorities can cause more than one transient priority to be mapped to a single local priority value, which could cause some execution to be out of deadline order.

3.2.3 Dynamic Change of Transient Priority

An execution may have different transient priorities at different times during its lifetime. The RT Daemon is responsible for implementing this part of the Dynamic Scheduling Service. There are four reasons that an execution's priority may be changed:

- Intermediate deadlines that are different from a client's overall deadline.
- Loss of time due to network delay and clock skew when a call is made on a server on another node.
- Aging due to the use of a dynamic-priority scheduling algorithm.
- Priority inheritance in the Real-Time Concurrency Control Service.

We describe each of these situations next.

Intermediate Deadlines. A real-time CORBA client could have an initial transient priority based solely on its importance with no deadline. It then might enter phases of its execution that must be done under deadlines. Thus, each time a new TDMI is started, the RT Daemon recalculates the client's transient priority. Similarly, when a TDMI is complete, the RT Daemon recalculates the client's transient priority using the deadline (if any) that was in effect before the TDMI was initiated.

Inter-node Communication. Another re-calculation of transient priority is done when a client makes a TDMI to a server, to account for time lost due to network communication. Assume that the client's deadline constraint is d_{client}. This means that the return message from the server with results for the client must be received by the client by d_{client} as measured on the client's clock. In our system, we assume synchronized clocks with maximum skew ϵ, and assume maximum network message delay δ. To calculate the transient priority at which a server should execute on behalf of the client, the server uses the deadline $d_{server} = d_{client} - \delta - \epsilon$ to pessimistically allow for δ message delivery time back to the client and an ϵ clock skew between its clock and the client's clock. Since this deadline is tighter than the client's deadline on whose behalf the server is executing, the TDMI will usually have a higher transient priority when executing in the server than it will while executing in the client.

Transient Priority Aging. Another change in an execution's transient priority is performed by the RT Daemons in our Dynamic Real-Time CORBA system enforcing *aging* of transient priorities. Aging is the process of increasing priority as time goes on, which is necessary in dynamic earliest-deadline-first scheduling. Each RT Daemon keeps track of the transient priorities on its node. A RT Daemon increases an execution's transient priority if, due to the passage of time, the execution's transient priority is too low compared to a newly-arrived execution on the node which the RT Daemon controls. Note that in our prototype the aging facility can be "turned off" for real-time scheduling policies that do not require aging, such as a static rate-monotonic-based policy.

As an example of priority aging, suppose that three TDMIs, T_1, T_2, and T_3 enter the system at time 0 with deadlines 17, 19 and 21 respectively. Considering that these are all relatively long deadlines, the Global Priority Service using EDF assigns them relatively low priorities: 20, 25 and 30 respectively (lower number indicates higher priority). Then at time 15 (15 seconds later), another TDMI T_4 enters the system with a deadline of 20. This deadline is close (5 seconds to complete the TDMI), and so TDMI T_4 must be given a high priority. However, the deadlines of the other TDMIs have also become tighter. The Dynamic Scheduling Service, through the RT Daemon, recalculates the priorities of all TDMIs with shorter deadlines than T_4. Thus, we might have the following new priority assignments: T_1: 3, T_2: 5, T_4: 7, T_3: 30. Notice that the priority of TDMI T_3 does not change, but that the relative positions of all priorities are correct.

Transient Priority Inheritance. Another source of possible of transient priority change is due to priority inheritance in the Real-Time CORBA Concurrency Control Service. When a TDMI requests a lock on a resource from the Real-Time Concurrency Control Service, the TDMI's execution priority is compared to those of all TDMIs holding conflicting locks on that resource. The Dynamic Scheduling Service (through the RT Daemon) raises the priorities of conflicting TDMIs with lower priorities to the requesting TDMI's priority, and the requesting TDMI is suspended. Whenever a lock is released, the Dynamic Scheduling Service resets the priority of the releasing TDMI to that of the highest priority TDMI it still blocks. If it no longer blocks any higher priority TDMIs, then the releasing TDMI is reset to its original priority. Finally, the highest priority blocked TDMI that can now run is allowed to obtain its lock and continue execution.

