An Adaptive Scheduling Approach in Real-Time CORBA

[1]Carlos Montez [1]Joni Fraga [2]Rômulo de Oliveira [1]Jean-Marie Farines
montez@lcmi.ufsc.br fraga@lcmi.ufsc.br romulo@inf.ufrgs.br farines@lcmi.ufsc.br
[1]LCMI - Depto de Automação Sistemas - Univ. Fed. de Santa Catarina
Caixa Postal 476 - 88040-900 - Florianópolis - SC - Brazil
[2]Inst. de Informática - Univ. Fed. do Rio Grande do Sul
Caixa Postal 15064 - 91501-970 - Porto Alegre - RS - Brazil

Abstract

CORBA is an emerging middleware infrastructure with open standardization that is receiving a good acceptance since it makes easier to program distributed objects. CORBA is being extended through the specification of interfaces and necessary abstractions to support applications with real-time constraints. These new abstractions will enable a variety of programming models for real-time applications. This paper presents an adaptive programming model for distributed real-time applications using CORBA concepts. The model combines the time polymorphic invocation technique with the (m,k)-firm guarantee.

1. Introduction

CORBA (Common Object Request Broker Architecture) is a middleware whose objective consists to facilitate the distributed programming in heterogeneous environments. Its open specifications [15], standardized by OMG (Object Management Group), are receiving growing acceptance. The architecture of reference is composed by an ORB (Object Request Broker), services and common facilities objects. The ORB is a kind of software bus, managing the communications among distributed objects in a transparent way. The services and facilities have standardized interfaces that support some basic functions used by the application objects.

Real-time systems are usually implemented using proprietary technologies and platforms that increase their development and maintenance costs. Recently, many distributed real-time systems are adopting the object orientation paradigm, and CORBA has been pointed as an open technology to be adopted in many real-time applications, when the aim is of extending the portability, interoperability, flexibility and reducing the costs of real-time systems.

The use of CORBA, as well as other open technologies, in distributed real-time systems is a recent area of research. In spite of that, this tendency comprises a large range of applications, involving systems that interact with deterministic environments, such as embedded applications and process control; as well as large scale systems, characterized by a dynamic computational load, such as multimedia applications on the Internet. Many of these application domains require real-time guarantees (off-line or on-line) from the networks, operating systems and middleware components, for its time constraints. However, the current CORBA standards are inadequate to support the real-time requirements.

CORBA was created targeting applications in general, which desire transparency in the distribution and in the way that resources are managed. On the other hand, these transparencies (*e.g.* location and migration transparencies) often are not acceptable in real-time applications. Besides, the CORBA interoperability protocol (IIOP - Internet Inter-Orb Protocol) is designed for general purpose systems, sending messages using TCP/IP connections. The TCP lacks predictability making the IIOP an inadequate protocol for several real-time applications. Furthermore, the current version of CORBA provides only two modes of invocations: synchronous and deferred synchronous; and this last one is only available by using the dynamic invocation interface. The lack of a really asynchronous model is a restriction that present real-time programmers must handle in some way. Also, CORBA does not have some useful facilities for real-time programming, such as invocation with timeout.

In 1995, a Special Interest Group (SIG) was formed within the OMG with the goal of extending the CORBA standard to target real-time applications. The final specification, the RT CORBA, resulting of this standardization effort, intended to present several abstractions (interfaces and mechanisms) enabling the execution of real-time applications, allowing the definition of a variety of real-time programming models.

The study described in this paper is part of a research we are conveying with the aim of developing a programming model for distributed real-time applications using concepts of RT CORBA. The proposed model has adaptive features: some scheduling decisions are based on the conditions of the system observed through monitoring deadline misses. In environments that do not offer predictability, the adaptive characteristic of the model can be seen as a form of responding to dynamic variations of the environment, by providing several levels of quality in the offered services.

Our innovative model combines two adaptive strategies — the time polymorphic invocation [22] and the (m,k)-firm guarantee [7]. Developers can built their applications using just one or both techniques. The scheduling approach is built by using mechanisms and concepts recently incorporated in the CORBA specifications for real-time programming [18].

This paper is divided in 6 sections. Section 2 describes some features incorporated in the specifications of real-time CORBA. Section 3 presents a discussion on adaptive scheduling techniques. Section 4 introduces our adaptive approach and describes our experience in implementing the model by using some real-time CORBA concepts. Section 5 lists related works. Finally, section 6 presents some final remarks.

2. Real-time CORBA

The lack of middleware support for real-time applications led the OMG in 1995 to create a work group with the goal of extending the CORBA standards. A RFI (Request for Information) document [16] was issued specifying requirements that should be maintained by an ORB supporting applications with time constraints. The RFI document defined requirements for the operational environment, for the ORB and for the services. Examples of operational environment requirements were: multithreading, priority sorted queues, clock synchronization and maximum communication latencies; and examples of ORB and services requirements were: capabilities for expressing and enforcing timing constraints, global priorities and time polymorphic invocations.

In September of 1997, a RFP (Request for Proposal) document [17] was published requesting proposals of real-time CORBA standardization. This first RFP document defined some requirements for priority-based scheduling[1]. The document demanded submissions covering some mandatory aspects: the definition of "schedulable entities", mechanisms for expressing and controlling priorities and for bounding method invocation blocking and priority inversion.

