An adaptive scheduling service for Real-Time CORBA

Alexandre Cervieri[1], Rômulo Silva de Oliveira[2], Cláudio F. Resin Geyer[1]

[1]Universidade Federal do Rio Grande do Sul - UFRGS
Instituto de Informática - Av. Bento Gonçalves, 9500 – Bloco IV
Porto Alegre, RS – Brasil
{cervieri, geyer}@inf.ufrgs.br
[2]Departamento de Automação e Sistemas - DAS-CTC-UFSC
Caixa Postal 476 - CEP 88040-900
Florianópolis, SC – Brasil
romulo@das.ufsc.br

Abstract. CORBA is an important standard middleware used in the development of distributed applications. It has also been used with distributed real-time applications, through its extension for real-time systems, RT-CORBA. RT-CORBA includes many mechanisms to reduce the non-determinism associated with ordinary CORBA. These mechanisms can be used to provide guarantees for hard real-time systems if the right support from the operating system and network protocols is available. RT-CORBA mechanisms can also be used to improve the timing behavior of soft real-time applications, when the lower layers are not able to provide guarantees. This paper proposes an adaptive scheduling service in the context of RT-CORBA to support the implementation of distributed soft real-time applications. The proposal is based on the adaptation of task periods, so as to reduce system load while still trying to meet the original deadline of all tasks. This is a best-effort approach that dynamically provides graceful degradation in case of overload. The adaptive service proposed in this paper is validated by a set of experiences based on mechanisms of RT-CORBA and TAO, the ORB implementation used.

1 Introduction

The large-scale availability of computer networks has motivated the search for ways to facilitate and to accelerate the development of applications in distributed systems. CORBA (Common Object Request Broker Architecture), created as an initiative of OMG (Object Management Group), is a middleware that provides a high degree of language and platform independence for the application [1] [2].

However, CORBA at first had no mechanisms to define and guarantee temporal behavior, as well as QoS requirements. These initial limitations made CORBA not appropriate for real-time applications. In order to overcome these deficiencies, OMG started an effort to define a real-time extension to CORBA denominated RT-CORBA [3] [4].

RT-CORBA is still a standard under development. It includes interfaces and mechanisms to allow the definition and the execution of a great variety of real-time applications. Even so, many of the mechanisms that try to maintain the predictability of the applications depend on another aspects such as support from the operating system to guarantee time requirements, a more predictable communication system, scheduling policies and mechanisms that guarantee the execution of tasks at the correct moment [5].

RT-CORBA still has many opened questions on this latter aspect. One of the essential points for the development of a real-time system is the choice of the correct scheduling policy for a given class of applications. Hard real-time applications do not allow deadline misses. Consequently, they demand a scheduling policy capable of providing off-line guarantees. Soft real-time applications allow a run-time scheduling approach, by the use of best-effort policies. In this case some quality degradation is accepted in order to satisfy deadlines, and even delays are allowed to same degree.

The aim of this work is to present an adaptive scheduling service in the context of RT-CORBA to support the implementation of distributed real-time applications. The proposal is based on the adaptation of task periods, with the objective of minimizing system load while still trying to satisfy the deadlines of all tasks. The adaptive service proposed in this paper is validated by a set of experiences based on mechanisms of the RT-CORBA standard and of TAO, the specific ORB implementation used.

This paper is organized so that in section 2 we make a brief revision of standard CORBA and RT-CORBA besides presenting some characteristics of TAO, the implementation used. Section 3 is a small revision of the literature on adaptive mechanisms in real-time systems. Finally, section 4 describes the adaptive scheduling service proposed. Its implementation is presented in section 5 and section 6 shows the results of the experiences. Final remarks appear in section 7.

2 RT – CORBA

Defined by OMG, CORBA is nowadays one of the most complete architectures for the development of distributed systems applications. Many of the benefits associated with CORBA stand in the use of an independent language (IDL – Interface Definition Language) to define object interfaces. It is possible to integrate applications not written for distributed systems and event legacy software in these systems because of the existence of IDL mappings for several languages used in the market.

The OMA (Object Management Architecture) is defined as a reference model for the technology of objects. The development of that architecture followed certain technical objectives so that it brought benefits to the management of objects, such as: conformity, distribution transparency, good performance in remote and local operations, expandable and dynamic behavior, architecture based on name service, access control, concurrency control, among others. OMG (Object Management Group) used the OMA to describe a generalization, a more abstract model, that is to say, a model of higher level, which is implemented by CORBA. It can be said that OMA serves as a “type” and CORBA as an “instance” of that type [6].

The ORB (Object Request Broker) is the communication middleware that allows client objects to send request and receive replies from server objects, which can execute locally or remotely in relation to the client. They have defined mappings of

IDL to several languages to allow the access to the ORB, while not defining details about its implementation. That guarantees some freedom for the developers and assures that a program specified for an ORB will be partially portable to any other ORB that is compliant with CORBA definitions. The Object Adapter, for its time, has the function of controlling the life cycle of the servers. It is responsible for server creation, activation and destruction [7].

