Real-Time CORBA[*]

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

Russell Johnston
NCCOSC
RDT&E DIV (NRaD)
53140 Systems Street
San Diego, CA 92152
russ@nosc.mil

Abstract

This paper describes the requirements for real-time extensions to the CORBA standard, which are being developed by the Object Management Group's Special Interest Group on Real-Time CORBA. The paper also surveys efforts that are developing Real-Time CORBA systems. It provides a more detailed description of the dynamic Real-Time CORBA system being developed at the US Navy's NRaD facilities and at the University of Rhode Island.

1 Introduction

The Object Management Group (OMG), an organization of over 600 distributed software vendors and users, has developed the Common Object Request Broker Architecture (CORBA) as a standard software specification for distributed object environments. In 1995 a Special Interest Group (SIG) was formed within the OMG with the goal of extending the CORBA standard with support for real-time applications. This real-time SIG (RT SIG), which itself is a consortium of academics, vendors, end-users, and government researchers, is developing requirements for extending/modifying CORBA to support real-time.

In this paper we describe Real-Time CORBA (RT CORBA). Section 2 is a brief description of the existing CORBA standard. Section 3 summarizes some of the requirements for RT CORBA that are outlined in the current draft of OMG's RT SIG whitepaper [RTSIG96]. Section 4 surveys current work in RT CORBA development and prototyping. Section 5 provides a more detailed look at the RT CORBA research and development at the Navy's NRaD facility and at the University of Rhode Island. Section 6 summarizes the status of RT CORBA.

[*]The authors' effort and the work presented in Section 5 are supported by the U.S. Office of Naval Research grant N000149610401.

2 CORBA Background

The OMG has been meeting approximately every six weeks since 1989. It has sub-groups that develop aspects of the CORBA standard including its relation to the Internet, business, manufacturing, telecommunications, operating systems, and other aspects. The CORBA specification process is evolutionary. Aspects of the standard are proposed, bid, debated and adopted piecemeal according to a roadmap established by the OMG. CORBA version 1.1 was released in 1992, version 1.2 in 1993, and version 2.0 in 1995. The V1.2 standard deals primarily with the basic framework for applications to access objects in a distributed environment. This framework includes an object interface specification and the enabling of remote method calls from a client to a server object. Issues such as naming, events, relationships, transactions, and concurrency control are addressed in Version 2.0 [OMG96]. Services such as time synchronization and security are expected to be addressed in later revisions. To date, the OMG has been remarkably successful in agreeing upon increments to the standard and vendors have quickly made products available that meet the evolving standard.

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. That is, a client program should be able to invoke a method on a server object whether the object is in the client's address space or located on a remote node 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 *Ob-*

ject Services, which facilitate standard client/server interaction with capabilities such as naming, event-based synchronization, and concurrency control.

CORBA IDL. CORBA IDL is a declarative language that describes the interfaces to server object implementations, including the signatures of all server object methods callable by clients. The IDL grammar includes a subset of ANSI C++ with additional constructs to support the method invocation mechanism.

Most common intrinsic C++ types are supported in CORBA IDL. The ORB handles differences in type representations among architectures (e.g. Big Endian, Little Endian) except for the native CORBA `octet` type, which is never translated. CORBA's IDL also specifies C++-like exception raising and handling. IDL does not provide syntax for implementing methods: an IDL binding to the C language has been specified for that purpose. Other language bindings include C++ and Smalltalk.

As an example, consider an object that acts as a shared table for sensor data (represented as long integer values) for clients in a distributed system. A simple CORBA IDL for a sensor_table object is:

```
interface sensor_table
 { readonly attribute short max_length;
  short put(in short index,in long data);
  long get(in short index);
 }
```

The IDL keyword `interface` indicates a CORBA object (similar to a C++ class declaration). A `readonly attribute` is a data value in the object that a client may read (the IDL compiler generates a remote method for reading each attribute). The IDL example also specifies two methods: *put*, which stores a sensor value at a index into the table; and *get* which returns a sensor value given an index.