In October of 1998, five proposals previously presented were joined in only one document [18] — the Realtime CORBA 1.0. This document emphasizes the definition of ORB mechanisms, allowing developers to choose the policies for real-time scheduling. There are no assumptions about the underlying operational environment but the document recognize that a POSIX conforming operating system is suitable to obtain predictability. The RT CORBA 1.0 describes threads as the schedulable entities. It specifies interfaces for managing threads that will allow a great portability to applications. Threads will be defined and controlled in an uniform way, no matter the execution platform uses Solaris, NT, or POSIX.1c threads.

Pool of threads and *request queues* are some resources that applications can manipulate directly. A pool of threads can be created for processing method requests received by a server. A request queue can be created and associated with that pool of threads (Figure 1).

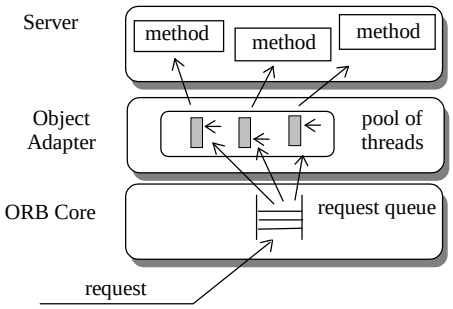

Figure 1. An example of using new RT CORBA mechanisms.

The *CORBA priority* is a kind of global priority that may cross over different ORBs, carried by invocation messages. Depending of the scheduling approach, this priority may be used by servers to sort the requests received. The RFP introduces two usage models: the *client-priority propagation model* and the *server-set priority model*. Using both strategies, the CORBA priority must be mapped to the operating system priorities before the method execution.

During an explicit binding, several steps must be done to interconnect objects, such as, to locate target objects and to initialize data structures that support the communication among objects. On the client side, the binding can be used to guarantee that their timeliness requirements will be respected (*e.g.*, through the selection of an appropriate transport protocol and of a timeout value for failure detection). On the server side, the binding allows the allocation of the necessary resources for request executions, such as threads and queues.

The main mechanism proposed in the CORBA specifications, which introduces flexibility in the application programming is the *interceptor*. Interceptors are application objects that may have their operations called by the ORB at different moments of a client invocation. It is mainly used for enforcing application policies. Interceptors were initially specified in the CORBA Security Service, and standardized by OMG in CORBA 2.2 [15]. However, that mechanism is sub-specified, and there is another OMG RFP document [19], with the proposal of obtaining some extensions to interceptor interfaces.

[1] A new RFP — the Real-Time CORBA 2.0 RFP — is expected for specifying dynamic scheduling, and facilities to deadline-based scheduling.

The RT CORBA specification is also based on another document — the new CORBA Message Service [20] — that introduces an asynchronous model of communication allowing applications to specify some QoS (Quality of Service) features related with messages. This service interfaces enable bounding the method invocation blocking, by specifying, for example, a maximum latency in the communication.

CORBA was aimed at building applications usually in large and heterogeneous environments and so, it is normally associated with dynamic systems and unpredictable load. However, RT CORBA can be applied to static systems, too. In those systems, with *a priori* knowledge of the computational load and enough computing resources for the worst case needs of this load, timing constraints can obtain an off-line guarantee (*i.e.* the satisfiability of timing constraints can be verified off-line).

Some new technologies, like GPS receivers (for obtaining a global time based on UTC) [4], ATM and POSIX specifications allow to have a predictable computing support for complex and distributed systems using deterministic approaches in real-time processing. Thus, abstractions introduced in RT CORBA 1.0 and RT CORBA 2.0 (next) specifications will allow the building of a variety of real-time programming models — from deterministic to best-effort approaches. The next sections present techniques for adaptive scheduling and the use of RT CORBA abstractions for building adaptive scheduling in open systems.

3. Adaptive scheduling

In overloaded systems, the predictability assumes a new meaning: — the capacity to guarantee during an overload that, tasks will have their timing constraints met following an order of importance [12]. It is opportune to note that the importance of tasks can assume also a dynamic semantic. For example, some tasks may have their importance values dynamically changed due to a recent history of deadline misses [7,11]. Another important property in real-time systems is *configurability* that refers to the capacity of application designers to indicate an order of importance for tasks. The two properties joined — overload predictability and configurability — will guarantee that in an overload situation, the less important tasks will fail before the most important ones [12].

In *Adaptive scheduling* [3,5,7,9,11,13,14,22] the scheduling decisions are taken dynamically to provide a graceful degradation and to cope with uncertainty in system load. Graceful degradation means, in real-time systems, to maintain overload predictability and configurability properties. Adaptive schedulers are used in real-time systems with dynamic and unpredictable load. But, even in deterministic environments where the computational load is known *a priori*, overload occurrences can happen. Not enough computing resources at critical instants or underestimated timing parameters (for example, worst-case execution time) can imply in overloads and, in this case, adaptive schedulers can be also used in such environments. Unlike conventional approaches, some adaptive approaches do not need to know *a priori* the worst-case execution time (WCET) of the tasks. The WCET determination, necessary in conventional approaches, is always an arduous work [6].