Clients and servers can also access ORB services and the CORBA standard services defined by OMG. We can highlight, for its importance in this work: (i) the name service, whose function is to identify an object based on its name, returning its reference, and vice-versa; (ii) the event service, which defines a generic interface for sending messages between multiple sources and multiple destinations. The event service includes an event channel that allows the generation of events without consumers and suppliers having to communicate directly. Two forms of interaction are offered: (i) push, the supplier is active and sends events to the channel; (ii) pull, the consumer is active and tries to receive an event from the channel (this operation may be blocking or not) [1].

In spite of the advantages that it brings to the development of distributed applications, CORBA 2.x does not satisfy all the demands of high performance and real-time applications, mainly because of the following reasons [8] [9]: lack of interfaces to define QoS (Quality of Service), lack of guarantees of QoS, lack of programming characteristics suitable for real-time and lack of optimizations for high performance.

Due to the CORBA drawbacks in supporting real-time applications, OMG have created in 1995 a work group with the objective of extending CORBA specifications. In October 1998, the five proposals presented were united in a single document [8], RT-CORBA (included in the specification of CORBA 2.4). It is nowadays in a process of developing version 2, which will integrate CORBA 3.

RT-CORBA identifies capacities that can be vertically (from the network layer to the application layer and vice-versa) and horizontally (end to end) integrated and managed by an ORB to guarantee the predictability about activities between CORBA clients and servers. Some of those capacities are:

- **Management of the communication infrastructure**: RT-CORBA should provide polices and mechanisms for the management of the communication infrastructure, from the choice of a connection to the exploration of advanced QoS characteristics, including threads interface, thread pools, explicit binding and others;
- **Operating system scheduling mechanisms**: ORBs explore the mechanisms of the operating system to schedule activities at the application level. RT-CORBA considers mainly real-time systems based on fixed priorities. In that way, these mechanisms correspond to the management of priorities of the threads scheduled by the operating system;
- **Real-time ORB**: a real-time ORB should provide standardized interfaces to allow applications to specify their demands of resources to the ORB, besides supporting communication between clients and servers in a transparent way;
- **Real-time services and applications**: a real-time ORB should also create an efficient environment, with end-to-end predictability, for high-level services and applications.

Since OMG does not supply an implementation of CORBA and RT-CORBA, there are alternatives to be analyzed. Among CORBA implementations and, more specifically, RT-CORBA implementations, TAO (The ACE ORB) is an important one. TAO can be considered more than just an ORB to support real-time communication, it is a complete architecture for execution of real-time applications, with as much hard as soft deadlines. The architecture of TAO contains the following characteristics and elements [10]: a real-time I/O sub-system, RT-ORB, protocol GIOP for real-time, RT-POA, optimized IDL compiler and presentation layer, optimizations of the memory management by minimizing the copy of data and means for QoS specification.

TAO is one of the ORBs that presents the most complete implementation of RT-CORBA. However, its implementation of some services differs from the standard definitions. One of those services is scheduling. The TAO scheduling service is responsible for allocating system resources in order to guarantee the needs of the applications. It was designed for hard real-time applications and its main objective is to guarantee that the demands for resources will be satisfied. It is divided in an off-line component and a run-time component. First, the possibility of a correct scheduling of all the operations is analyzed and their priorities are assigned. Run-time components offer fast access to priorities values and coordinate mode change operations.

Another TAO service that was improved for the use in real-time applications was the event service. Basically the method of interaction push was affected [11]: (i) consumers and suppliers of events can specify their demand and execution characteristics using QoS parameters; (ii) correlation and filtering mechanisms are centralized in the event channel; (iii) consumers can specify time-out dependent events.

3 Adaptation Mechanisms for Real-Time Applications

Even in computer systems specifically built for the execution of real-time applications there are many opportunities for adaptation. This can happen because of the load generated by the application not having a well-known limit that can be analyzed off-line. Even if the load has a well characterized demand for resources, it may not be economically feasible to guarantee its behavior in a worst-case scenario.

Although most of the real-time literature is about adaptation through quality of service negotiation, adaptation can also be a unilateral action. In the case of a unilateral adaptation the following scenarios are possible:

- A variation in the application behavior triggers an adaptation of the support.
- A variation in the support behavior triggers an adaptation of the support itself.
- A variation in the application triggers an adaptation of the application itself. In systems where the application can reserve resources, the application itself must manage reserved resources.
- A variation in the support behavior triggers an adaptation of the application. This situation is very common in distributed systems, where variations of the response time can be associated with sending messages on the network.

Real-time applications in a distributed environment are subject to variations in the response time of its tasks. Those variations can be caused by delays in message transmissions on the network or changes in some processing node used by the application. There is the need for a continuous adaptation during the application lifetime. Adaptation mechanisms described in the literature that can be used in this context will be examined in this section:

- **Delaying a task**. The simplest and more frequent form of adaptation is simply to relax the deadline concept. One of the first papers proposing this approach appears in Jensen et. al. [12]. They propose that the conclusion of each task contributes to the system with a benefit and the value of this benefit can be expressed as a function of the instant of task conclusion (time-value function).
- **Changes in the task period**. In a real-time application a lot of tasks are executed periodically. In general, these tasks have their period defined off-line. A way to provide adaptability in the application is to allow this period to vary dynamically during its execution. This way, the quality of the application, represented here by the period of its tasks, would be adapted to the performance of the platform where it executes.
- **Canceling a task execution**. A more radical form of flexibilization is simply not to execute some tasks when the performance is below the desired level. In the case of applications with repetitive tasks, it is possible to cancel a specific task activation or to cancel the task completely.
- **Changes in the task execution time**. In this adaptation mechanism, tasks are scheduled so they respect their respective deadlines. In case of overload, the task execution time is reduced. In order to implement that, it is necessary for each task to have options of the type quality versus execution time. This approach is usually known as Imprecise Computation [13].