Client code in C to access a sensor_table object in a CORBA environment might look like:

```
long retval;
sensor_table *p;
p = bind("my_sensor_table");
retval = p->get(500);
```

Here, the client declares a pointer, *p*, to a *sensor_table* object called *my_sensor_table*. The client then makes a call to an ORB service to locate and bind the pointer to a reference to a remote server containing the *sensor_table* object. To retrieve a value from the *sensor_table* at index 500, the client issues the method invocation: `p->get(500)`. This method invocation

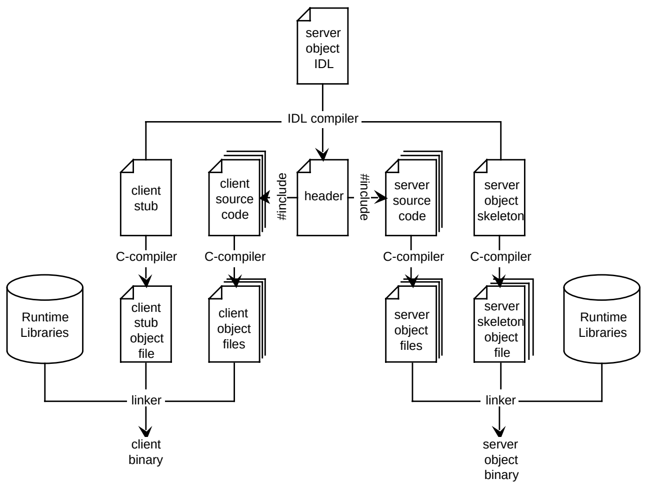

Figure 1: CORBA IDL Compilation Process

assumes that a *sensor_table* server was previously implemented and registered with the CORBA ORB.

Implementing Clients and Servers. The process of implementing a client and server object is shown in Figure 1. The IDL specification is processed by an IDL compiler, which generates a header file for the CORBA object, stub code for linking into the client, and skeleton code for the server object. The client stub contains code that hides details of interaction with the server from the client code. Client stubs stand in for normal method calls by transparently directing normal-appearing C++ method requests into the ORB. Server skeleton code is used by the ORB in forwarding method invocation requests to the server, and in returning results to the client.

The ORB. An ORB provides the services that:

- locate a server object implementation for servicing a client's request;
- establish a connection to the server;
- communicate the data making up the request;
- activate and deactivate objects and their implementations;
- generate and interpret object references.

A client and its stubs, a server and its skeleton, and the interaction through the ORB are shown in Figure 2. Some parts of the ORB are not shown, such as the Dynamic Invocation Interface since it is not fully developed in the CORBA standard nor is it expected to

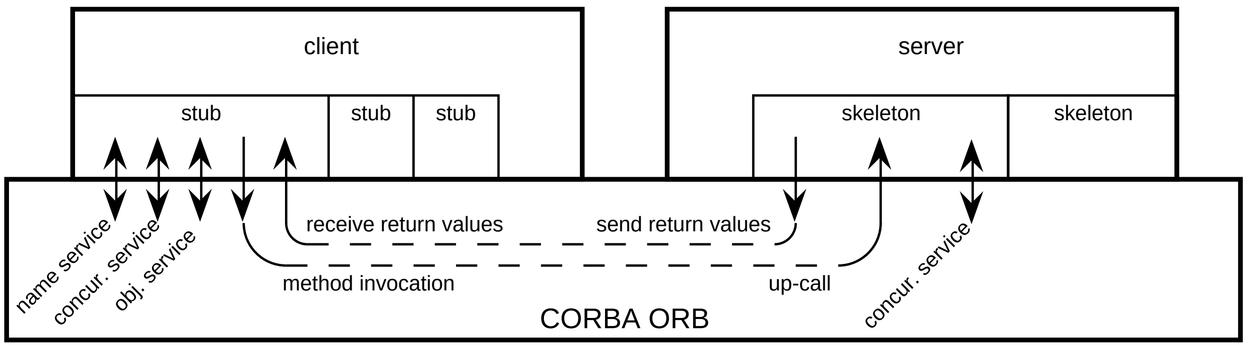

Figure 2: The ORB

be used in real-time applications. The stubs and skeletons are produced by the IDL compiler as described before. Figure 2 also shows calls to some object services.

Object Services. CORBA 2.0 contains the specifications for the following Object Services:

- *Naming.* This service provides the ability to bind a name to an object relative to a naming context. It guarantees unique names for objects.
- *Event.* This service provides basic capabilities for notification of named events. Suppliers can generate events without knowing the IDs of consumers. Consumers can use events without knowing the IDs of suppliers.
- *Life Cycle.* This service allows creatation and destruction of objects in the CORBA system.
- *Persistence.* This service allows making objects persistent on some storage medium.
- *Transactions.* This service allows construction of transactions, which are atomic collections of client calls to server objects.
- *Concurrency Control.* This service allows objects to be locked by clients. The locking scheme is a version of database read/write locking.
- *Externalization.* This service allows objects to be passed in a CORBA environment and among environments.
- *Relationship.* This service allows the expression of semantic relationships among objects.

