# Escalonamento Adaptativo Usando 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 - Brasil

[2]Inst. de Informática - Univ. Fed. do Rio Grande do Sul
Caixa Postal 15064 - 91501-970 - Porto Alegre - RS - Brasil

**Resumo**

*CORBA é uma infra-estrutura de middleware emergente com padronização aberta que está recebendo uma grande aceitação por facilitar a programação de objetos distribuídos. CORBA está sendo estendido através da especificação de interfaces e abstrações necessárias para suportar aplicações com restrições temporais. Essas novas abstrações habilitarão uma variedade de modelos de programação para aplicações de tempo real. Este artigo apresenta um modelo de programação adaptativo para aplicações distribuídas de tempo real usando conceitos CORBA. O modelo combina a técnica de invocação envolvendo polimorfismo temporal com a garantia (m,k)-firm.*

**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.*

***Palavras-chave**: Sistemas de tempo real, CORBA, Escalonamento adaptativo.*

## 1. Introdução

CORBA (*Common Object Request Broker Architecture*) é um *middleware* cujo o objetivo consiste em minimizar as dificuldades existentes na programação distribuída para ambientes heterogêneos. Suas especificações abertas, padronizadas pelo consórcio OMG (*Object Management Group*), estão recebendo uma crescente aceitação. A arquitetura de referência especificada pela OMG [17] é composta por um ORB (*Object Request Broker*), especificações de objetos de serviços e de facilidades. O ORB é um tipo de barramento de *software*, que manuseia as comunicações entre objetos distribuídos de uma forma transparente. Os objetos de serviços e facilidades, envolvendo interfaces padronizadas, suportam algumas funções básicas usadas pelos objetos de aplicação.

Sistemas de tempo real usualmente são implementados usando tecnologias e plataformas proprietárias que aumentam seus custos de desenvolvimento e manutenção. Recentemente, muitos sistemas distribuídos de tempo real estão adotando o paradigma de orientação a objetos e as especificações CORBA. A meta é estender as propriedades de portabilidade, interoperabilidade, flexibilidade e redução de custos aos sistemas de tempo real contemporâneos.

O uso de CORBA, assim como outras tecnologias abertas, em sistemas de tempo real distribuídos é uma área de pesquisa recente. Essa tendência abrange um amplo conjunto de aplicações, envolvendo desde sistemas que interagem com ambientes deterministas, tais como aplicações embarcadas (*embedded*) e de controle de processos; até sistemas de larga escala,

caracterizados por uma carga computacional dinâmica, tais como aplicações multimídia distribuídas na Internet. Muitos dos domínios dessas aplicações requerem garantias (*off-line* ou *on-line*) de tempo real das redes, sistemas operacionais e componentes de *middleware*, no sentido de atender suas restrições temporais. Entretanto, os padrões CORBA atuais, bem como outras tecnologias abertas, mostram-se inadequados para o suporte a requisitos de tempo real.

CORBA foi criado para aplicações de propósito geral, que desejam transparências na distribuição e na forma que os recursos são geridos. Por outro lado, essas transparências (*p. ex.*, na localização ou na migração de *software)*, usualmente, não são aceitáveis em aplicações de tempo real. Além disso, o protocolo de interoperabilidade CORBA (IIOP - *Internet Inter-Orb Protocol*) foi projetado para enviar mensagens através de conexões TCP/IP. O protocolo TCP sofre da falta de previsibilidade, o que torna o IIOP um protocolo inadequado para diversas aplicações de tempo real. Também, a versão atual de CORBA fornece apenas dois modos de invocação: síncrono e síncrono deferido; e esse último é disponível apenas através do uso da interface de invocação dinâmica. A falta de um modelo realmente assíncrono é uma restrição que programadores de tempo real atualmente precisam lidar. Acrescenta-se também, que CORBA não possui algumas facilidades úteis para a programação tempo real, tal como invocações com *timeout*.

Em 1995, um grupo de interesse especial foi formado dentro da OMG com objetivo de estender o padrão CORBA destinando-o para aplicações de tempo real. A especificação final resultante desse esforço — o RT CORBA — apresenta diversas abstrações (interfaces e mecanismos) que habilitam a execução de aplicações de tempo real, permitindo a definição de uma variedade de modelos de programação de tempo real.

O estudo descrito neste artigo é parte de uma pesquisa que está sendo conduzida com o objetivo de desenvolver um modelo de programação distribuída para aplicações de tempo real usando conceitos do RT CORBA. O modelo proposto tem características adaptativas, pois algumas decisões de escalonamento são baseadas nas condições observadas do sistema através de monitoramentos. Em ambientes que não oferecem previsibilidade, a característica adaptativa do modelo pode ser vista como uma forma de responder às variações dinâmicas do ambiente, através do fornecimento de vários níveis de qualidade nos serviços oferecidos.

O modelo proposto combina duas estratégias de escalonamento adaptativo — a invocação com polimorfismo temporal [22] e a garantia (m,k)-firm [7]. Projetistas podem construir seus sistemas usando apenas uma ou ambas as técnicas. A abordagem de escalonamento é construída através do uso de serviços e conceitos recentemente incorporados nas especificações CORBA para tempo real [20].

Este artigo é dividido em 6 seções. A seção 2 descreve algumas características incorporadas nas especificações de RT CORBA. A seção 3 apresenta uma discussão sobre técnicas adaptativas. A seção 4 introduz nossa abordagem adaptativa e descreve nossa experiência em implementar o modelo através do uso de alguns conceitos de RT CORBA. A seção 5 lista trabalhos relacionados. Finalmente, a seção 6 apresenta algumas observações finais.

## 2. Real-time CORBA

A falta de suporte de *middleware* para aplicações de tempo real levou o consórcio OMG a criar em 1995 um grupo de trabalho com objetivo de estender os padrões CORBA. Um documento RFI (*Request for Information*) [15] foi lançado especificando requisitos que deveriam ser mantidos por um ORB que suporta aplicações com restrições temporais. O

documento definiu um conjunto de requisitos para o ambiente operacional, para o ORB e serviços. Os requisitos de ambiente operacional podem ser resumidos em: *multithreading*, filas ordenadas por prioridade, sincronização de relógios e latência máxima de comunicação delimitada. Já os requisitos de ORB e serviços envolvem certas facilidades para expressar e implementar restrições temporais, a noção de prioridade global, invocações com polimorfismo temporal, etc.