There are several proposals in the literature about how to measure system quality and how to act on the system to maximize its quality. For example, in Lu et. al. [14] the quality of a system is measured by the deadline miss rate and by the rate of system utilization in a given measuring window. Control is implemented by changing the operation mode of tasks and by deciding to accept or reject tasks for execution. Beccari et. al. [15] also identifies overloads in the system through the processor utilization and the control is implemented through graceful degradation of task periods. Other works found in the literature that follow the same direction can be found in Welch et. al. [16], Shin & Meissner [17] and Abdelzaher et. al. [18] [19].

Table 1 presents a qualitative analysis of some adaptive approaches in the literature.

Table 1. Some adaptive techniques in the literature

Proposal	Measuring	Acting	Experiences	Platform
Lu et. al. [14]	Deadline miss rate and system usage within a measuring window.	Changes in the task operation mode and take the decision when accept or reject new tasks.	Simulation	Single processor.
Brandt et.	System usage.	CPU usage determines	Linux/UNIX	Single

al. [15]		the system operation mode (requirements and relative benefit).	systems.	processor.
Beccari et. al. [20]	System usage.	Changes in the period of soft real time tasks (hard real time tasks are not affected), graceful degradation.	Simulation and prototype still under development for a VME-based system.	Single processor.
Shin & Meissner [17]	System quality (each task has a benefit value depending on the period).	Changes in the period of tasks and/or tasks relocation.	Simulation (random tasks and sample application)	Multi processor.
Abdelzaher et. al. [19]	Throughput in packets per second.	Changes in the QoS level (communication) required by the tasks.	The Open Group (TOG) 7.2 kernel	Client/Server (comm. system)
Abdelzager et. al. [18]	The system benefit is calculated when every new task arrives.	Accept or reject the execution of tasks modules or the whole task, following the QoS level negotiated.	RTPOOL implemented over OSF Mach RT-mk7.2	Multi processor.

4 Adaptive Scheduling Service

In the previous section many adaptation mechanisms were introduced. Considering that a lot of real-time applications have its structure based on periodic activities, we opted for an approach directed to this type of activity. The proposal of this work was inspired by Lu et. al. [14] and other works that use a feedback loop to control the system during overloads.

In the same way of Lu et. al. [14], our control of the system is based on information collected from all activities that compose the system. This information is analyzed by a periodic process, which will be called as AdaptiveService in this work. Lu et. al. [14] includes adaptation in the system by considering the utilization rate of the system (U(t)) and the deadline miss rate (MR(t), miss ratio function) of the tasks inside a measuring window (MW miss-ratio window). The measure of deadline miss-ratio is quite adapted for systems with firm deadlines, where there is no benefit in concluding a task after its deadline.

Unlike the approach used by [14], which was created for a monoprocessor, we deal with the execution of soft real-time tasks on a distributed system. We decided to quantify the system quality by comparing the deadlines and the response times of the real-time tasks. This quality measure will be denominated delay profile (DY(t)), which is observed within a measuring window called delay window (DW). In this work it is assumed that the deadline of each task is equal to the nominal period of the same task (D=P). Therefore, DY(t) can be expressed as:

$$DY(t) = \sum_{t-DW}^{t} \frac{(R_i - D_i)}{N} \qquad (1)$$

t: current instant of time
DW: delay window, the time window used for calculation of DY(t)
R$_i$: response time of task i for a certain activation
D$_i$: task deadline
N: number of conclusions (of all tasks) within window DW

Actuation is done on the task periods. This actuation could be made for each task in a specific way. For example, there could be an adaptation factor especially calculated for each task by considering each task importance. In this work it is assumed that all tasks have the same importance. DY(t) is used for the calculus of an actuation factor on the period of each task in the following way:

$$\begin{aligned} &factor = K_P * E(t) \Rightarrow \\ &factor = K_P * (DY(t) - SDY) \Rightarrow \\ &factor = K_P DY(t) - K_P SDY \end{aligned} \qquad (2)$$

where E(t) denotes the value of the error in a certain instant and it can be expressed by DY(t)-SDY, where SDY is the desirable average delay for the system. K_P (proportional constant) and SDY are values defined by the designer and/or programmer of the system according to mathematical calculations or previous experimentation. A small value for K_P will define a smooth actuation on the system.

The factor is used by the AdaptiveService for the calculus of the effective period of each task by:

$$P_{effective} = P_{\min} + factor(P_{\max} - P_{\min}) \qquad (3)$$

Besides the values of K_P, SDY and DW, the mechanism for the designer/programmer to specify the actuation also includes the values of P_{min} (minimum period) and P_{max} (maximum period) of each task. According to the equation above, the AdaptiveService is capable of varying the period of each task (the effective period) according to the factor calculated previously, but always within the limits established by the minimum and maximum period of each task.