The CORBA roadmap for future services calls for time synchronization, security, system startup, redundancy, recovery, query processing, and other services to be provided.

3 Real-Time CORBA

RT CORBA is being defined by the OMG RT SIG. The essence of its definition is:

RT CORBA deals with the expression and enforcement of real-time constraints on end-to-end execution in a CORBA system.

Consider a real-time scenario where a client needs to perform a *get* method from the sensor table server within timing constraints. This interaction means that the client must have some way of expressing timing constraints on its request, and that the CORBA system must provide an ORB and Object Services that support enforcement of the expressed timing constraints. It also means that the underlying operating systems on the client and server nodes, along with the network that they use to communicate, can support enforcement of real-time constraints.

That is, there are two main categories of RT CORBA requirements: requirements on the operating environment (operating systems and networks); and requirements on the CORBA run-time system. In this section, we summarize some of these requirements. The full set of requirements can be found in the RT SIG whitepaper [RTSIG96].

3.1 Operating Environment

Some requirements of the RT CORBA operating environment are:

- *Synchronized Clocks.* All clocks on nodes in an ORB should be synchronized to within a bounded skew of each other.
- *Bounded Message Delay.* The underlying communication mechanism should ensure a worst-case message delay from one CORBA system task to another.

- *Priority-based Operating Environment Scheduling.* All components used in the underlying CORBA environment should support priority-based scheduling and queueing where a higher priority task is scheduled before a lower-priority task. This scheduling should be preemptive where possible (such as CPU scheduling).

- *Operating Environment Priority Inheritance.* All components used in the underlying CORBA environment that synchronize tasks by blocking one task for another should implement priority-inheritance.

3.2 CORBA Run-Time System.

Some requirements of the ORB and Object Services components of the CORBA standard are:

- *Time Type.* The CORBA standard should specify a standard type for absolute time and relative time.

- *Transmittal of Real-Time Method Invocation Information.* The standard should allow the following information to be established by the client and attached to its method invocation request so that the information is available to the ORB, ORB Services, skeletons, and server implementations: Deadline; Importance; Earliest Start Time; Latest Start Time; Period; what quality of service (QoS) is required; and should the ORB offer a guarantee. This information is likely needed to enforce real-time constraints through techniques like assigning priorities and setting operating system alarms.

- *Global Priority.* The ORB should establish priorities for all execution. These priorities should be "global" across the ORB. That is, the priorities of any tasks that compete for any resource in the CORBA environment should be set relative to each other. This requires that the competing tasks have priorities that "make sense" relative to each other. Several priority schemes, such as rate-monotonic priority assignment, earliest-deadline-first priority assignment, or a variation that weights tasks based on importance, are possible. The standard will not dictate how priorities are set. Instead, the standard will specify only that the information needed to set priorities is available, and that the priorities will be enforced.

- *Priority Queueing of All CORBA Services.* All CORBA-level software should use priority based queuing. Enforcing priorities at all points in the end-to-end path, including CORBA service requests, is desirable for soft real-time and necessary for hard real-time. For instance, queues of requests for CORBA 2.0 services such as Naming or Lifecycle should be priority queues.

- *Real-Time Events.* The CORBA environment should provide the ability for clients and servers to determine the absolute time value of "events". These events may include the current time (provided by a Global Time Service), or named events provided by the CORBA 2.0 Event Service. The specification of timing constraints requires the determination of the time for the constraint. These times are absolute times. Most application specifications denote these times as relative offsets from events. For instance "within 10 seconds" typically means "within 10 seconds of the current time" and thus needs the current time. "Within 10 seconds of completion of Task A" needs the time that a named event for "completion of Task A" occurred. Furthermore, events should be delivered in an order reflecting either the priority of the event or the priority of the event consumer, or both.

- *Priority Inheritance.* All RT CORBA-level software that queues one task while another is executing should use priority inheritance. This requirement includes the locking done by the CORBA 2.0 Concurrency Control Service, but also includes simple queuing such as waiting for the Naming Service.

- *Real-Time Exceptions.* The CORBA exception mechanism should be extended to raise the exceptions including missed deadline and violated guarantee. These exceptions should be handled within the context of the CORBA exception handling mechanism.

- *Documented Execution Times.* The CORBA standard should specify that vendors must publish worst case bounds for all execution in their product.

- *ORB Guarantees.* If the client specifies that it wishes a guarantee for a certain QOS with timing constraints, the ORB should be able to either guarantee it, or raise an exception.