3.1 Adaptive Techniques

In real-time applications, usually several tasks are executed periodically. One technique for providing adaptability is to adjust task periods at run time. This technique may be used, for example, by changing the presentation (or sampling) frequency of frames in a video-on-demand application. The adaptation of task periods can be also used in some feedback control systems [9]. However, this technique has a drawback: by changing the period of one task on a distributed system, it may be necessary to change the frequency of related tasks in the same sub-system [7].

The imprecise computation [3,11] is another technique that, by allowing a combination of deterministic guarantee with graceful degradation, is used for handling overload. In this technique each task is decomposed in two portions: a mandatory portion and an optional portion. The mandatory portion of the task must be completed before the deadline of the task for producing an approximate result of an acceptable quality. The optional portion refines the result produced by the mandatory part. During an overload, a “minimum” level system operation can be guaranteed in a deterministic way by executing only the mandatory parts. The time polymorphic invocation [22] is a kind of imprecise computation technique implemented using multiple versions.

In [3] is discussed the concept of cumulative errors produced by the use of the imprecise computation. This kind of error results from the consecutive imprecise execution (only the mandatory portion) of a task**.** A scheduling heuristic is presented which maintain the cumulative error of a periodic task under a certain bound. This aim is reached by guaranteeing that at least one precise execution of a task occurs inside a window of *k* successive task arrivals.

Recently, some works [1,2,7,8] extended the conventional concepts of firm deadlines, allowing periodic tasks to miss deadlines. In those approaches, it is necessary to maintain the number of missed deadlines under a certain upper bound, otherwise, a temporal failure is assumed in the system. In [7] Hamdaoui *et al.* proposed

the concept of (m,k)-firm deadline defining that each periodic task must have at least *m* deadlines met in any window of *k* consecutive task arrivals. The tolerated upper bound of missed deadlines given by *k-m*. A dynamic failure is assumed in the system when this bound is exceeded. DBP (Distance Based Priority Assignment) is the scheduling heuristic defined in this technique. Its aim is to minimize the number of dynamic failures. DBP assigns higher priorities to tasks that are close to exceed the upper bound specified for missed deadlines.

On the next section, it is introduced our adaptive scheduling model, that composes some features of the (m,k)-firm concept with time polymorphic techniques, enabling the development of adaptive applications to execute in highly dynamic environments. This new technique includes one more level of flexibility than the approaches described above.

4. An adaptive programming model using RT-CORBA

In a previous work involving CORBA, we investigated how to apply some adaptive techniques to build distributed real-time applications [14], and how to get global-time services supplied by a CORBA object on large scale systems [4]. The main efforts in our research now is to explore the adequacy of the RT CORBA abstractions to enable non-critical real-time applications to execute on dynamic environments. This section presents the APMC (Adaptive Programming Model for RT CORBA) — an adaptive model based on real-time objects to be used with RT CORBA. This model is founded on an adaptive scheduling technique: the *(p+i,k)-firm deadline*, described initially in [13].

4.1 The *(p+i,k)-firm deadline* technique

The adaptive scheduling proposed in the *(p+i,k)-firm deadline* concept uses a task model allowing precise and imprecise executions of tasks. In any window of *k* consecutive task arrivals, each periodic task must have: (i) at most *k-(p+i)* missed deadlines; and (ii) at least *p* precise executions out of those task arrivals that meet their deadlines. The lower bound to the number of task arrivals that met their deadlines is given by (*p+i*); *i* represents the lower bound to the number of imprecise executions that met their deadlines when *p* precise executions are completed in a window of *k* consecutive invocations.

The (p+i,k)-firm approach is an extension of the (m,k)-firm deadline [7] that includes some properties of imprecise computing described in [3] by Chung *et al.* Considering a task using the (m,k)-firm deadline concept, this task can be interpreted from the point of view of the cumulative error [3]. That is, when a task is missing deadlines, its quality is degraded by accumulating errors so that, it is necessary to complete at least *m* executions in any window of *k* invocations to keep the error rate at acceptable values. The extension of the (m,k)-firm deadline with precise and imprecise executions allows a task to execute in an imprecise way, producing approximate results and having an accumulated error smaller than when a deadline is missed. Besides, if *p* precise executions in a window of *k* consecutive invocations guarantee a certain quality, then the scheduler will have another level of flexibility for treating overloads. An imprecise execution represents a shorter execution time and the scheduler may use that for controlling the number of missed deadlines of the task set.

The concept of dynamic failure described in [7] is related only to the condition (i) presented above. In our approach, this concept also is extended to guarantee the condition (ii): — a dynamic failure is assumed in a task when more than *k-(p+i)* deadlines are missing, *or* less than *p* precise executions occurs in any window of *k* invocations.

It is possible to show that both the concepts (m,k)-firm deadline and of cumulative errors in imprecise computation (Chung) can be represented by using the (p+i,k)-firm deadline (Table 1).

(m,k)-firm	*(p+i,k)-firm*
(1,2)	(1+0,2)
(2,3)	(2+0,3)
⋮	⋮
(m,k)	(m+0,k)

Chung	*(p+i,k)-firm*
1 precise in any 2 executions	(1+1,2)
1 precise in any 3 executions	(1+2,3)
⋮	⋮
1 precise in any k executions	(1+(k-1),k)

Table 1. Mapping (m,k)-firm and Chung models to the (p+i,k)-firm deadline.