The computation of DY(t) along the time considers that the deadline is always equal to the nominal period of each task. Along the execution, the effective period can be altered according to the adaptation factor calculated by the adaptive service. Even so, it is always used the same deadline for each task, which is equal to the respective nominal period. The adaptation is achieved by varying the task period and not the task deadline.

5 Adaptive Scheduling Service Implementation on TAO

The proposed adaptation mechanism is designed for applications composed by periodic tasks. Applications can also contain aperiodic tasks and passive objects. The class diagram of one possible implementation of the mechanism can be seen in figure 1. The main elements of figure 1 are detailed in the next paragraphs.

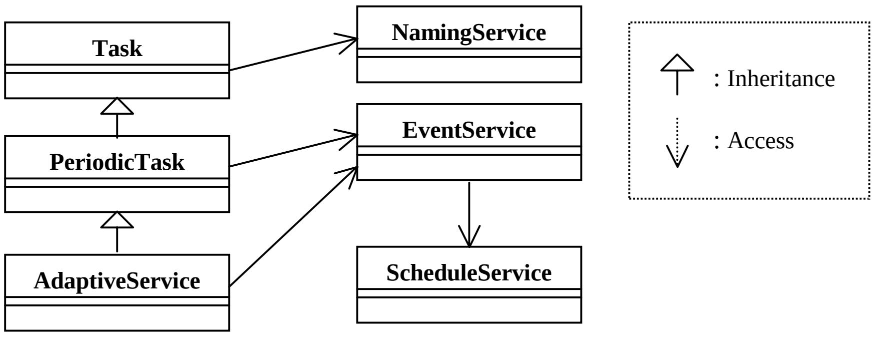


Fig. 1. Class diagram of the adaptive model implementation

Task
It represents aperiodic tasks and passive objects. It is implemented as a servant that will incarnate a CORBA object. At the moment of its creation it registers a name with the name server (NamingService) so that the active objects can find them and make method invocations.

Each task is a CORBA object and it is activated in its own POA, configured with the PriorityModelPolicy as ClientPropagated, in order to respect the priority of the objects that make the invocation.

PeriodicTask
It represents the periodic tasks. This class is abstract and cannot be directly instantiated. It requires the definition of a derived class. A periodic task (PeriodicTask) is, ultimately, a consumer that enrolls in the event service (EventService) of TAO to receive time-out events in specific time intervals. This time interval is the period defined in the RT_Info of the task during the off-line phase of the scheduling.

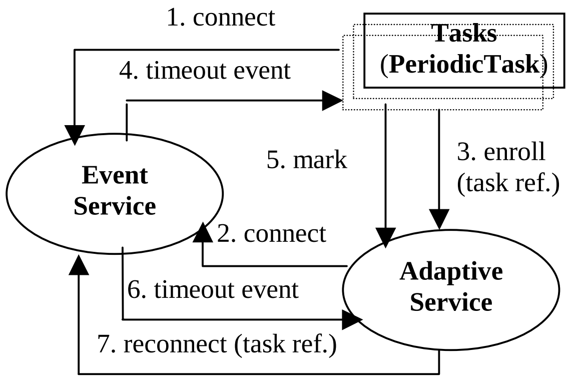
Each PeriodicTask also have an own POA and it is configured with PriorityModelPolicy as ServerDeclared (to use the priority defined in its RT_Info) and a ThreadPool with the number of static and dynamic threads defined by the programmer at the moment instances are created.

AdaptiveService
The adaptive service (AdaptiveService) is implemented as a periodic task that needs to be described as an RT_Info in the off-line phase of scheduling. Its period must be equal to DW and can be configured in the RT_Info.

The adaptive service is responsible for collecting data from the periodic tasks (the aperiodic tasks will not be considered by the service) and for posterior actuation on the system. This process can be seen in figure 2.

In order to use the scheduling service of TAO it is necessary to divide the work in two phases: off-line and run-time. At the off-line phase the timing properties of each task (including the adaptation service itself) are registered by using data structures called RT_Infos.

At the beginning of the run-time phase each periodic task should obtain a reference for the adaptation service. It then should invoke an adaptation service method to enroll itself, also informing the values of minimum and maximum period. Each periodic task must be implemented as an instance of a subclass of PeriodicTask. That class provides mechanisms to obtain the task response time by using timers and then to invoke automatically the method mark of the adaptation service in order to inform the value measured.



1. Each task connects to the event service to receive timeout events in intervals defined by the period configured in the RT_Info in the off-line stage of the application.
2. The adaptive service also connects to the event service to receive timeout events in accordance to the required period.
3. Each task enrolls itself in the adaptive service informing Pmax, Pmin and also its reference.
4. After all tasks being connected, the event service starts generating timeout events in intervals defined by the tasks periods (which are present in the RT Info).
5. After each task execution, the response time is measured and sent to the adaptive service.
6. In each activation, the adaptive service calculates the value of DY(t) within the DW window and the new factor.
7. In case of the actual factor is different of the last one, the Adaptive Service requires the the event service reconnect all the tasks sending the new effective period.