4 RT CORBA Development

There have been several RT CORBA projects initiated over the past few years. This section describes some of the original approaches to RT CORBA and then surveys current approaches as reported in the OMG's Request For Information (RFI) on RT CORBA [RTSIG97]. The next section then provides detail on one approach: the dynamically scheduled RT CORBA implementation from NRaD/URI.

CORBA On RT Operating Systems. One original approach to RT CORBA was to install a non-real-time ORB on real-time operating systems. This was the approach initially taken by Iona Technologies that released a "Real-Time ORB" which is essentially their non-real-time ORB ported to the Lynx and VXWorks real-time operating systems. However, these ported ORBs do not take advantage of most of the operating system's real-time features. Furthermore, although implementation on a real-time operating system may be necessary for RT CORBA, it is not sufficient to enforce end-to-end timing constraints in a distributed system.

Fast CORBA. Several projects including ORBs originally created by Lockheed Martin, and by vendors in DARPA's Distributed Interactive Simulation (DIS++) High Level Architecture sought to realize "real-time ORBs" that are stripped-down, faster, versions of existing ORBs. They removed features like CORBA's Dynamic Invocation Interface and allowed special protocol, fixed point-to-point connections of clients to servers that by-passed most CORBA features. Such high performance might also be necessary in a RT CORBA system, but it may not be sufficient for predictable enforcement of end-to-end timing constraints.

MITRE's RT CORBA. One of the first projects to incorporate expression and enforcement of end-to-end timing constraints into a CORBA system was designed by Peter Krupp and Bhavani Thuraisingham's group at MITRE in Bedford, Ma. [TKSW94, EB96]. This work identified requirements for the use of real-time CORBA in command and control systems and prototyped the approach by porting the ILU ORB from Xerox to the Lynx real-time operating system. They then provided a distributed scheduling service supporting rate-monotonic and deadline-monotonic techniques. The resulting infrastructure combines a POSIX-compliant real-time operating system, a real-time ORB, and an ODMG-compliant real-time ODBMS [EB96]. MITRE has successfully transferred this technology to the US Airforce AWACS program, for incorporation into improvements scheduled for delivery in 1998 [RTSIG97]. One application of this approach is in information survivability (e.g., when a user may attempt to bring down a system by over-using its resources) [KTM97].

CORBA on the New Attack Submarine. The Naval UnderSea Warfare Center (NUWC) and Lockheed Martin are investigating RT CORBA techniques for the US Navy's New Attack Submarine's (NSSN) C3I system [GWP96]. The NSSN C3I system specifications mandate real-time requirements and a CORBA system to interconnect subsystems such as Sonar, Radar, Navigation, and Combat Control. One part of the NSSN C3I project at NUWC has investigated the impact of RT CORBA over an ATM network. In addition to identifying sources of latencies, this project also proposed a CORBA Latency Server to supply latency estimates that may be required in RT CORBA operation [Pal97].

Washington University's RT CORBA. Researchers at Washington University in St. Louis are developing a high-performance endsystem architecture for real-time CORBA called TAO (The ACE ORB) [RTSIG97]. Their objective is to identify the key architectural patterns and performance optimizations necessary to build high-performance, real-time ORBs. The focus of the scheduling work at Washington University is on hard real-time systems, requiring *a priori* guarantees of QoS requirements. The key components of TAO include a Gigabit I/O subsystem; a real-time inter-orb protocol; a method for specifying QoS requirements; a real-time scheduling service; a real-time object adapter with a real-time event service; and presentation layer components.

The TAO system requires that the underlying operating system and network provide resource-scheduling mechanisms to support real-time guarantees. For instance, the operating system must support scheduling mechanisms that allow the highest priority task to run to completion. Furthermore, real-time tasks should be given precedence at the network level to prevent them from being blocked by low priority applications.

The I/O subsystem optimizes conventional OS I/O subsystems to execute at Gigabit rates over high-speed ATM networks [HGSP96]. TAO's Real-time Inter-ORB Protocol (RIOP) is a mapping of the general inter-ORB protocol that allows applications to transfer their QoS parameters from clients to server objects. When a message is sent from a client to a server, the RIOP packages attributes, such as priority and execution period, in the message in order to specify QoS requirements. Along with the QoS specification included in its method invocations, TAO allows for QoS specification at the IDL level as well. Because TAO performs *a priori* scheduling analysis, applications must specify their QoS needs in order to guarantee resource availability. Components that have real-time needs convey their QoS requirements to the CPU through an IDL construct called a RT_Task. TAO's Real-time Scheduling Service performs off-line feasibility analysis of real-time tasks based on the task QoS specifications. The Scheduling Service also assigns priorities to threads during the analysis. The Object Adapter is responsible for demultiplexing, scheduling and dispatching client requests onto object implementations. In TAO, the presentation layer transforms typed operation parameters from high-level to low-level representations (and vice-versa) via client-side stubs and server-side skeletons. These stubs and skeletons are generated by a highly optimizing IDL compiler.