Em setembro de 1997, um documento RFP (*Request for Proposal*) [16] foi publicado solicitando propostas de padronização de um CORBA para aplicações de tempo real (RT CORBA). Esse primeiro documento RFP, o Real-Time CORBA 1.0 RFP, definiu alguns requisitos para escalonamento baseado em prioridades[1]. Esse documento solicitou submissões cobrindo alguns aspectos obrigatórios, tais como, a definição de "entidades escalonáveis", mecanismos para expressar e controlar prioridades e para delimitar tempos de bloqueio na invocação de métodos e na inversão de prioridades.

Em outubro de 1998, cinco propostas apresentadas previamente foram reunidas em uma única proposta [20] — o RT CORBA 1.0. Esse documento enfatiza a definição de mecanismos de ORB, que permitem aos projetistas selecionarem as políticas para escalonamento de tempo real. Não existem suposições prévias sobre o ambiente operacional subjacente mas o documento reconhece que um sistema operacional em conformidade com POSIX é adequado para a obtenção de previsibilidade. O RT CORBA 1.0 descreve *threads* como as entidades escalonáveis. Interfaces são especificadas para gerir threads o que permite uma grande portabilidade às aplicações. *Threads* podem ser definidas e controladas de forma uniforme, independentemente se a plataforma de execução possui *threads* Solaris, NT, ou POSIX.1c.

*Pool de threads* e *filas de requisição* são recursos que aplicações podem manipular diretamente. Um *pool de threads* pode ser criado para processar requisições de métodos recebidas por um servidor. Uma fila de requisição pode ser criada e associada com o *pool de threads* (Figura 1).

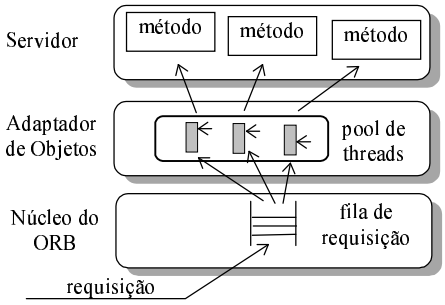


**Figura 1. Um exemplo de uso de novos mecanismos do RT CORBA.**

A *prioridade CORBA* é um tipo de prioridade global que pode atravessar diferentes ORBs, dentro de mensagens de invocação. Dependendo da abordagem de escalonamento, essa prioridade pode ser usada pelos servidores para ordenar as requisições recebidas. Portanto, o documento RFP introduz dois modos de utilização: os modelos *client-priority propagation* e *server-set priority*. Usando ambas as estratégias, a prioridade CORBA da invocação deve ser mapeada para as prioridades do sistema operacional antes da execução do método invocado.

Durante um *binding explícito*, vários passos devem ser tomados para interconectar objetos,

[1] Um novo RFP — o Real-Time CORBA 2.0 RFP — é esperado em breve para especificar escalonamento dinâmico, e facilidades para escalonamento baseado em deadlines.

tais como localizar objetos destino e iniciar estruturas de dados que suportem comunicação entre objetos. No lado do cliente, o *binding* pode ser usado para garantir que seus requisitos temporais sejam respeitados (*p. ex.*, através da seleção de um protocolo de transporte apropriado e de um valor de *timeout* para detecção de falhas). No lado do servidor, o *binding* permite a alocação dos recursos necessários na execução das requisições, tais como *threads* e filas.

O principal mecanismo proposto nas especificações CORBA, que introduz flexibilidade na programação da aplicação é o *interceptador*. Interceptadores são objetos de aplicação cujas operações podem ser chamadas pelo ORB em diversos momentos de uma invocação. Esses mecanismos são geralmente usados para implementar as políticas que regulam o comportamento das aplicações. Inicialmente, eles foram especificados no Serviço de Segurança CORBA, e posteriormente padronizados pela OMG no CORBA 2.2 [17]. A especificação RT CORBA 1.0 não trata sobre interceptadores, porque já existe um documento RFP na OMG [19] propondo extensões para interfaces de interceptadores.

A especificação RT CORBA também é baseada em um outro documento que especifica o novo Serviço de Mensagens CORBA [18]. Esse serviço introduz um modelo assíncrono de comunicação permitindo às aplicações especificarem requisitos de QoS (*Quality of Service*) relacionados com mensagens. O uso desse serviço, possibilita a delimitação de tempos máximos de bloqueios de invocação de métodos, através da especificação, por exemplo, de uma latência máxima na comunicação.

As especificações CORBA foram introduzidas com objetivo de facilitar a construção de aplicações distribuídas geralmente em ambientes de larga escala e heterogêneos. Assim, o seu uso é geralmente associado a sistemas com carga dinâmica e imprevisível. Entretanto, CORBA também pode ser aplicado em sistemas estáticos. Nesses sistemas, com um conhecimento *a priori* da carga e dos recursos computacionais necessários para os piores casos de ocorrência dessa carga, as restrições temporais impostas à aplicação podem obter garantias *off-line*. Algumas novas tecnologias, tais como receptores GPS (para obter um tempo global baseado no UTC) [4], redes ATM e especificações POSIX, permitem a obtenção de um suporte de computação previsível, necessário ao processamento de tempo real usando abordagens deterministas em sistemas distribuídos e complexos. Assim, as abstrações introduzidas nas especificações RT CORBA 1.0 (e futuramente em RT CORBA 2.0) permitirão a construção de uma variedade de modelos de programação de tempo real fundamentados em diferentes abordagens de escalonamento — desde deterministas até as *best-effort*. Nas próximas seções, são apresentadas técnicas para escalonamento adaptativo e o uso de abstrações RT CORBA para o desenvolvimento de modelos adaptativos em sistemas abertos.

## 3. Escalonamento adaptativo

Em sistemas onde ocorrem sobrecargas, a previsibilidade assume um novo significado: — a *previsibilidade na sobrecarga* é a capacidade de garantir que durante uma sobrecarga, as tarefas terão suas restrições temporais atendidas seguindo uma ordem de importância [12]. É oportuno observar que o grau de importância de cada tarefa pode assumir também uma semântica dinâmica. Por exemplo, algumas tarefas podem ter seus valores de importância mudados dinamicamente devido a um histórico recente de perdas de deadline [7,10]. Uma outra propriedade importante em sistemas de tempo real é a *configurabilidade* que se refere à capacidade dos projetistas de aplicação indicarem uma ordem de importância no conjunto das tarefas. As duas propriedades combinadas — *previsibilidade na sobrecarga* e

*configurabilidade* — garantirão que em uma situação de sobrecarga, as tarefas menos importantes falharão antes das mais importantes [12].