Our proposed model can be viewed as a combination and generalization of those models. First, it extends the (m,k)-firm deadline concept, by enabling tasks to execute in an imprecise way, obtaining better results than deadline misses. Second, it enables the imprecise computation model to encompass deadline misses what is natural to happen on dynamic systems.

4.1.1 Scheduling policies in the (p+i,k)-firm approach.

In APMC there is a group of tasks competing for service in a CPU. Each task has a deadline (p+i,k)-firm, and each task invocation (activation) has a deadline value. The aim of a system scheduler is to schedule tasks avoiding dynamic (temporal) failures. Conceptually, FIFO queues are defined for the arrival of task invocations (Figure 2). Each task *j* is associated to an arrival queue that assures that invocations of the same task will be served by the scheduler in their arrival order. Only task invocations at the head of a queue are candidates to be served. These invocations are inserted in the ready queue according to a *priority-assignment policy*. Besides attribution of

priorities, a *precision-acceptance policy* selects which version (precise or imprecise) of the task will be executed. The precision-acceptance policy can also entirely reject a task invocation.

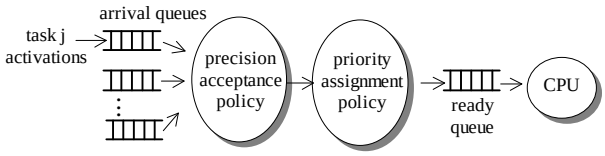

Figure 2. Scheduling approach of the APMC.

In the APMC scheduling approach, the precision-acceptance and the priority-assignment policies work in an integrated way, selecting the invocation in a precise or imprecise version, and assigning its priority[2].

4.1.2 Tasks with (p+i,k)-firm deadline. The scheduling policy in APMC is based on a history of precise, imprecise executions and missed deadlines. An *execution history* of a task, or simply *history*, is a k-tuple that stores the state of the last *k* task invocations. Let the following representation for each invocation state: *P* for precise execution, *I* for imprecise execution, and *X* for missed deadline. To each new state, the history is shifted (from right to left) and the new state added. For example, for a task with deadline (2+0,4)-firm, a history "PPPX" indicates that this task missed deadline in its last invocation (the *X* on the right).

Let the following variables for each task *j* with (p+i,k)-firm deadline:

$p_j(k)$: number of precise executions on the last k invocations;

$i_j(k)$: number of imprecise executions on the last k invocations;

$x_j(k)$: number of missed deadlines on the last k invocations.

where $k = p_j(k) + i_j(k) + x_j(k)$

By definition, a normal task execution (without dynamic failure) occurs when:

$$\text{(i) } x_j(k) \le k - (p+i) \wedge$$
$$\text{(ii) } p_j(k) \ge p$$

The condition (i) limits the number of missed deadlines, and (ii) specifies the minimum acceptable number of precise executions. Both the conditions, (i) and (ii), cannot be guaranteed because it is assumed a non-deterministic environment. However, like the DBP algorithm in [7], the priority-assignment policy will guarantee higher priorities for tasks close to violating condition (i). Besides, the precision-acceptance policy will never release less than *p* precise executions in any window of *k* invocations.

Two important attributes in a (p+i,k)-firm deadline specification are *missed-deadline autonomy* and *imprecise-execution autonomy*. The first one specifies the number of consecutive deadlines that a task can still miss, at a specific time, without the occurrence of a dynamic failure; and the second specifies the number of consecutive imprecise invocations that a task still tolerates. Let us take as example four tasks: τ_1: (4+0,4)-firm, τ_2: (2+2,4)-firm, τ_3: (0+2,4)-firm and τ_4: (2+0,4)-firm. Considering that in all these tasks, the last *k* invocation deadlines were met in the precise way, then:

- τ_1 and τ_2 do not have autonomy to miss deadline, but τ_2 has autonomy to execute the next 2 invocations in the imprecise way;
- τ_2 and τ_3 have autonomy to execute the next 2 invocations in the imprecise way, but τ_2 does not have deadline miss autonomy;
- τ_3 and τ_4 have autonomy to miss the next 2 deadlines, but τ_4 does not have autonomy to execute in the imprecise way.

While the priority-assignment policy is driven by the missed-deadline autonomy, the precision-acceptance policy is driven by the imprecise-execution autonomy concept. In other words: — while the priority-assignment policy assigns priorities trying to avoid missed deadlines in tasks that have less freedom to miss deadlines; the precision-acceptance policy will select task invocations for imprecise executions, for tasks that have more freedom to execute their imprecise versions.

4.1.3 Scheduling heuristic.

The priority-assignment policy

Let $d_j(k)$ represent the value of the missed-deadline autonomy of a task *j*. The interpretation of this value is presented below:

$d_j(k)$	Interpretation
0	failure
1	if miss next deadline then fail
⋮	
n	if miss next n deadlines then fail

It is possible to obtain $d_j(k)$ using a function similar to another one introduced in DBP heuristic [7]. Let $pm_j(n,s)$ denote for a task *j*, the position (from the right to the left) of the n^{th} deadline met in a precise or imprecise execution in the history *s*. If there are less than *n* met deadlines in *s*, then this function returns *k+1*. For example, $pm_j(1,\text{“XPP"}) = 1$, $pm_j(1,\text{“XPX"}) = 2$, $pm_j(2,\text{“IXP"}) = 3$, $pm_j(2,\text{“XXP"}) = 4$.