The TAO system also provides a real-time capability within the CORBA Event Service [RTSIG97]. These extensions introduce several components augmenting the Event Service to support event scheduling and minimize dispatch latency, based on *a priori* knowledge of participating consumer(s)/supplier(s) and periodic rate-based events. The interfaces of the RT Event Service include QoS parameters that allow consumers and suppliers to specify their execution requirements and characteristics. These parameters are used by the event dispatching mechanism to integrate with the system-wide real-time scheduling policies to determine dispatching ordering and preemption strategies. In a real-time system, some consumers can execute whenever an event arrives from any supplier. Other consumers can execute only when an event arrives from a specific supplier, or when multiple events have arrived from a particular set of suppliers. The RT Event Service provides filtering and correlation mechanisms that allow consumers to specify logical OR and AND event dependencies. When those dependencies are met, the RT Event Service dispatches all events that satisfy the consumers dependencies. The RT Event Service allows consumers to specify event dependency timeouts and propagates temporal events in coordination with system scheduling policies.

CHORUS/COOL Real-time CORBA. The CHORUS/COOL ORB is a flexible real-time ORB that is being developed by Chorus Systems. Their design enforces a strict separation between resource management policy and mechanism. Their philosophy also calls for providing applications full control over operating system-level resources. Given this philosophy, the goals of the CHORUS/COOL ORB include: a flexible binding architecture; producing minimum CORBA on a minimal ORB; and a real-time operating environment that provides access to fine grain resource management.

The CHORUS/COOL architecture extends the general CORBA binding model to allow for explicit binding, which is done by invoking appropriate operations on the Object Adapter. This extension allows an application programmer to define customized and dynamic object bindings. The "componentized" technology provided by CHORUS/COOL ORB allows components of the ORB core and associated services to be customized to the needs of the application. This feature allows programmers to use the minimal implementation of the ORB that is required by the application. When CHORUS/COOL ORB is integrated with Chorus Systems' CHORUS/ClassiX real-time operating system, the application programmer has access to fine grain resources. Such resources include concurrency control (mutexes, etc.), priority-based scheduling, and memory management.

The COOL ORB from Chorus Systems does not provide many real-time features itself, but rather relies on the CHORUS operating systems. A strength of the COOL ORB is that it imposes minimal overhead on top of the native operating systems.

Other RT CORBA Technology. Other RT CORBA technology and position papers on RT CORBA are collected on the OMG's Web server under the RFI responses for RT CORBA technology at:

`www.omg.org/library/schedule/Realtime_RFI.htm`

5 NRaD/URI RT CORBA System

Our team at the University of Rhode Island and the US Navy's NRaD facility have developed a prototype that implements many of the requirements of RT CORBA. It is designed to support expression and enforcement of dynamic end-to-end timing constraints within a CORBA system [TKSW94, WBTK95]. It is implemented on Sparc workstations running Solaris 2.5 (with POSIX threads) using Iona Technology's Orbix_2.0.1MT(multi-threaded) as the CORBA baseline. All of our prototype software assumes an operating environment that is compliant with the POSIX real-time operating system standard. This environment satisfies most of the RT CORBA white paper operating environment desired capabilities of Section 3.1.

Our NRaD/URI RT CORBA system consists of a new CORBA service for Global Priority, and modifications of existing CORBA services for Events, and

Concurrency Control. Our RT CORBA system also includes several new IDL types for expressing real-time parameters, along with library code that is added to stubs of clients and skeletons of servers. Together these components support the expression and enforcement of *timed distributed method invocations* (TDMIs) [WBTK95] where real-time constraints are expressed on clients' CORBA method calls and enforced by the CORBA system.

5.1 Timed Distributed Method Invocations

A TDMI uses a new CORBA *RT_Environment* structure and a new C++ *RT_Manager* class to convey real-time information. A *RT_Environment* structure contains attributes that include *importance*, *deadline*, and *period*. Other real-time and quality-of-service parameters can also be added to the *RT_Environment*. A *RT_Manager* class contains a *RT_Environment* structure and methods for setting the *RT_Environment* attributes, starting a TDMI, and completing a TDMI.