*Escalonamentos adaptativos* [3,6,7,9,10,14,22,24] são aqueles onde as decisões de escalonamento são tomadas dinamicamente, para lidar com a incerteza na carga do sistema, objetivando a obtenção de uma *degradação suave*. Em sistemas de tempo real, degradação suave significa a manutenção das propriedades de previsibilidade na sobrecarga e configurabilidade. *Escalonadores adaptativos* são usados em sistemas de tempo real quando existe carga dinâmica e imprevisível. Entretanto, mesmo em ambientes deterministas, onde a carga computacional é conhecida *a priori*, é possível a ocorrência de sobrecargas. Recursos insuficientes em instantes críticos ou mesmo parâmetros temporais subestimados (por exemplo, os piores casos de tempo de execução) podem implicar em sobrecargas e, nesse caso, escalonadores adaptativos também podem ser usados nesses ambientes. De forma diversa às abordagens convencionais, algumas abordagens adaptativas não necessitam o conhecimento *a priori* dos piores casos de tempo de execução (*WCET – Worst Case Execution Time*) das tarefas. A determinação de *WCET* das tarefas, necessária em abordagens convencionais, sempre é um trabalho árduo [6].

### 3.1 Técnicas adaptativas

Em aplicações de tempo real, usualmente diversas tarefas são executadas periodicamente. Uma técnica para fornecer adaptabilidade é o ajuste dos períodos das tarefas em tempo de execução. Essa técnica pode ser usada, por exemplo, através da mudança da freqüência de apresentação de quadros em uma aplicação de vídeo sob demanda. A adaptação dos períodos das tarefas também pode ser usada em alguns sistemas de controle realimentados [9].

A computação imprecisa [10] — uma outra técnica que permite a combinação de garantia determinista com degradação suave — é usada para o tratamento de sobrecarga. Nessa técnica, cada tarefa é decomposta em duas partes: uma parte obrigatória e uma parte opcional. A parte obrigatória de uma tarefa deve ser completada antes do deadline da tarefa para produzir um resultado aproximado com uma qualidade aceitável. A parte opcional refina o resultado produzido pela parte obrigatória. Durante uma sobrecarga, um nível “mínimo” de operação do sistema pode ser garantido de forma determinista, através da execução apenas das partes obrigatórias. A invocação com polimorfismo temporal é uma técnica de computação imprecisa, implementada usando múltiplas versões e descrita em [22].

Em [3], Chung *et al.* discutem o conceito de erros cumulativos, produzidos pelo uso da computação imprecisa. Esse tipo de erro é resultado de execuções *imprecisas* consecutivas (somente a parte obrigatória) de uma tarefa**.** Uma heurística de escalonamento é apresentada que mantém o erro cumulativo de uma tarefa periódica abaixo de um certo valor limite. Esse objetivo é alcançado através da garantia que pelo menos uma execução *precisa* (parte obrigatória e parte opcional) da tarefa ocorra a cada janela de *k* ativações sucessivas da tarefa.

Recentemente, alguns trabalhos [1,2,7,8] estenderam o conceito convencional de deadline *firm*, permitindo que tarefas periódicas percam deadlines sem falhar. Nesses trabalhos, é obrigatório para cada tarefa periódica do sistema, que se mantenha o número de deadlines perdidos abaixo de um determinado valor limite. De outra forma, uma falha temporal é assumida no sistema. Em [7], Hamdaoui *et al*. propuseram o conceito do deadline (m,k)-firm, definindo que uma tarefa periódica deve ter pelo menos *m* deadlines atendidos em cada janela de *k* ativações. O limite superior tolerado de perdas de deadlines é dado por *k-m.* Uma falha dinâmica é assumida no sistema quando esse limite é excedido. O DBP (*Distance Based*

*Priority Assignment*) é a heurística de escalonamento usada com essa técnica no sentido de minimizar o número de falhas dinâmicas. Essa heurística de escalonamento atribui as mais altas prioridades para as tarefas que estão próximas de exceder o limite superior especificado para as perdas de deadlines.

Na próxima seção é introduzido o nosso modelo de escalonamento adaptativo, que compõe algumas das características da garantia (m,k)-firm com a técnica de polimorfismo temporal, possibilitando o desenvolvimento de aplicações adaptativas, que visam executar em ambientes altamente dinâmicos. Essa nova técnica insere um nível a mais de flexibilidade do que as abordagens descritas acima.

## 4. Um modelo de programação adaptativo usando abstrações RT-CORBA

Em alguns trabalhos prévios envolvendo CORBA, nós investigamos como aplicar algumas técnicas adaptativas em aplicações de tempo real distribuídas [14], e como obter serviços de tempo global fornecidos por objetos CORBA em sistemas de larga escala [4]. Atualmente, o principal esforço em nossas pesquisas é explorar a adequação das abstrações RT CORBA em aplicações de tempo real envolvendo ambientes dinâmicos. Essa seção apresenta o APMC (*Adaptive Programming Model for RT CORBA*) — um modelo adaptativo baseado em objetos de tempo real para serem usados com RT CORBA. Esse modelo é fundamentado em uma técnica de escalonamento adaptativo: o *deadline (p+i,k)-firm*, descrito inicialmente em [13].

### 4.1 A técnica do *deadline (p+i,k)-firm*

O escalonamento adaptativo proposto no conceito de deadline *(p+i,k)-firm* usa um modelo de tarefas que permite execuções precisas e imprecisas de tarefas. Nessa técnica, em qualquer janela de *k* ativações consecutivas de uma tarefa periódica, as seguintes propriedades devem ser verificadas: (i) no máximo *k-(p+i)* deadlines podem ser perdidos; e (ii) pelo menos *p* execuções devem ser precisas dentre as ativações da tarefa que possuem seus deadlines atendidos. A cada janela de *k* ativações, o limite inferior para o número de ativações com deadlines atendidos é dado por (*p+i*) (portanto, é necessário pelo menos *i* execuções imprecisas quando apenas *p* execuções precisas são completadas).

A abordagem (p+i,k)-firm é uma extensão do deadline (m,k)-firm [7] que inclui algumas propriedades da computação imprecisa descrita em [3] por Chung *et al*. Considerando uma tarefa concebida com o conceito de deadline (m,k)-firm, essa tarefa pode ser interpretada do ponto de vista do erro cumulativo [3]. Isto é, quando a tarefa perde deadlines consecutivos, sua qualidade é degradada, acumulando erros de forma que, é necessário completar pelo menos *m* execuções em cada janela de *k* invocações para manter a taxa de erro dentro de valores aceitáveis. A extensão do deadline (m,k)-firm com execuções precisas e imprecisas, permite uma tarefa executar na forma imprecisa, produzindo resultados aproximados e tendo um erro acumulado menor que quando um deadline é perdido. Além disso, se *p* execuções precisas em uma janela de *k* invocações garantem uma certa qualidade para a tarefa, então o escalonador terá um outro nível de flexibilidade para tratar sobrecargas. Uma execução imprecisa representa um tempo de computação menor, e o escalonador pode usá-la para controlar o número de perdas de deadlines de um conjunto de tarefas.