Using this function, the value of missed-deadline

[2] Actually, the model presented in Figure 2 is generic. The priority-assignment policy could be used, for example, to implement the EDF scheduling [10], where the priorities are assigned according to the invocation deadlines. The precision-acceptance police could be used in the Red Tasks Only strategy [8] that discards invocations whenever possible, once the discards does not exceed a certain value (a skip factor).

autonomy can be calculated by:

$$d_j(k) := k - pm_j(p+i,s) + 1$$

For example, a task declared with a (2+0,4)-firm deadline and a history "PPXX" would have: $d_j(k) = 4 - pm_j(2, \text{“}PPXX\text{"}) + 1 = 4 - 4 + 1 = 1$, indicating that it will fail if this task misses the next deadline. However, if the history of this task was "XPXP": $d_j(k) = 4 - pm_j(2, \text{“}XPXP\text{"}) + 1 = 4 - 3 + 1 = 2$, indicating that the task could still miss the next deadline without failing.

The $d_j(k)$ calculation can be applied directly, by mapping the calculated value to the operating system native priorities. For example, supposing the highest priority is 0 (like occurs in some operating systems), for any activation of task *j* is enough to do:

$$priority_j := d_j(k)$$

The precision-acceptance policy

Let $v_j(k)$ represent the imprecise-execution autonomy of task *j*. The $v_j(k)$ interpretation is presented below:

$v_j(k)$	Interpretation
0	failure
1	if execute one invocation on imprecise way then fail
⋮	
n	if execute n invocations on imprecise way then fail

The calculation of $v_j(k)$ uses a $pp_j(n,s)$ function, that denotes, for a task *j*, the position of the n^{th} deadline met with a precise execution in the history *s*. If there are less than *n* precise executions in *s*, then this function returns *k+1*. For example, $pp_j(1,\text{“}XIP\text{"}) = 1$, $pp_j(1,\text{”}XPI\text{"}) = 2$, $pp_j(2,\text{“}PXP\text{"}) = 3$, $pp_j(2,\text{“}XXP\text{"}) = 4$.

Using this function, the value of the imprecise execution autonomy can be calculated as:

$$v_j(k) := k - pp_j(p,s) + 1$$

For example, a task declared with a deadline (2+2,4)-firm and with a history "PPII" would have: $v_j(k) = 4 - pp_j(2, \text{“}PPII\text{"}) + 1 = 4 - 4 + 1 = 1$, indicating that the next execution cannot be in the imprecise way. However, if the history of this task was "IPIP": $v_j(k) = 4 - pp_j(2, \text{“}IPIP\text{"}) + 1 = 4 - 3 + 1 = 2$, it indicates that the task could still have an imprecise execution.

The main function of the precision-acceptance policy is to decide, for each task, a precise or imprecise version for the next invocation. Therefore, for each task of the system, there is a variable (nx_j) which stores this information.

As the scheduling approach is not overload-aware [12] (*i.e.*, it cannot foresee deadline misses), the precision-acceptance policy is driven by overload occurrences. In any indication of a missed deadline in the task set of the system, a task invocation is selected for executing its imprecise version (reducing its quality), based on its imprecise-execution autonomy,.

The precision-acceptance policy selects the task invocation on the top of arrival queues which has the highest imprecise-execution autonomy ($v_j(k)$) and whose nx_j specifies precise version (*i.e.*, $\max(v_j(k)) \wedge nx_j = \text{“precise”}$). The imprecise version of this task is selected to execute ($nx_j := \text{“imprecise”}$) on the next and on the following invocations, until the critical situation indicated by the imprecise-execution autonomy has reached ($v_j(k)=1$; that is, the task cannot run its imprecise version in the next activation otherwise a dynamic failure will happen). In this last case, that task is assigned to execute its precise version again ($nx_j := \text{“precise”}$).

4.2 Evaluation of the scheduling approach

The (p+i,k)-firm deadline model is being evaluated through simulations, that try to establish the behavior of its scheduling approach in overload situations. Two important metrics in the simulations, are the reduction in the probability of dynamic failures; and the gradual reduction in the quality of each task. It is desirable that, in overload situations, the deadline misses to be distributed among all the tasks aiming to minimize the amount of dynamic failures. Furthermore, a task should not be penalized much more (in an unequal way) than the others tasks. Thus, solutions such as “to discard one task completely” (or discarding almost all its invocations), that can reduce a lot the number of dynamic failures in overload situations, are considered bad solutions.

The simulator, written in C language, allows to configure the number and the time constraints of the system tasks. Each task receives invocations, whose arrival times are exponentially distributed. In the experiments, whose results are presented in the Figure 3, five tasks with the same time constraints (*w.r.t.*, computation times and relative deadlines) are used.