In a TDMI, the client expresses real-time constraints on a CORBA method invocation as attributes of a *RT_Environment* structure. Our RT CORBA run-time system attaches the *RT_Environment* structure to all execution that results from the client's TDMI request. Other parts of our RT CORBA run-time system examine this structure to acquire information necessary to enforce the expressed real-time requirements by doing things such as establishing priority and setting alarms.

Figure 3 shows an example of a RT CORBA client that invokes a TDMI to a *get* operation on a `Sensor_Table` object. The client first creates a *RT_Manager* object (Label 1 in Figure 3). It then binds to the appropriate server (Label 2). Next, it calls the *RT_Manager* functions necessary to set the real-time parameters (Label 3). In Figure 3 the client uses a *Set_Time_Constraint*() method to set a relative deadline of 3 seconds from the current time. Other parts of Figure 3 are explained throughout this section.

Once the real-time parameters are established, our RT CORBA run-time system uses library code to enforce the implied constraints by setting a *transient priority* for the TDMI and by setting operating system alarms to detect when crucial times, such as deadlines, have arrived. A transient priority is a single integer that is derived by the new RT CORBA Global Priority Service based on the information in the *RT_Environment* for the TDMI. The Global Priority Service ensures that the transient priority is mean-

```
      #include Sensor_Table.hh  // from IDL compiler
      #include RT_Manager.h    // from RT CORBA
      #include Sensor_Table_i.h // from impl.
          :
(1)    RT_Manager rt_mgr; // create RT_Manager obj.
       Sensor_Table* Sensor_Table_Obj;
         :
       int main() // main procedure of CORBA client
       {
         :
         // bind to the appropriate Sensor_Table

(2)     Sensor_Table_Obj =
         Sensor_Table::_bind("Sensor_Table_Server");

        CORBA::Long track_id = 42;

         try {
            :
            :  // set constraints and
               // scheduling parameters

              // deadline = NOW + 3 seconds
(3)      rt_mgr.Set_Time_Constraint_Now(BY,REL,3,0);

(4)      rt_mgr.Start_RT_Invocation();
           // * calculate transient priority
           // * call RT Daemon to register
           // * map transient priority to this
           //     node's priority
           // * set this thread to new priority
           // * arm the timer

(5)      Track_Record track =
             Sensor_Table_Obj->Get(track_id,
                              rt_mgr.Get_RT_Env());
(6)      rt_mgr.End_RT_Invocation();
           // * call RT Daemon and deregister
           // * disarm the timer
           // * restore this thread to  orig. prio.
         }

       // catch RT_Exception
(7)      catch(const RT_Exception &rtp) {
           cout << ``RT_Exception Raised :''
                << rtp.reason << endl;
         }
         :
       }
```

Figure 3: Example TDMI to *get* operation of `Sensor_Table` Object

ingful relative to all other transient priorities in the RT CORBA system. How this is done is described in Section 5.3. In Figure 3, the library code to do this enforcement is contained in the *RT_Manager* method *Start_RT_Invocation*() (Label 4).

The method call in Figure 3 is: `Sensor_Table_Obj->Get(track_id, rt_mgr.Get_RT_Env())` (Label 5). Note the inclusion of the *RT_Environment* that was set earlier and is returned by the *RT_Manager*'s *Get_RT_Env* method. After the method invocation, the *RT_Manager* *End_RT_Invocation*() call (Label 6) ends the TDMI by re-setting the client's priority to its value before the client performed the TDMI, and disarms the deadline timer.

Our RT CORBA run-time system also augments the CORBA exception mechanism to handle real-time exceptions. These exceptions are derived from the attributes set in the *RT_Environment*, such as a deadline and/or a period. In the example of Figure 3 a violation of the expressed deadline is caught as a CORBA exception (Label 7). Our prototype implements the mapping of deadline exceptions to CORBA exceptions by catching the signal sent by the POSIX alarm that was set for deadline and then raising the CORBA exception in library code.

5.2 Global Time Services

For expressed timing constraints to be meaningful in a distributed system, a common global notion of time must be supported. Our RT CORBA system implements this by synchronizing the clocks (our prototype uses a variant of the NTP protocol [Mil91]) and by providing a Global Time Service, which clients and servers can call to get the current time. In the example of Figure 3, the Global Time Service is called in the `rt_mgr.Set_Time_Constraint_Now(BY,REL,3,0);` call (Label 3) due to the `REL` paramenter that specifies a deadline (`BY`) relative to the current time. That is, the library code calls the Global Time Service to get the current time, and then adds three seconds to establish the deadline.