O conceito de falha dinâmica descrita em [7] é relacionado somente com a condição (i) apresentada acima. Em nossa abordagem, esse conceito é estendido para também garantir a condição (ii): — “uma falha dinâmica é assumida em uma tarefa quando o número de deadlines não atendidos supera a *k-(p+i)*, ou quando menos que *p* execuções precisas ocorrem

em uma janela de *k* invocações".

É possível mostrar que os conceitos de deadline (m,k)-firm e de erro cumulativo na computação imprecisa (Chung) podem ser representados através do uso do deadline (p+i,k)-firm. A Tabela 1 mostra alguns exemplos de descrição de tarefas usando o (m,k)-firm e a técnica de Chung, e suas correspondências no modelo (p+i,k)-firm.

| *(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* |
|---|---|
| Pelo menos 1 precisa a cada 2 execuções | (1+1,2) |
| Pelo menos 1 precisa a cada 3 execuções | (1+2,3) |
| ⋮ | ⋮ |
| Pelo menos 1 precisa a cada k execuções | (1+(k-1),k) |

**Tabela 1. Mapeando modelos de Chung e (m,k)-firm para o deadline (p+i,k)-firm.**

O modelo proposto pode ser visto como uma combinação e uma generalização dos modelos acima. Primeiro, porque ele estende o conceito de deadline (m,k)-firm, permitindo que uma tarefa execute de forma imprecisa, obtendo melhores resultados do que o caso de uma perda de deadline. Segundo, porque ele possibilita ao modelo de computação imprecisa incorporar perdas de deadline, o que é natural que ocorra em sistemas dinâmicos.

### 4.1.1 Políticas de escalonamento na abordagem (p+i,k)-firm

No APMC é suposto um grupo de tarefas competindo por serviço em uma CPU. Cada tarefa tem um deadline (p+i,k)-firm, com suas invocações (ativações) submetidas aos seus próprios valores de deadline. O objetivo do escalonador do sistema é escalonar as tarefas evitando falhas (temporais) dinâmicas. Conceitualmente, filas FIFO são definidas para a chegada de invocações de tarefas (Figura 2). A cada tarefa *j* do sistema é atribuída uma fila de chegada, assegurando que invocações de uma mesma tarefa serão servidas pelo escalonador na ordem de suas chegadas. Somente invocações de tarefas no topo de suas filas são candidatas a serem servidas, sendo inseridas em uma fila de prontos de acordo com uma *política de atribuição de prioridades*. Além dessa atribuição de prioridades, uma *política de aceitação de precisão* seleciona qual versão (precisa ou imprecisa) da tarefa será executada. A política de aceitação de precisão poderia também rejeitar completamente uma invocação de tarefa.

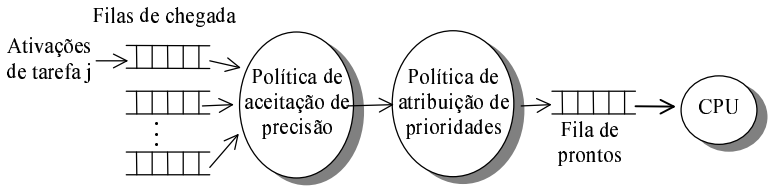


**Figura 2. Abordagem de escalonamento do modelo APMC.**

Na abordagem de escalonamento do APMC, as políticas de aceitação de precisão e de atribuição de prioridades trabalham de forma integrada, selecionando a invocação na versão precisa ou imprecisa, e atribuindo sua prioridade[2].

[2] Na verdade, esse modelo é genérico. A política de atribuição de prioridades poderia ser usada, por exemplo, para implementar o escalonamento EDF (*Earliest Deadline First*) [11], onde as prioridades são atribuídas de acordo com a urgência dos deadlines das invocações. A política de aceitação de precisão poderia ser usada com a estratégia *Red Tasks Only* [8] que descarta invocações sempre que possível, uma vez que os descartes não excedam um determinado valor (um fator de *skip*).

### 4.1.2 Tarefas com deadline (p+i,k)-firm

A política de escalonamento no APMC é baseada em um histórico de execuções precisas, imprecisas e de perdas de deadlines. Um *histórico de execução* de uma tarefa ou simplesmente *histórico* é uma k-tupla que armazena o estado das últimas *k* invocações da tarefa. Seja a seguinte representação para cada estado de invocação: *P* para execução precisa, *I* para execução imprecisa, e *X* para perda de deadline. Para cada novo estado, o histórico é deslocado (da direita para esquerda) e o novo estado adicionado. Por exemplo, em uma tarefa com deadline (2+0,4)-firm, um histórico *"PPPX"* indica que essa tarefa perdeu deadline em sua última invocação (o *X* na direita).

Também nesse modelo adaptativo, são definidas para cada tarefa *j* com deadline (p+i,k)-firm, as seguintes variáveis:

$p_j(k)$ : número de execuções precisas nas últimas k invocações;
$i_j(k)$ : número de execuções imprecisas nas últimas k invocações;
$x_j(k)$ : número de deadlines perdidos nas últimas k invocações.

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

Por definição, uma execução normal (sem falha dinâmica) de uma tarefa ocorre quando:

(i) $x_j(k) \leq k - (p+i) \wedge$
(ii) $p_j(k) \geq p$

A condição (i) limita o número máximo de deadlines perdidos, e (ii) especifica o número mínimo aceitável de execuções precisas. Ambas as condições não podem ser garantidas porque é assumido um ambiente não determinista. Entretanto, como no algoritmo DBP [7], a política de atribuição de prioridades irá garantir maiores prioridades para tarefas próximas de violar a condição (i). Além disso, a política de aceitação de precisão nunca irá liberar menos que *p* execuções precisas em quaisquer janelas de *k* invocações.