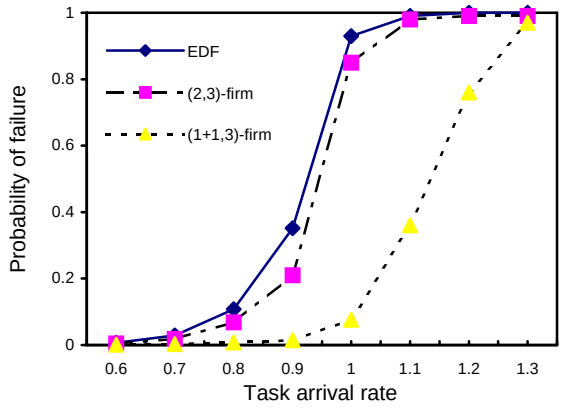

Figure 3. Comparing scheduling approaches.

The relative deadline of each task is specified as being five times its computation time. In the (p+i,k)-firm approach, the computation times of the imprecise versions

are calculated as being 20% of the computation times of the precise versions. The experiments consider that all task invocations are served completely, even after missing their deadlines.

The (p+i,k)-firm scheduling approach is compared to the DBP in the (m,k)-firm, using, respectively, tasks with (1+1,3)-firm and (2,3)-firm deadlines. Therefore, in both approaches, one deadline out of any three consecutive invocations may be missed. Task invocations with the same priority are ordered in both approaches, by using absolute deadline values assigned on their arrivals. In the Figure 3, the two approaches are also confronted with traditional EDF (Earliest Deadline First) [10].

The outcome shows that the (p+i,k)-firm scheduling approach obtains a substantial reduction in the amount of dynamic failures. The main reason is that in overload situations, the approach in (p+i,k)-firm can transitorily execute some task invocations in its imprecise versions, reducing the average load, and consequently the amount of deadline misses. The possibility of imprecise executions is more a degree of flexibility that the (p+i,k)-firm model presents, and that is decisive for reducing the number of dynamic failures.

4.3 Programming the APMC using RT CORBA

Figure 4 presents a scenario of the APMC model using CORBA abstractions. Our scheduling approach is implemented by a CORBA service object presented in all nodes — the *Adaptive Service*. This service is responsible for enforcing the APMC heuristics of the priority-assignment and precision-acceptance policies by scheduling server methods invocations. A method is defined either with two code versions (precise and imprecise versions), by having a (p+i,k)-firm deadline; or with just one version, by having a (p+0,k)-firm deadline (that is equal to a (m,k)-firm deadline [7]). Each method, with a (p+i,k)-firm deadline, must be subscribed in the Adaptive Service, before its first invocation. During a method invocation with a (p+i,k)-firm deadline, the Adaptive Service selects one version dynamically (if there are two versions) by using the precision-acceptance policy. For the remote client that makes the request the execution of the invoked method is transparent.

Overload occurrences are treated using the adaptive techniques solely in the server node. The client must assign an absolute deadline to the method invocation before sending the request. This kind of invoking request is called a *real-time invocation*.

In order to make the deadlines values meaningful across the nodes, all the clocks in the system must be synchronized within a bounded skew of each other. The APMC rely on a *Global Time Service* [4]. This service extends the CORBA Time Service, providing clock synchronization and a global time notion. The Global Time Service can be implemented in a large and dispersed system, using GPS receiver technologies, that enable a distributed application to obtain an accurate global time value based on UTC time.

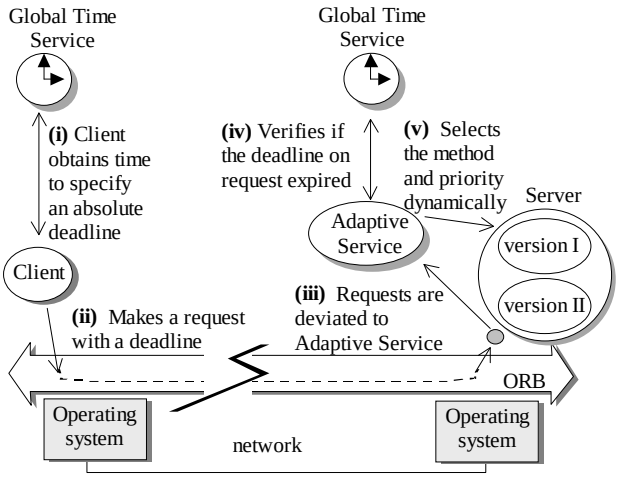

Figure 4. The real-time invocation in APMC.

In our first prototype implementation, the real-time invocation model considers only one-way requests, *i.e.*, clients do not block waiting returns of server. Also, each invocation is executed completely, even after their deadline. There are no discards or aborts of request executions, even knowing that a firm deadline conceptually does not give any benefit when missed. Therefore, the step (iv) in Figure 4 is not implemented, but we are extending the initial prototype for that. These extensions will also consider two-way requests, and an exception will be raised both in client and server sides, to treat discarded invocation or dynamic failure signaling.

ORB interceptors [15,19], pool of threads and request queues [18] are useful mechanisms to implement the APMC model. Server interceptors are used to hook the APMC services into the ORB, *i.e.*, requests are turned to the Adaptive Service in a transparent way using interceptors. A pool of threads is created on server to execute client requests. The pre-allocation of a pool of threads defined in the binding increases the server performance. Request queues are associated to the pool of threads and used to implement the scheduling model presented in the Figure 2.