Fig. 2. Operation of the adaptive scheduling service

The adaptation service has, therefore, the function of calculating the value of DY(t) at each activation (time interval DW) and of calculating the value of the adaptation factor to be applied to the periods of all periodic tasks. The effective period calculated for each task is then used to request to the event service a new connection to each task for time-out events, but with this new value as time interval between successive activations. The response time of the adaptation service to load increases is, therefore, dependent of the response time of the event service to re-connecting the consumers for new intervals of time-out events. Situations that demand immediate response to sudden increases of load would need a fast event service when dealing with the reconnections of consumers.

6 Experiences

6.1 Application structure

There are many classes of distributed real-time applications that could be used to test the adaptation mechanism proposed in this work.

A class of real-time applications that presents many possibilities for studying is the class of industrial automation systems. This kind of system has many different types of tasks, aperiodic, sporadic, periodic, with different levels of importance.

This work sought to identify and to represent some of the types of tasks found in this type of system.

- **Sensor** and **Actuator** were the elements of automation applications chosen to represent passive objects. They are derived from class Task and they have methods defined in IDL that can be invoked by active elements of the system.
- We identified three possible types of periodic tasks in automation systems. They will be defined as subclasses of PeriodicTask: (i) **Logger** represents the tasks that are responsible for the collection of data in order to log the system status along the time, it has a period reasonably high (one second); (ii) **Operator** represents monitoring processes (collection and presentation of the information) that interact with a human operator; (iii) class **Alarm** represents monitoring processes that can identify problems in the system and they are supposed to have a period smaller than that of other tasks in order to be always updated with the last data collected in the system.

The application created simulates the behavior of each class defined above and reproduces the oneway or twoway dependencies. The next section presents details on the registering of these tasks, their temporal restrictions and dependencies.

6.2 Conditions of the experiences

In order to validate the proposed adaptation mechanism, we made several experiments considering the same test scenario. After considering the application class used as test base (industrial automation), where in most cases the initial configuration is seldom altered, we assumed the same number of tasks and the same number of hardware equipments (computers, sensors, actuators, etc.) for all experiments. Experiments differ only about the details of the proposed adaptation mechanism.

For all the experiments described in this section, a network formed by two machines was assumed, with the hardware and software configuration presented in table 2.

Table 2. Hardware and software configuration for the experiments

Machine	Hardware	Software
Node 1	Pentium III 650MHz	Windows 2000 Advanced Server operating system Visual C++ 6.0 compiler

	128MB RAM	ACE version 5.2 TAO version 1.2
Node 2	AMD K7 Duron 750MHz 128MB RAM	Windows 2000 Advanced Server operating system Visual C++ 6.0 compiler ACE version 5.2 TAO version 1.2

Tasks were distributed among the computers in a static and arbitrary way. Table 3 shows the task allocation used in all experiments, as well as their execution times, period, dependencies registered with the TAO scheduling service and the respective priority calculated by that TAO service.

Table 3. Task allocation to nodes

Machine	**Tasks**						
	Time	**Worst Exec. Time (ms)**	**Period (ms)**	**Dependencies**		**Priority**	
				oneway	**twoway**	**CORBA**[1]	**Win 2k**
Node 1	sensorA	10	-	-	-	2	1
	actuatorA	20	-	-	-	0	15
	actuatorB	20	-	-	-	0	15
	alarm1	20	100	actuatorA actuatorB	sensorA sensorB	0	15
	operator1	300	500	-	sensorA sensorB	1	2
Node 2	sensorB	10	-	-	-	2	1
	alarm2	20	100	actuatorA actuatorB	sensorA sensorB	0	15
	operator2	300	500	-	sensorA sensorB	1	2
	logger1	500	1000	-	sensorA sensorB	2	1
	logger2	500	1000	-	sensorA sensorB	2	1
	Adaptive Service	40	1000	-	-	2	1
	Naming Service	-	-	-	-	-	-
	Event Service	-	-	-	-	-	-
	Schedule Service	-	-	-	-	-	-

[1] The value zero represents the higher priority and the higher value the lower priority.

6.3 Measured Results

Initially we executed the application without the adaptive scheduling service (AdaptiveService), but we had calculated DY(t) for each period during its execution. Table 4 shows the averages of period and delay for each periodic task and figure 3 shows the evolution of DY(t) along the time for executions of two minutes.

Table 4. Average values for period and delay without adaptation

Task	Period Mean (P)	Delay Mean (R-D)[2]
alarm1	104,58 ms	1248,34 ms
alarm2	103,14 ms	226,56 ms
operator1	500,07 ms	-230,78 ms
operator2	500,08 ms	-348,12 ms
logger1	1000,3 ms	-780,59 ms
logger2	1000,71 ms	-732,34 ms

The negative values for the delay in table 4 indicate that the task finished before its deadline.