Dois atributos importantes na técnica de deadline (p+i,k)-firm são a *autonomia para perda de deadlines* e a *autonomia para execução imprecisa*. O primeiro especifica, em um dado momento, o número de deadlines consecutivos que uma tarefa pode perder sem que ocorra uma falha dinâmica; e o segundo especifica o número de invocações imprecisas consecutivas que uma tarefa tolera. Supondo como exemplo quatro tarefas: $\tau_1$: (4+0,4)-firm, $\tau_2$: (2+2,4)-firm, $\tau_3$: (0+2,4)-firm e $\tau_4$: (2+0,4)-firm. Considerando que em todas essas tarefas, nas últimas *k* invocações os deadlines foram atendidos na forma precisa, então:

- $\tau_1$ e $\tau_2$ não têm autonomia para perder deadline, mas $\tau_2$ tem autonomia para executar as próximas 2 invocações de forma imprecisa;
- $\tau_2$ e $\tau_3$ têm autonomia para executar as próximas duas invocações na forma imprecisa, mas $\tau_2$ não tem autonomia para perder deadline;
- $\tau_3$ e $\tau_4$ têm autonomia para perder os próximos 2 deadlines, mas $\tau_4$ não tem autonomia para executar na forma imprecisa.

Enquanto a política de atribuição de prioridades é dirigida pelo conceito de *autonomia para perdas de deadline*, a política de aceitação de precisão é dirigida pelo conceito de *autonomia para execução imprecisa*. Em outras palavras: enquanto a política de atribuição de prioridades redefine prioridades tentando evitar perdas de deadlines das tarefas que têm menos folga para perder deadlines; a política de aceitação de precisão irá selecionar invocações de tarefas na

forma imprecisa, em tarefas que têm mais folga para executar suas versões imprecisas.

### 4.1.3 Heurísticas de escalonamento

*Política de atribuição de prioridades*

Seja $d_j(k)$ o valor da *autonomia de perdas de deadlines* de uma tarefa *j*. A interpretação desse valor é apresentado abaixo:

| $d_j(k)$ | Interpretação |
|---|---|
| 0 | Estado de falha |
| 1 | Se perder próximo deadline então falha |
| ⋮ | |
| n | Se perder os próximos n deadlines então falha |

É possível se obter $d_j(k)$ usando uma função similar à introduzida na heurística DBP [7]. Seja $pm_j(n,s)$ a função que denota para uma tarefa *j*, a posição (da direita para esquerda) da $n^{ésima}$ execução com deadline atendido na forma precisa ou imprecisa no histórico *s*. Se existem menos que *n* deadlines atendidos em *s*, então essa função retorna *k+1*. Por exemplo, $pm_j(1,\text{“XPP"}) = 1$, $pm_j(1,\text{“XPX"}) = 2$, $pm_j(2,\text{“IXP"}) = 3$, $pm_j(2,\text{“XXP"}) = 4$.

Usando essa função, o valor da autonomia de perdas de deadline pode ser calculado por:

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

Por exemplo, uma tarefa declarada com um deadline (2+0,4)-firm e um histórico *"PPXX"* teria: $d(k) = 4 - pm_j(2, \text{“PPXX"}) + 1 = 4 - 4 + 1 = 1$, indicando que se essa tarefa perder o próximo deadline irá falhar. Entretanto, se o histórico dessa tarefa fosse *"XPXP"*: $d_j(k) = 4 - pm_j(2, \text{“XPXP"}) + 1 = 4 - 3 + 1 = 2$, indicaria que essa tarefa poderia ainda perder o próximo deadline sem falhar.

O valor de $d_j(k)$ pode ser aplicado diretamente, através do seu mapeamento nas prioridades nativas do sistema operacional. Por exemplo, supondo que a prioridade mais alta é 0 (como ocorre em alguns sistemas operacionais), para qualquer ativação de uma tarefa *j* é suficiente fazer:

$$\text{prioridade}_j := d_j(k)$$

*Política de aceitação de precisão*

Seja $v_j(k)$ a função que representa a *autonomia de execução imprecisa* da tarefa *j*. A interpretação de $v_j(k)$ é apresentada na tabela abaixo:

| $v_j(k)$ | Interpretação |
|---|---|
| 0 | Estado de falha |
| 1 | Se executar uma invocação na forma imprecisa então falha |
| ⋮ | |
| n | Se executar n invocações na forma imprecisa então falha |

O cálculo de $v_j(k)$ usa uma função $pp_j(n,s)$, que denota, para uma tarefa *j*, a posição do $n^{ésimo}$ deadline atendido com uma execução precisa no histórico *s*. Se existirem menos que *n* execuções precisas em *s*, então essa função retorna *k+1*. Por exemplo, $pp_j(1,\text{“XIP"}) = 1$, $pp_j(1,\text{”XPI"}) = 2$, $pp_j(2,\text{“PXP"}) = 3$, $pp_j(2,\text{“IXP"}) = 4$.

Usando essa função, o valor da autonomia de execução imprecisa pode ser calculado como:

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

Por exemplo, uma tarefa declarada com um deadline (2+2,4)-firm e com um histórico *"PPII"*

teria: $v_j(k) = 4 - pp_j(2, \text{“PPII"}) + 1 = 4 - 4 + 1 = 1$, indicando que a próxima execução não pode ser na forma imprecisa. Entretanto, se o histórico dessa tarefa fosse *"IPIP"*: $v_j(k) = 4 - pp_j(2, \text{“IPIP"}) + 1 = 4 - 3 + 1 = 2$, indicando que a tarefa poderia ainda ter uma execução imprecisa.

A principal função da política de aceitação de precisão é decidir, em cada tarefa, a versão precisa/imprecisa para a próxima invocação. Portanto, para cada tarefa do sistema, existe uma variável ($nx_j$) que armazena essa informação.

Como a abordagem de escalonamento não é *overload-aware* [12] (*i.e.*, não pode prever perdas de deadline), a política de aceitação de precisão é dirigida pelas ocorrências de sobrecarga. A cada indicação de uma perda de deadline no conjunto de tarefas do sistema, uma invocação de tarefa é selecionada para executar sua versão imprecisa (reduzindo sua qualidade), baseado na sua autonomia de execução imprecisa.

A política de aceitação de precisão seleciona a invocação de tarefa, no topo de uma das filas de chegada, que possui o maior valor de autonomia de execução imprecisa ($v_j(k)$) e cujo $nx_j$ especifica versão precisa (*i.e.*, $\max(v_j(k)) \wedge nx_j$ = “precisa”). A versão imprecisa dessa tarefa é selecionada para executar na próxima e nas invocações seguintes ($nx_j$ := “imprecisa”), até que a situação crítica indicada pela autonomia de execução imprecisa é alcançada ($v_j(k)=1$; isto é, a tarefa não pode executar sua versão imprecisa na próxima ativação, de outra forma uma falha dinâmica ocorrerá). Quando essa última situação é alcançada, a tarefa é assinalada para executar sua versão precisa novamente ($nx_j$ := “precisa”).