5.3 Global Priority Service and Distributed Real-Time Scheduling

Real-Time scheduling is performed by the our CORBA run-time system in cooperation with other components, such as the local real-time operating systems' schedulers. Recall that this scheduling is based on a transient priority, which is a single integer that is derived from a function of the attributes in the *RT_Environment* for a TDMI. The transient priority is only valid while the TDMI is active. That is, the client and all execution on its behalf assume the transient priority during the execution of the TDMI, but the client resumes a previous priority when the TDMI completes. Schedulers and queues throughout the distributed RT CORBA system, such as RT POSIX priority-based operating system schedulers, use these transient priorities to order all execution that is associated with a TDMI.

The transient priority for each TDMI is established by our RT CORBA Global Priority Service. This service uses a uniform function (uniform for all clients and servers in the system) to compute transient priority as a function of the attributes in the *RT_Environment* object associated with the TDMI. Our prototype uses a function to compute transient priorities that orders priorities based on the *importance* attribute first, and then based on the *deadline* attribute - essentially establishing a global *earliest-deadline-first within importance level* scheduling policy throughout our RT CORBA system. The call to the Global Priority Service in the example of Figure 3 is made in the library code for the `rt_mgr.Start_RT_Invocation()` (Label 4). Changing the calculatation 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 RT CORBA Global Priority Service.

The implementation of the Global Priority Service in the our prototype is accomplished in a combination of library code and a *RT Daemon* process running on each node. The library code calculates the transient priority. The RT Daemon on each node maps the transient priority to the priorities available on the local real-time operating system. In our prototype, which uses RT Solaris operating systems with 60 local priorities, 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 TDMIs on a Solaris node, the mapping would ensure the highest probability of each TDMI being at a unique priority.

Additionally, the RT Daemon enforces *aging* of transient priorities. Aging is the process of increasing priority as time goes on, which is necessary in dynamic earliest-deadline-first scheduling. The RT Daemon keeps track of the transient priorities on its node. The RT Daemon increases a TDMI's transient priority if, due to the passage of time, the TDMI's priority is too low compared to a newly-arrived TDMI. The aging facility can be "turned off" for real-time scheduling policies that don not require aging, such as a static rate-monotonic-based policy.

5.4 Real-Time Event Service

Recall from Section 2, that the current CORBA event service allows for the exchange of named events in the CORBA system. For instance, a client might synchronize with another client by waiting for that client to generate a CORBA event. Our RT CORBA system has implemented a modified *RT Event Service* that prioritizes the delivery of events and delivers the time that the event occurred. Prioritized events are based on the transient priorities of the producers and consumers and are important to maintain global real-time priority scheduling. Delivery of the (global) time of the event occurrence is important to allow events to be used to establish timing constraints relative to them.

Our implementation of a RT Event Service is based on IP multicasting [Moy] and takes advantage of the multithreaded environment of Solaris 2.5. Each node has a CORBA *Event_Channel* interface [OMG96] and is configured to "listen" to a pre-defined IP multicast group. Each real-time event has a unique event ID number, which is mapped to the IP address for the multicast group. Suppliers transport real-time event data to each RT Event Channel by multicasting to its IP address. Event consumers can wait for delivery of real-time events to the IP multicast groups associated with the events, or they can invoke the local RT Event Channel to retrieve the real-time event. In our prototype, each RT Event Channel buffers the incoming events in priority order so that consumers can look for the buffered high priority real-time events first. If the real-time event data is not in the buffer, then the RT Event Channel raises a RT exception to the consumer, which is handled as described in Section 5.1.

5.5 RT Concurrency Control Service

Recall from Section 2, that CORBA provides a Concurrency Control Service to maintain consistent access to servers. Our prototype RT CORBA system includes a RT Concurrency Control Service that implements *priority inheritance* [Raj91]. When a TDMI requests a lock on a resource from the RT Concurrency Control Service, the TDMI transient priority is compared to those of all TDMI's holding conflicting locks on that resource. Conflicting clients with lower priorities are raised to the requesting TDMI's priority, and the requesting TDMI is suspended. Whenever a lock is released, the releasing TDMI resets its priority to that of the highest priority TDMI it still blocks (this is possible since clients can hold several locks of different types). 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.

5.6 Products and Applications

This NRaD/URI RT CORBA software is packaged as a set of daemon processes, library code, and IDL definitions, that are suitable for extending commercial CORBA software systems that execute on RT POSIX-compliant operating systems. Currently it is being delivered to several companies and organizations for inclusion into their CORBA software including Iona Technologies, Tri-Pacific Corporation, Computing Devices International, MITRE, and several US Navy programs.