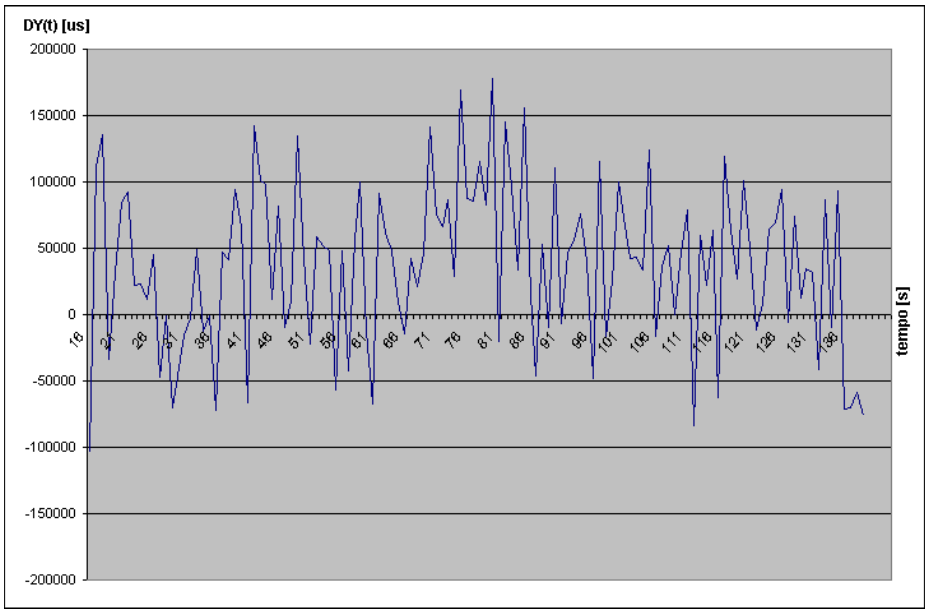


Fig. 3. DY(t) for a system without adaptation

By analyzing table 4 and figure 3 one observes that there are significant delays affecting specially the two tasks with larger priority (alarm1 and alarm2). In spite of possessing the largest priority, those tasks are also the most demanding in terms of

[2] R is the response time of a task in a specific activation and D is the task deadline. We consider the deadline equal to the nominal period of a task (D=P).

processor time. They end up suffering delays for this reason, for the blocking caused by Windows and also for the characteristics of the TCP/IP protocol that maintains sending queues based on the FIFO police.

Table 5. Average values with adaptation, initial parameters

Task	Period Mean (P)	Delay Mean (R-D)
alarm1	115,82 ms	34,50 ms
alarm2	115,47 ms	24,78 ms
operator1	687,87 ms	-348,11 ms
operator2	687,16 ms	-321,7 ms
logger1	1668,80 ms	-713,25 ms
logger2	1659,74 ms	-640,37 ms

One of the first tests with the adaptive service used K_P=1/250000 and SDY=-50000 us. The results can be seen in table 5 and figure 4. The delays are a little smaller (but still existent) than those found before, but the adaptation is still unstable. It has oscillations around the value determined for SDY. Besides, it is noticed that the delay decreased even with the tasks (alarm1 and alarm2) executing with smaller average periods.

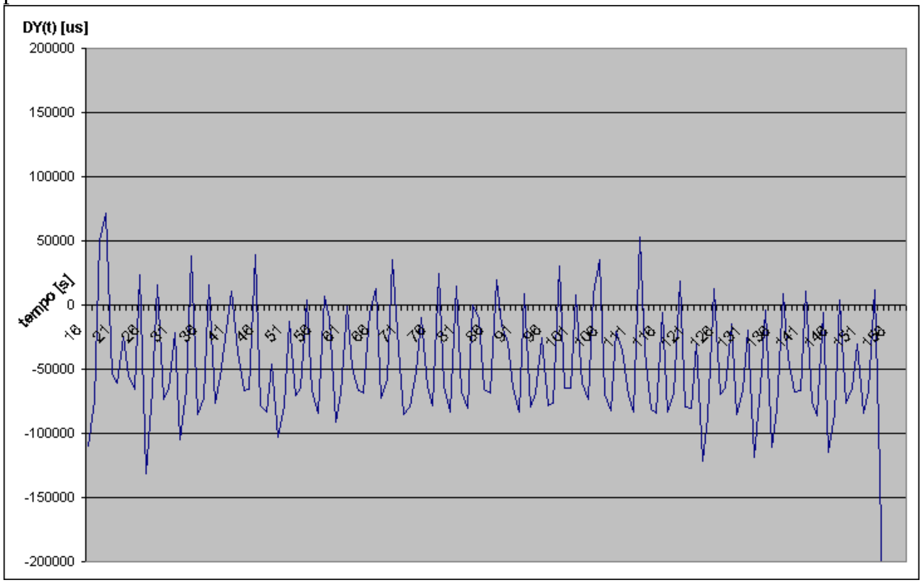


Fig. 4. DY(t) for a system with adaptation, initial parameters

The stability of the control implemented by the adaptation mechanism depends on the values of K_P and SDY. The adjustment of these constants could be made through mathematical calculus from control engineering, since we use a simple proportional control. Even so, it was not an objective of this work to deepen the study in this area. It was sought, through experimentation, to alter the values of constant K_P and SDY so that the actuation on the system becomes smooth. In the case of small oscillations, we would like to maintain the values of DY(t) negative.