New RT ORB mechanisms also make possible the Adaptive Service to assign priorities to invocations in a heterogeneous environment. That is the case of CORBA priority that is an universal and platform-independent priority scheme. In the *client-priority propagation* model (mentioned in the Section 2), each invocation request method can conveys a CORBA priority (into an IOP context service [15]) that is used on the server side by the threads that dispatches the invocations. The Adaptive Service inserts dynamically a new calculated priority

value into the messages, following the priority-assigned policy from the APMC.

The APMC model prototype extends the RT CORBA 1.0, enabling the deadline specified by the client to be also stored into an IOP context service[3]. This strategy allows some transparency, because only the client and the Adaptive Service must be aware of this deadline value.

For implementing the APMC model, we have used the TAO [21] — an ORB compliant with CORBA 2.2 specification [15] — briefly described in the Section 2. TAO has some real-time facilities, and its open source-code, available from Internet, enabled us to implement some features, and make some patches necessaries to implement our Adaptive Service (such as to implement requests deviations, because the TAO version used does not implement interceptors).

5. Related works

The DHDA project [23] was one of the first works that established some real-time requirements to CORBA. This project uses a best-effort programming approach to support end-to-end time constraints in a dynamic environment, with objects being added and removed, and with changing timing constraints. The goal is the development of real-time extensions to CORBA, following a generic philosophy of adherence to open systems (*e.g.*, using COTS components, CORBA, POSIX operating systems, and ATM networks).

TAO [21] is a real-time ORB that in deterministic environments enable the execution of hard real-time applications (such as avionics and embedded systems). In order to provide an end-to-end QoS, TAO integrates the network interfaces, OS I/O subsystem, ORB, and middleware services. The CORBA IDL is extended to allow the description of timing constraints (*e.g.*, WCET) associated with each method. A RT Scheduling Service is implemented using the Rate Monotonic policy [10].

The complementary DHDA and TAO researches on dynamic and static environments influenced the RT CORBA specification. The initial goal was to specify broad and flexible RT CORBA abstractions.

The QuO project [24] establishes a programming model to allow the specification of routines for treatment of changes in the execution environment. The aim was to implement adaptive applications that can execute on highly dynamic environments such as the Internet. The QuO model differs from the APMC, since the QuO extends CORBA IDL specifications [15], by creating several languages to describe QoS object interfaces.

In [6], CORBA interceptors are used to monitor requests to CORBA objects, and the result of that monitoring is used in an adaptive scheduling approach called Task-pair Scheduling. This scheduling is a kind of implementation of the imprecise computation using two versions. In this approach, the monitoring results may lead to the decision of canceling the processing of the main version and activating the other version. This last one has a smaller code and may represent an exception treatment. The imprecise computation technique implemented on Task-pair Model, differs from that of the APMC, since in the Task-pair scheduling, it is necessary to guarantee CPU time for the imprecise version.

In the RTRD [5] meta-object protocols are used on a CORBA middleware, allowing applications to control its own temporal behavior. That flexible framework provides facilities for building different types of soft timing constraints specified by the applications (*e.g.*, multimedia). The scheduling approach allows only best-effort policies.

All the projects and experiences above, extending or using CORBA for real-time applications [5,6,21,23,24], increased the interest in achieving real-time features in CORBA, leading to the development of the OMG RT-CORBA specifications. The APMC model is one of initial efforts involving new released RT-CORBA standards [18-20].

6. Conclusions

Usually, real-time systems are implemented using proprietary technologies and platforms that increase their development and maintenance costs. Recently, CORBA has been adopted in many real-time applications. The aim is to extend properties of portability, interoperability, flexibility and costs reduction to the contemporary real-time systems.

With the new abstractions offered by the real-time ORB that is being specified by OMG, several kinds of models for real-time application programming can be developed. This paper presented the APMC — an adaptive object oriented model for real-time programming in CORBA. The APMC allows the programming and an adaptive control of real-time applications based on the (p+i,k)-firm deadline approach. This approach can be viewed as a combination and generalization of the imprecise computation and the (m,k)-firm deadline. First, it permits a task with a (m,k)-firm deadline to execute in an imprecise way, obtaining better results than a deadline miss. Second, it enables the imprecise computation model to encompass deadline misses what is natural to happen on dynamic systems.

Now we are developing a testbed to evaluate the model and its attributes for treating overload occurrences by decreasing the probability of dynamic failures. In this

[3] We hope that the next RT CORBA 2.0, which will be issued in 1999, will specify a similar approach to carry deadline values in client invocation request messages.

way, simulations are been done to verify the performance of the (p+i,k)-firm deadline. These simulations also compare it to the other adaptive techniques cited in this paper.

Furthermore, our adaptive approach, founded on RT CORBA concepts, is being evaluated through programming experiments involving transmission of image frames. All experiments use JPEG encoding to compress image frames. JPEG standard offers excellent compression ratios in continuous and simple images, by eliminating spatial redundancies. This standard can be also used for motion video, offering good results. However, the main motive to use JPEG in our experiments is the fact that their decoders can trade off speed (CPU usage) against image quality, by using fast but inaccurate approximations to the required calculations. In all experiments a client sends previous stored JPEG frames to a remote server that implements the decoder in two versions — one version offers better quality but consumes more CPU time, another version offers worse quality but needs less CPU time. In each experiment, we are modeling the system using distinct deadline values in the client requests, and distinct (p+i,k)-firm constraints on the server implementation. We are also comparing our approach with the (m,k)-firm [7] in these experiments.