6 Conclusion

The specification and development of RT CORBA is still emerging. The OMG's RT CORBA SIG is finalizing its whitepaper describing the philosophy and requirements for RT CORBA [RTSIG96]. The OMG had a Request For Information in Feburary 1997 in which vendors and end-users were encouraged to describe their product/need for RT CORBA. These results are available at [RTSIG97]. The RT SIG plans to issue Requests For Proposals (RFPs) within the next year. These RFPs will ask vendors to propose actual additions/modifications to the standards to support real-time.

Research and prototyping has been conducted and in conjunction with the RT SIG efforts, Section 5 described the on-going NRaD/URI development of a RT CORBA system which is focusing on some of the features for dynamic expression and enforcement of end-to-end timing constraints. CORBA clients can now express timing requirements, such as deadlines, importance and quality of service, on requests that they make to servers. Once these requirements are specified, the new and/or extended object services provide for their enforcement. The Global Priority Service ensures that all CORBA requests are scheduled at all points in the distributed system according to the same (real-time) policy. The Real-Time Concurrency Control Service provides CORBA object-level locking with priority inheritance. The Real-Time Event Service enforces the distribution of real-time events in priority order with real-time enforcement of event response time. Washington University's ORB is similar to the NRaD/URI approach, but focuses on hard-real-time static priority scheduling. They have also produced interesting results that investigate the impact of networks and protocols on RT CORBA. Lockheed

Martin is developing a similar RT CORBA system. The original RT CORBA development from MITRE is now emphasizing research towards a modular approach to ORB design where ORB components, including RT components, can be configured to meet various requirements.

Tackling the substantial requirements posed by using CORBA in a real-time environment is a monumental undertaking, but necessary if standard, open, distributed computing environments are to be used in real-time applications. The results of the various efforts surveyed in this paper are important steps towards achieving this goal. However, many other steps, including significant research, are still needed to produce a viable RT CORBA specification and implementation.

References

[EB96] E. Bensley, P. Krupp, et. al. Object-oriented approach for designing evolvable real-time command and control systems. In *Proceedings of the Workshop on Object-Oriented Real-Time Dependable Systems*, Feb. 1996.

[GWP96] R. Ginis, V. Wolfe, and JJ Prichard. The Design of an Open System with Distributed Real-Time Requirements. In *Proceedings of the Second IEEE Real-Time Applications Symposium*; June 1996.

[HGSP96] T. Harrison, A. Gokhale, D. Schmidt, and G. Parulkar. Operating system support for a high-performance, real-time CORBA. In *International Workshop on Object-Orientation in Operating Systems: IWOOOS 1996 Workshop*. Oct. 1996.

[KTM97] P.Krupp, B. Thuraisingham, and J. Maurer. Survivability Issues for Evolvable Real-Time Command and Control Systems. *DARPA Information Survivability Workshop*. 1997.

[Mil91] D.L. Mills. Internet time synchronization: The network time protocol. *IEEE Transactions on Communications*, 39(10), Oct 1991.

[Moy] J. Moy. Request for comments - no. 1584; proteon, inc. Network Working Group.

[OMG96] OMG. *CORBAservices: Common Object Services Specification.* OMG, Inc., 1996.

[Pal97] R.Pallack. Real-Time Messages Communication For C3I Systems. Master's Thesis. The University of Rhode Island Department of Computer Science. May, 1997. Available at http://www.cs.uri.edu/rtsorac.

[RTSIG96] The Realtime Platform Special Interest Group of the OMG. CORBA/RT white paper. ftp site: `ftp.osaf.org/whitepaper/Tempa4.doc`. Jan., 1997.

[RTSIG97] The Realtime Platform Special Interest Group of the OMG. CORBA/RT white paper. Feb., 1997. WWW site:

`www.omg.org/library/schedule/Realtime_RFI.htm`

[Raj91] Ragunathan Rajkumar. *Synchronization in Real-Time Systems: A Priority Inheritance Approach.* Kluwer Academic Publishers, Boston, MA, 1991.

[TKSW94] B. Thuraisingham, P. Krupp, A. Schafer, and V. Wolfe. On Real-Time Extensions To The Common Object Request Broker Architecture. In *Proceedings of the Object Oriented Programming, Systems, Languages, and Applications (OOPSLA) Workshop on Experiences with CORBA*. Oct. 1994.

[WBTK95] V. Wolfe, J. Black, B.Thuraisingham, and P. Krupp. Towards Timed Distributed Method Invocations. In *The Proceedings of the Fourth International High Performance Computing Conference.* December 1995.