After some tests we arrived at the values of K_P=1/420000 and SDY=-60000 us, whose results can be seen in table 6 and figure 5.

Table 6. Average values for a system with adaptation, final parameters

Task	Period Mean (P)	Delay Mean (R-D)
alarm1	113,84 ms	9,81 ms
alarm2	112,44 ms	7,98 ms
operator1	709,66 ms	-339,85 ms
operator2	705,81 ms	-339,99 ms
logger1	1733,3 ms	-734,04 ms
logger2	1720,36 ms	-649,03 ms

It can be observed in figure 5 an increased stability of the system. After the initial peak of load the system acts in a way to improve DY(t). The consequence of making SDY more negative (in the case, -60000 us) is the maintenance of value DY(t) close to value SDY due to the works of the adaptation service. Then, the average delay (showed in table 6) were smaller than what was observed in the system without adaptation. In short, every task was able to have a response time close to their respective deadlines, with only a small change in its period.

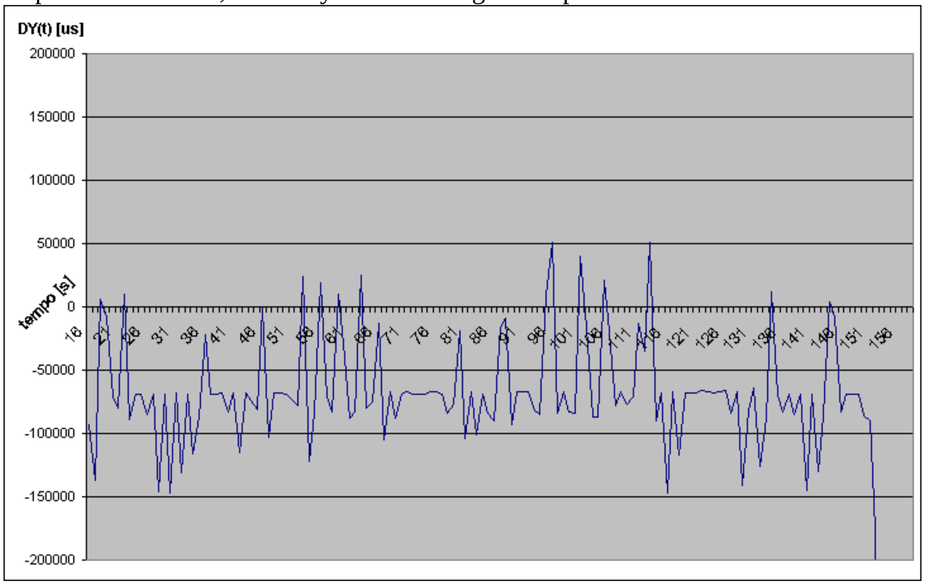


Fig. 5. DY(t) for a system with adaptation, final parameters

The values of the constants K_P and SDY depend on the conditions of each application. As said previously, the control engineer or the system designer must define the appropriate values for those parameters according to mathematical models or experimental measurement done specifically about the application considered.

The results presented in this section show that the proposed adaptation mechanism could be used with real applications as long as the necessary configurations for each application are resolved.

7 Conclusions

CORBA brought advantages for the development of distributed applications. Even so, it presents some drawbacks for real-time applications that impelled the definition of a specific extension, RT-CORBA, which is still under development. Existing implementations try to follow its recommendations, but many times they create their own solutions to make its ORB more predictable and with some extra facilities to create real-time applications.

The most important aspect to be taken into consideration is that CORBA and its extension RT-CORBA are a middleware. They depend much on the operating system and on the hardware used. Because of that, no matter how much RT-CORBA (or, more specifically, an implementation of it, like TAO) tries to solve its definition deficiencies to provide deadline and QoS guarantees, it is dependent of the operating system and network.

This paper presented an adaptive scheduling service for RT-CORBA to be used with real-time applications that tolerate a best-effort approach.

The main focus of concern was periodic tasks with timing requirements. Although the approach presented here will not be feasible for absolutely all applications, it can be used whenever there is some flexibility about the period of tasks. Actuation is based on a feedback loop that supplies information about response times to an adaptive service. We have implemented the adaptation by varying the period of the tasks. The variation of the period is derived from the calculus of what was denoted DY(t), delay profile. The delay profile is the measure of how much early or late each task finished with relation to its deadline (period) within a specific measure window (DW, delay window).

The idea of the mechanism presented in this paper is basically to sacrifice the task period to maintain under control its delay. This way it is exercised a control over system behavior in case of overload. Unlike most systems, where an overload generates random degradation, the approach proposed here allows a controlled degradation in the moments of overload. It is also important to notice that the adaptation is dynamic, which means it is capable of answering to load oscillations that occur during the execution of the system. The mechanism automatically obtains from the system the largest possible quality, while respecting the deadlines of the tasks.

Validation of the proposed mechanism was made through its implementation using RT-CORBA mechanisms and scheduling and event services available in ORB TAO. The results show that for a given experimental application the mechanism improved system quality, as defined by the comparison between response time and deadline of each task.