### 4.2 Avaliação da abordagem de escalonamento

O modelo (p+i,k)-firm e sua abordagem de escalonamento estão sendo avaliados através de simulações que visam estabelecer o comportamento da abordagem de escalonamento em situações de sobrecarga. Duas características importantes aferidas nas simulações são as capacidades de redução das probabilidades de falhas dinâmicas, e de redução gradual na qualidade de cada tarefa. A primeira característica implica que, em situações de sobrecarga, as perdas de deadline devem ser distribuídas entre as tarefas de forma a minimizar a quantidade de falhas dinâmicas. A segunda característica implica que, dentro do possível, uma tarefa não deve ser penalizada de forma desproporcional com relação a outras. Assim, soluções tais como o descarte completo de uma tarefa (ou de quase todas suas ativações) que pode reduzir muito o número de falhas dinâmicas em situações de cargas altas, são consideradas ruins, pois violam essa última característica.

O simulador permite a configuração do número de tarefas e as características de cada uma delas. Cada tarefa recebe requisições, cujos tempos de chegadas são distribuídos exponencialmente conforme uma *Distribuição de Poisson*. Nos experimentos, cujos resultados são apresentados na Figura 3, são utilizadas cinco tarefas com as mesmas restrições temporais (*i.e.*, mesmos tempos de computação e deadlines relativos). Os deadlines das tarefas são especificados como sendo cinco vezes os tempos máximos de computação. Para a simulação da abordagem (p+i,k)-firm, os tempos de computação das versões imprecisas são calculados como sendo aproximadamente 20% dos tempos da versão precisa. Os experimentos executados consideram que as ativações de tarefas, mesmo perdendo seus deadlines, são servidas até o final.

A abordagem de escalonamento (p+i,k)-firm é comparada com a do (m,k)-firm, usando, respectivamente, tarefas com restrições (1+1,3)-firm e (2,3)-firm. Portanto, em ambas as abordagens é tolerada a perda de um deadline cada três ativações de tarefas. Tanto a

abordagem do (m,k)-firm quanto a do (p+i,k)-firm usam os valores de deadlines absolutos para desempatar requisições que possuem a mesma prioridade. A título de comparação, as duas abordagens também são confrontadas com a tradicional EDF (*Earliest Deadline First*) [11].

| | **Falhas dinâmicas** | | |
|---|---|---|---|
| taxa média de chegada de tarefas | *EDF* | *(2,3)-firm* | *(1+1,3)-firm* |
| **0.60** | 0.01 | 0.01 | 0.00 |
| **0.70** | 0.04 | 0.03 | 0.01 |
| **0.80** | 0.15 | 0.09 | 0.02 |
| **0.90** | 0.40 | 0.28 | 0.04 |
| **0.95** | 0.63 | 0.48 | 0.06 |

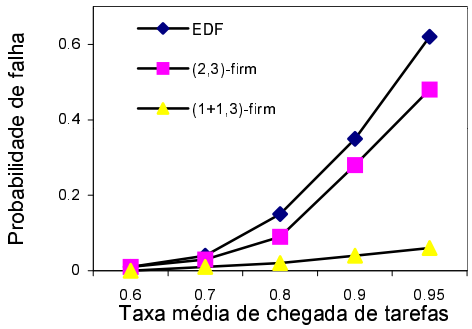


**Figura 3. Comparação entre abordagens de escalonamento.**

Os resultados mostram que a abordagem (p+i,k)-firm consegue um redução substancial na quantidade de falhas dinâmicas com relação as outras duas abordagens. O motivo é que em situações de sobrecarga, a abordagem (p+i,k)-firm pode executar transitoriamente algumas ativações de tarefas em suas versões imprecisas, reduzindo a carga média, e conseqüentemente a quantidade de perdas de deadlines. Ou seja, a possibilidade de execuções imprecisas é mais um grau de flexibilidade que apresenta o (p+i,k)-firm e que é determinante para o melhor desempenho em relação as outras abordagens.

A redução de falhas dinâmicas na abordagem (p+i,k)-firm (da mesma forma que ocorre no (m,k)-firm) também é obtida através da redução da ocorrência de perdas de deadline consecutivas de ativações de uma mesma tarefa. A Tabela 2 apresenta o resultado de um experimento para uma carga de 70%. Nessa tabela, é possível observar que enquanto na abordagem em (p+i,k)-firm não ocorreu em nenhuma tarefa cinco perdas de deadlines consecutivas, no EDF as perdas de deadline raramente são isoladas e ocorrem geralmente em seqüência.

| | **Perdas consecutivas de deadlines** | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **> 10** |
| **EDF** | 416 | 162 | 75 | 45 | 25 | 14 | 8 | 6 | 4 | 3 | 21 |
| **(2,3)-firm** | 650 | 209 | 81 | 31 | 13 | 5 | 2 | 0 | 0 | 0 | 0 |
| **(1+1,3)-firm** | 482 | 56 | 11 | 3 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Tabela 2. Perdas consecutivas de deadline com uma carga média de 0.70.**

### 4.3 Programando o modelo APMC com RT CORBA

A Figura 4 apresenta um cenário de uso do modelo APMC. A abordagem de escalonamento é implementada por um objeto de serviço CORBA presente em todos os nós — o *Serviço Adaptativo*. Esse serviço é responsável por implementar as heurísticas do APMC (políticas de atribuição de prioridades e de aceitação de versões precisas) no escalonamento de invocações a métodos do servidor. Um método é definido ou com duas versões (precisa e imprecisa) de código, através de um deadline (p+i,k)-firm; ou apenas com uma versão, através de um deadline (p+0,k)-firm (que equivale ao comportamento (m,k)-firm [7]). Cada método, com deadline (p+i,k)-firm precisa ser registrado no Serviço Adaptativo, antes de sua primeira invocação. Durante uma invocação de método com deadline (p+i,k)-firm, o Serviço Adaptativo seleciona uma de suas versões dinamicamente (se houver diferentes versões)

usando a política de aceitação de precisão. Para o cliente remoto que originou a requisição, é transparente a forma como o método invocado é executado.