3.3 Performance Test Results

We have performed tests on the prototype implementation of our Dynamic Real-Time CORBA system in order to determine how well it enforces expressed timing constraints. We measured the number of missed client deadlines in our prototype and in a non-real-time CORBA environment (no scheduling service). In general, the test results indicate that our prototype, with an explicit scheduling service implemented, misses fewer deadlines than a non-real-time CORBA system with no scheduling service. A full description of the tests and their results can be found in [3].

4 Conclusion

This paper has presented a Scheduling Service for a Dynamic Real-Time CORBA System. The OMG's RT SIG has specified the need for a global priority for every execution in the system. The Global Scheduling Service provides this global priority, as well as the maintenance of the global priority for the duration of the execution. The Global Priority Service provides the initial transient priority for the execution to ensure that all CORBA requests are scheduled at all points in the distributed system according to the same real-time policy. The Global Scheduling Service then maps the transient priority to the possibly limited priority set of the server's local operating system. The Global Scheduling Service is also responsible for recalculating the transient priority for an execution to reflect changes in system conditions. These changes in timing conditions can include new TDMI's, a call to a server on another node, aging due to a dynamic-priority scheduling algorithm, or priority inheritance due to concurrency control.

The design of our Global Scheduling Service facilitates changes in the choice of scheduling algorithm. Our prototype uses earliest-deadline-first, but other algorithms, such as rate-monotonic or deadline-monotonic can easily be used instead. While the current prototype uses only importance and deadline, we are investigating which other Quality of Service (QoS) parameters can be factored in as scheduling parameters and how to use them to generate transient priorities.

We are using our experience in developing a dynamic scheduling service to design and build a scheduling service for a static real-time CORBA system. The scheduling service will involve using the PERTS [2] real-time analysis tool off-line to determine schedulability of a real-time CORBA system of clients and servers, and to assign priorities a priori to each execution in the system.

References

[1] E. Bensley, et. al. Object-oriented approach for designing evolvable real-time command and control systems. In *WORDS '96*, February 1996.

[2] Jane W. S. Liu, et. al. PERTS: A prototyping environment for real-time systems. Technical Report UIUCDCS-R-93-1802, The University of Illinois, Urbana, May 1993. Commercial version information available at www.tripac.com.

[3] Victor Fay Wolfe, et. al. Expressing and enforcing timing constraints in a dynamic real-time corba system. Technical Report TR-98-260, The University of Rhode Island, Jan 1998. submitted to The Real-Time Systems Journal.

[4] P. Krupp, A. Schafer, B. Thuraisingham, and V. F. Wolfe. On real-time extensions to the common object request broker architecure. In *Proceedings of the Object Oriented Programming, Systems, Languages, and Applications (OOPSLA) '94 Workshop on Experiences with the Common Object Request Broker Architecture (CORBA)*, Sept. 1994.

[5] C. L. Liu and J. W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the ACM*, 20(1):46–61, 1973.

[6] The Realtime Platform Special Interest Group of the OMG. CORBA/RT white paper. ftp site: ftp://ftp.osaf.org/whitepaper/Tempa4.doc, Dec 1996.

[7] OMG. *CORBAservices: Common Object Services Specification.* OMG, Inc., 1996.

[8] R. Rajkumar. *Synchronization in Real-Time Systems: A Priority Inheritance Approach.* Kluwer Academic Publishers, Boston, MA, 1991.

[9] D. Schmidt, D. Levine, and S. Mungie. The design of the tao real-time object request broker. *Computer Communications Journal.* to appear.

[10] V. F. Wolfe, L. C. DiPippo, R. Ginis, M. Squadrito, S. Wohlever, I. Zykh, and R. Johnston. Real-time CORBA. In *Proceedings of the Third IEEE Real-Time Technology and Applications Symposium*, June 1997.