The adaptation mechanism used acts in a homogeneous way on the tasks, so the same adaptation factor is used with the period of all tasks. Improvements may be obtained by the calculation of a specific adaptation factor for each task, also considering each task importance. The choice of the common minimum multiple of the task periods as the period of the adaptation service was ad-hoc, although many works in the literature do the same. It is an open question if this choice is the best for any application. Possible work in the future include experiences to verify those aspects and also to analyze the scalability of the solution, since the tests described in this paper used only two computers. The replacement of TAO event service for another specifically designed to generate periodic events and to make fast changes in established connections may improve the response time of the adaptive service as a whole.

References

1. Mowbray, Thomas J.; Ruh, William A.; Inside CORBA: Distributed Object Standards and Applications. USA: Addison-Wesley, 1998.
2. Orfali, Robert; Harkey, Dan; Client/Server Programming with JAVA and CORBA, 2. ed. USA: Wiley Computer Publishing, 1998.
3. Feng, W.; Syyid, U.; Liu, J. W.-S. Providing for an Open, Real-Time CORBA. Disponível por WWW em http://www.researchindex.com (Dez. 2000).
4. O'Ryan, Carlos; Schmidt, Douglas C.; Kuhns, Fred; Spivak, Marina; Parsons, Jeff; Pyarali, Irfan; Levine, David L. Evaluating Policies and Mechanisms to Support Distributed Real-Time Applications with CORBA. Disponível por WWW em http://www.cs.wustl.edu/~schmidt/corba-research-realtime.html (Nov. 2000).
5. Gill, Christopher D.; Levine, David L.; Schmidt, Douglas C. The Design and Performance of a Real-Time CORBA Scheduling Service. Disponível por WWW em http://www.researchindex.com (Dez. 2000).
6. Pope, Alan; The CORBA Reference Guide: Understanding the Common Object Request Broker Architecture. USA: Addison-Wesley, 1998.
7. Henning, Michi; Vinoski, Steve; Advanced CORBA Programming with C++. USA: Addison-Wesley, 1999.
8. OMG; Realtime CORBA. Alcatel; Hewlett-Packard Company; Lucent Technologies, Inc.; Object-Oriented Concepts, Inc.; Sun Microsystems, Inc.; Tri-Pacif. OMG Document orbos/98-01-08. Disponível por WWW em http://www.cs.wustl.edu/~schmidt/PDF (Oct. 2000)
9. Schmidt, Douglas C.; Levine, David L.; Cleeland, Chris. Architectures and Patterns for Developing High-performance, Real-time ORB Endsystems. Setembro, 1998. Disponível por WWW em http://www.cs.wustl.edu/~schmidt/corba-research-realtime.html (Oct. 2000)
10. Schmidt, Douglas C.; Levine, David L.; Mungee, Sumedh. The Design of the TAO Real-Time Object Request Broker. Computer Communications Journal, 1997. Disponível por WWW em http://www.researchindex.com (Dez. 2000).
11. Harrison, Timothy H.; Levine, David L.; Schmidt, Douglas C.; The Design and Performance of a Real-Time CORBA Event Service. Object-Oriented Programming Systems, Languages and Applications (OOPSLA), Outubro 1997. Disponível por WWW em http://www.researchindex.com (Aug. 2001).
12. Jensen, E. D.; Locke, C. D.; Tokuda, H.; A Time-Driven Scheduling Model for Real-Time Operating Systems. Proceedings of the IEEE Real-Time Systems Symposium, pp. 112-122, Dez. 1985.
13. Liu, J. W. S. et. al., Algorithms for Scheduling Imprecise Computations. IEEE Computer, pp.58-68, May 1991.
14. Lu, Chenyang; Stankovic, John A.; Abdelzaher, Tarek F.; Tao, Gang; Son, Sang H.; Marley, Michael; Performance Specifications and Metrics for Adaptive Real-Time Systems.21st IEEE Real-Time Systems Symposium, Nov. 2000.
15. Beccari, G.; Caselli, S.; Reggiani, M.; Zanichelli, F.; Rate Modulation of Soft Real-Time Tasks in Autonomous Robot Control Systems. 11th Euromicro Conference on Real-Time Systems, England, June 1999.

16. Welch, L. R.; Shirazi, B. A.; Ravindran, B.; Adaptive Resource Management for Scalable, Dependable Real-Time Systems: Middleware Services and Applications to Shipboard Computing Systems. IEEE Real-time Technology and Applications Symposium, June 1998.
17. Shin, K.G.; Meissner, C. L.; Adaptation and Graceful Degradation of Control System Performance by Task Reallocation and Period Adjustment. 11th EuroMicro Conference on Real-Time Systems, June 1999.
18. Abdelzaher, T. F.; Atkins, E. M.; Shin, K. G.; QoS negotiation in real-time systems and its application to automatic flight control. IEEE Real-Time Technology and Applications Symposium, June 1997.
19. Abdelzaher, E. M.; Shin, K. G.; End-host Architecture for QoS-Adaptive Communication. IEEE Real-Time Technology and Applications Symposium, June 1998.
20. Beccari, G.; Caselli, S.; Reggiani, M.; Zanichelli, F.; Rate Modulation of Soft Real-Time Tasks in Autonomous Robot Control Systems. 11th Euromicro Conference on Real-Time Systems, England, June 1999.