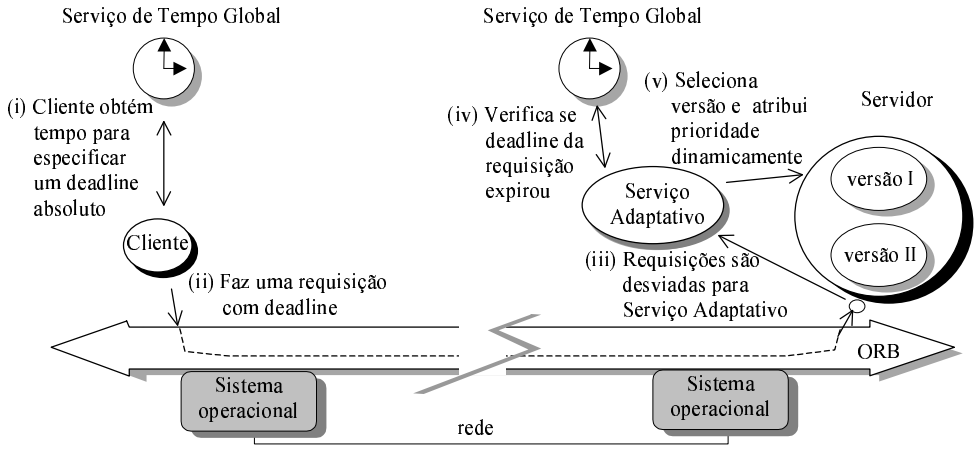


**Figura 4. A invocação de tempo real no APMC.**

Ocorrências de sobrecarga são tratadas usando técnicas adaptativas somente no nó do servidor. O cliente atribui um deadline absoluto para a invocação do método antes de enviar a requisição. Esse tipo de requisição de invocação é denominada uma *invocação de tempo real*.

Para tornar os valores de deadlines coerentes entre os nós, todos os relógios do sistema devem estar sincronizados entre si, dentro de um valor máximo (um *skew*). O APMC se baseia em um *Serviço de Tempo Global* [4]. Esse serviço estende o Serviço de Tempo CORBA, fornecendo sincronização de relógios e uma noção de tempo global. O Serviço de Tempo Global pode ser implementado em um sistema grande e disperso, usando receptores GPS, que possibilitam a uma aplicação distribuída obter um valor de tempo global preciso baseado no tempo UTC.

Em nosso primeiro protótipo, o modelo de invocação de tempo real considera apenas requisições *one-way*, *i.e.*, clientes não se bloqueiam aguardando retornos de servidores. Também nesse protótipo inicial, cada invocação é executada completamente, mesmo após o deadline. Não existem descartes ou abortos de execuções de requisição, apesar de conceitualmente um método com deadline *firm* não apresentar benefício quando executado depois de seu valor de deadline. Portanto, o passo (iv) na Figura 4 não é implementado, apesar do protótipo inicial estar sendo estendido nesse sentido. Essas extensões também irão considerar requisições *two-way*, e uma exceção será levantada tanto no cliente quanto no servidor, para tratar invocações descartadas ou sinalizações de falhas dinâmicas.

Interceptadores [17,19], *pool de threads* e fila de requisições [20] são mecanismos úteis para implementar o modelo. Interceptadores no servidor são usados para incorporar os serviços APMC no ORB, *i.e.*, requisições são desviadas para o Serviço Adaptivo de forma transparente usando interceptadores. Um *pool de threads* é criado no servidor para executar requisições de clientes. A pré-alocação de um *pool de threads* definida em um *binding* aumenta o desempenho do servidor. Filas de requisições são associadas ao *pool de threads* e usadas para implementar o modelo de escalonamento apresentado na Figura 2.

Novos mecanismos e abstrações do RT ORB também tornam possível ao Serviço Adaptativo atribuir prioridades às invocações. Esse é o caso da prioridade CORBA, que é um esquema de prioridades universal e independente de plataforma. No modelo *client-priority propagation* (citado na seção 2), cada mensagem de requisição de invocação de um cliente pode conduzir

em um campo (*IOP context service* [17]) uma prioridade que é usada no lado do servidor pelas *threads* que despacham as invocações. O Serviço Adaptativo aproveita esse campo, e insere dinamicamente na mensagem um valor calculado de prioridade, segundo a política de atribuição de prioridades do APMC.

O protótipo do modelo APMC estende o RT CORBA 1.0, possibilitando que o deadline especificado pelo cliente também seja armazenado dentro de um campo *IOP context service*[3]. Essa estratégia permite alguma transparência, porque somente o cliente e o Serviço Adaptativo precisam estar cientes da existência do deadline.

Para implementar o modelo APMC, tem sido usado o TAO [21] — um ORB em conformidade com a especificação CORBA 2.2 [17] — brevemente descrito na seção 5. O TAO possui algumas facilidades próprias para programação tempo real, e seu código-fonte é aberto e disponível na Internet. Isso nos possibilitou implementar alguns mecanismos necessários, ou fazer algumas alterações para permitir a implementação do nosso Serviço Adaptativo (por exemplo, foram implementados mecanismos para desvios de requisições, porque a atual versão do TAO ainda não possui interceptadores).

## 5. Trabalhos relacionados

O projeto DHDA [23] foi um dos primeiros trabalhos a estabelecer requisitos de tempo real em CORBA. Esse projeto usa uma abordagem de programação *best-effort* para suportar restrições temporais fim a fim em um ambiente dinâmico, com objetos sendo adicionados e removidos, e com restrições temporais que podem mudar a cada invocação. O objetivo principal é o desenvolvimento de extensões de tempo real para CORBA, seguindo uma filosofia genérica de aderência a sistemas abertos (por exemplo, usando COTS, sistemas operacionais POSIX, e redes ATM).

O TAO [21] é um ORB de tempo real que em ambientes deterministas possibilita a execução de aplicações de tempo real *hard* (tais como sistemas de aviação e embarcados). Com objetivo de fornecer QoS fim a fim, TAO integra as interfaces de rede, subsistemas de E/S, ORB, e serviços de *middleware*. A IDL CORBA é estendida para permitir a descrição de restrições temporais associadas com cada método (tais como *WCETs*). Um serviço de escalonamento de tempo real é implementado usando a política *Rate Monotonic* [11]. O TAO também estende o Serviço de Eventos CORBA (de acordo com os requisitos descritos no documento RFI CORBA [15]) fornecendo importantes funcionalidades requeridas por aplicações de tempo real, tais como escalonamento e despacho de eventos por prioridades.

As pesquisas complementares do DHDA e do TAO sobre ambientes dinâmicos e estáticos influenciaram as especificações do RT CORBA, cujo objetivo inicial era especificar abstrações CORBA amplas e flexíveis para qualquer tipo de aplicação de tempo real.