Acknowledgements

This work was supported in part by CNPq and CAPES. We would like to thank Clovis Petry for his valuable help in the APMC implementation.

7. References

[1] G. Bernat, A. Burns, “Combining (n m)-Hard Deadlines and Dual Priority Scheduling”, *In Proc. of the 18th IEEE RTSS*, Dec. 1997.

[2] M. Caccamo, G. Butazzo, “Exploiting Skips in Periodic Tasks for Enhancing Aperiodic Responsiveness”, *In Proc. of the 18th IEEE RTSS*, Dec. 1997.

[3] J. -Y. Chung, J. W. S. Liu, K. -J. Lin, “Scheduling Periodic Jobs that Allow Imprecise Results”, *IEEE Transactions on Computer*, 39(9), 1990, pp.1156-1174.

[4] J. Fraga, J-M. Farines, C. Montez, “Um Serviço de Tempo Global para Sistemas Distribuídos de Larga Escala”, SBRC'98, Rio de Janeiro, Brazil, May 1998, in Portuguese.

[5] O. Furtado, F. Siqueira, J. Fraga, J-M. Farines, “A Reflective Model for Real-Time Applications in Open Distributed Systems”, *Proc. IFIP/IFAC WRTP´96*, Brazil, Nov. 1996.

[6] M. Gergeleit, E. Nett, M. Mock, “Supporting Adaptive Real-Time Behavior in CORBA”, *Proc. of 1st IEEE WMDRTSS*, San Francisco, CA, Dec. 1997.

[7] M. Hamdaoui, P. Ramanathan, “A Dynamic Priority Assignment Technique for Streams with (m,k)-Firm Deadlines”, *In IEEE Trans. on Computer*, Apr. 1995.

[8] G. Koren, D. Shasha, “Skip-Over: Algorithms and Complexity for Overloaded Systems that Allow Skips”, *In Proc. of the 16th IEEE RTSS*, Pisa, Italy, Dec. 1995.

[9] T. Kuo, A. K. Mok, “Incremental Reconfiguration and Load Adjustment in Adaptive Real-Time Systems”, *IEEE Trans. on Computers*, Vol. 46, No. 12, Dec. 1997.

[10] C. L. Liu, J. W. Layland, “Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment”, Journal of the ACM, 20(10), Jan. 1973.

[11] J. W. S. Liu, W. Shih, K. -J. Lin, R. Bettati, J. -Y. Chung, “Imprecise Computations”, *Proceedings of IEEE*, Vol. 82, No. 1, Jan. 1994, pp. 83-94.

[12] M. J. Marucheck, J. K. Strosnider, “Some Insight into the Fault Recovery Properties of Priority-Driven Schedulers”, *The Real-Time Systems Journal*, Oct. 1995.

[13] C. Montez, J. Fraga, R. S. Oliveira, J-M. Farines, “A Abordagem de Escalonamento (p+i,k)-firm”, submitted to CLEI’99, 1999, in Portuguese.

[14] C. Montez, R. S. Oliveira, J. Fraga, “An Adaptive Model for Programming Distributed Real-Time Applications in CORBA”, *Proc. of XVIII International Conference of Chilean Computer Science Society, SCCC’98*, Antofagasta, Chile, Oct. 1998.

[15] OMG, “The Common Object Request Broker: Architecture and Specification - Revision 2.2”, Object Management Group (OMG), Feb. 1998.

[16] OMG, Realtime Platform SIG, “Realtime Technologies - RFI - Request for Information”, Object Management Group (OMG), Document realtime/96-08-02 96, Aug. 1996.

[17] OMG, “Realtime CORBA 1.0 RFP - Request for Proposal”, Object Management Group (OMG), Document orbos/97-09-31, Sep. 1997.

[18] OMG, “Realtime CORBA - Joint Revised Submission”, Object Management Group (OMG), Document orbos/98-10-05, Oct. 1998.

[19] OMG, “Portable Interceptor RFP - Request for Proposal”, Object Management Group (OMG), Document orbos/98-09-11, Sep. 1998.

[20] OMG, “CORBA Messaging - Joint Revised Submission”, Object Management Group (OMG), Document orbos/98-05-05, May 1998.

[21] D. C. Schmidt *et al.*, “TAO: A High Performance Endsystem Architecture for Real-Time CORBA”, *IEEE Communications Magazine*, 14(2), Feb. 1997.

[22] K. Takashio, M. Tokoro, “Time Polymorphic Invocation: A Real-Time Communication Model for Distributed Systems”, *In Proc. of the IEEE WPDRTS’93*, 1993.

[23] V. F. Wolfe, *et al.*, “Expressing and Enforcing Timing Constraints in a Dynamic Real-Time CORBA System”, University of Rhode Island, Department of Computer Science and Statistics, Technical report TR97-252, Jun.1997.

[24] J. A. Zinky, D. E. Bakken, R. D. Schantz, “Architectural Support for Quality of Service for CORBA Objects”, *Theory and Practice of Object System*, Vol. 3(1). 1997.