O projeto QuO [9] estabelece um modelo de programação que permite a especificação de rotinas para o tratamento de variações da carga no ambiente de execução. A meta é implementar aplicações adaptativas que podem executar em ambientes bastante dinâmicos como, por exemplo, a Internet. O modelo QuO difere do APMC, pois o primeiro estende as especificações de IDL CORBA, criando linguagens próprias para a descrição de QoS nas interfaces de objetos.

---

[3] Acreditamos que a especificação vindoura RT CORBA 2.0, que deverá ser publicada em 1999, especificará uma abordagem similar para transportar valores de deadline em mensagens de requisição de invocação dos clientes.

Em [6], interceptadores CORBA são usados para monitorar requisições para objetos CORBA, e os resultados desse monitoramento são usados em uma abordagem de escalonamento adaptativa denominada *Task-pair Scheduling*. Esse escalonamento é um tipo de implementação da computação imprecisa usando duas versões. Nessa abordagem, os resultados de monitoramento podem levar a decisão de cancelar o processamento da versão principal e ativar outra versão. Essa outra versão possui um código menor que pode representar um tratamento de exceção. A técnica de computação imprecisa implementada no modelo *task-pair* difere do APMC, já que no escalonamento *task-pair* é necessário garantir tempo de CPU para a versão imprecisa.

No RTRD [5], protocolos de meta-objeto são usados para complementar *middlewares* CORBA convencionais, permitindo às aplicações controlarem suas próprias propriedades temporais. Esse *framework* bastante flexível provê facilidades para a construção de diversos tipos de restrições temporais *soft* especificadas pelas aplicações. A abordagem de escalonamento é voltada apenas para políticas *best-effort*.

Todos os projetos e experiências acima, usando ou estendendo as especificações CORBA para aplicações de tempo real, aumentaram o interesse em obter funcionalidades de tempo real em CORBA, levando ao desenvolvimento das especificações do RT-CORBA da OMG. O modelo APMC é um desses esforços iniciais envolvendo os recém liberados padrões RT-CORBA [18-20].

## 6. Comentários finais

Usualmente, sistemas de tempo real são implementados usando tecnologias e plataformas proprietárias que aumentam seus custos de desenvolvimento e manutenção. Recentemente, especificações CORBA têm sido adotadas em muitas aplicações de tempo real, objetivando estender as propriedades de portabilidade, interoperabilidade, flexibilidade e redução de custos aos sistemas de tempo real contemporâneos.

Com as novas abstrações oferecidas pelo RT ORB que está sendo especificado pela OMG, vários tipos de modelos para programação de aplicações de tempo real podem ser desenvolvidos. Esse artigo apresentou o APMC — um modelo adaptativo orientado a objetos para programação tempo real em CORBA. O APMC permite a programação e um controle adaptativo de aplicações de tempo real baseadas na abordagem de deadline (p+i,k)-firm. Essa abordagem pode ser vista como uma combinação e generalização da computação imprecisa e do deadline (m,k)-firm. Essa técnica introduzida nesse texto estende o conceito (m,k)-firm permitindo que uma tarefa possa se executar de forma imprecisa, obtendo melhores resultados que uma simples perda de deadline. O (p+i,k)-firm, olhado sob o ponto de vista da computação imprecisa, pode ser considerado como uma extensão dessa última, onde é permitido perdas de deadline. Isso é desejável se considerarmos a aplicação da computação imprecisa em sistemas dinâmicos.

Atualmente, um conjunto de testes vem sendo desenvolvido para avaliar o modelo e sua capacidade de reduzir a probabilidade de falhas dinâmicas em situações de sobrecarga. Dessa forma, simulações estão sendo feitas para verificar o desempenho de sistemas com deadline (p+i,k)-firm, fazendo a comparação com outras técnicas adaptativas citadas nesse artigo. Parte desses testes foi apresentada neste texto.

Nossa abordagem de escalonamento, fundamentada em conceitos RT CORBA, também está sendo avaliada através da programação de experimentos envolvendo a transmissão de quadros de imagens usando codificação JPEG na compressão. O padrão JPEG oferece taxas de

compressão excelentes em imagens contínuas e simples, através da eliminação de redundâncias espaciais. Esse padrão também pode ser usado com bons resultados para vídeos em movimento. Entretanto, o principal motivo de usar JPEG em nossos experimentos é o fato que seus decodificadores podem negociar velocidade (uso de CPU) contra qualidade de imagem, através de uso de algoritmos de aproximações rápidas, porém inexatas. Em todos os experimentos, um cliente envia quadros JPEG previamente armazenados para um servidor remoto que implementa o decodificador em duas versões — uma oferecendo melhor qualidade e usando mais CPU, e outra que oferece uma qualidade "pobre" mas tendo um tempo de execução menor. Em cada experimento, estamos modelando o sistema usando diferentes valores de deadline nas requisições de clientes, e diferentes deadlines (p+i,k)-firm na implementação do servidor. Nesses experimentos, também estamos comparando nossa abordagem com o (m,k)-firm [7].

## Agradecimentos

Este trabalho foi parcialmente suportado pela CAPES e CNPq. Gostaríamos de agradecer a Clovis Petry pela sua ajuda na implementação do APMC.

## 7. Referências

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

[2] M. Caccamo, G. Butazzo, "Exploiting Skips in Periodic Tasks for Enhancing Aperiodic Responsiveness", *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, Brasil, Maio 1998.

[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*, Brasil, 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 Deadlines (m,k)-firm", *IEEE Trans. on Computer*, Apr. 1995.

[8] G. Koren, D. Shasha, "Skip-Over: Algorithms and Complexity for Overloaded Systems that Allow Skips", *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] 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.

[11] C. L. Liu, J. W. Layland, "Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment", *Journal of the ACM*, 20(1), Jan. 1973.

[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, "An Adaptive Scheduling Approach in Real-Time CORBA", *Proc. of 2nd IEEE International Symposium on Object-oriented Real-time Distributed Computing - ISORC'99*, Saint-Mallo, France, May 1999.

[14] C. Montez, R. S. Oliveira, J. Fraga, "An Adaptive Model for Programming Distributed Real-Time Applications in CORBA", *Proc. of SCCC'98*, Antofagasta, Chile, Oct. 1998.

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

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

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

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

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

[20]Object Management Group (OMG), “Realtime CORBA - Joint Revised Submission”, *Document orbos/98-10-05*, Oct. 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”, *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 e 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 e Practice of Object System*, Vol. 3(1). 1997.