# Modelo de Programação de Aplicações Tempo-Real em Sistemas Abertos

**Frank Siqueira e Joni Fraga**

## Resumo

Este trabalho apresenta um modelo para o desenvolvimento de aplicações distribuídas com requisitos de tempo-real, e inserido no contexto de sistemas abertos e heterogêneos.

O modelo de programação baseia-se na arquitetura padrão CORBA de objetos distribuídos, estendida através de reflexão computacional com o intuito de atender a requisitos de tempo-real. Desta forma, os aspectos relacionados com o tratamento de restrições temporais e de sincronização presentes em sistemas tempo-real são integrados no modelo de programação utilizando um meta-objeto para cada objeto base. As chamadas de métodos em objetos base são redirecionadas para os meta-objetos correspondentes, no sentido de verificar as restrições temporais e de sincronização impostas sobre o método solicitado. O objeto base trata apenas dos aspectos funcionais da aplicação, o que simplifica o trabalho do programador por liberá-lo da preocupação com aspectos relacionados às restrições impostas pelo ambiente tempo-real na programação do objeto base da aplicação.

Após a apresentação das principais características do modelo proposto, analisaremos o seu mapeamento sobre uma plataforma de execução baseada no sistema operacional Solaris.

## 1 Introdução

O desenvolvimento de aplicações distribuídas em ambientes heterogêneos consiste em um grande desafio para a indústria de *software*. Da mesma forma, o cumprimento de requisitos de tempo-real neste tipo de sistema tem se mostrado altamente necessário. Neste artigo nos preocuparemos com o desenvolvimento de aplicações distribuídas com restrições do tipo *soft real-time*, que permitam a adoção de uma política de melhor esforço (*best-effort*). Uma série de aplicações, envolvendo de sistemas multimídia a sistemas bancários, apresentam estas características.

Apresentaremos um modelo que tem por objetivo simplificar a programação de sistemas tempo-real em redes de computadores heterogêneas, que apresente, entre outras características, transparência em relação à distribuição e à heterogeneidade do sistema, e flexibilidade e adaptabilidade no desenvolvimento de aplicações. O processamento tempo-real introduz novos requisitos relacionados a correção temporal e performance. Este comportamento deve ser provido pelo modelo, que deve fornecer meios de expressar e implementar restrições de tempo-real em sistemas abertos e distribuídos.

*Laboratório de Controle e Microinformática, Depto de Eng. Elétrica, Univ. Federal de Santa Catarina, Florianópolis SC, Brasil.*
*Cx. Postal 476, Cep 88040-900. Fone 55.48.231.9202, Fax 231.9770, Mail: { frank. fraga }@lcmi.ufsc.br, Ftp e WWW: lcmi.ufsc.br*

O modelo proposto segue o paradigma de orientação a objetos, e deve permitir a interoperabilidade entre componentes distribuídos heterogêneos. A OMG (*Object Management Group*) propôs uma arquitetura para suportar este tipo de aplicação, amplamente aceita pela indústria de *software*. O ORB (*Object Request Broker*) é o componente mais importante da arquitetura proposta pela OMG [OMG92]. Ele permite que objetos façam e recebam requisições de métodos transparentemente em um ambiente distribuído e heterogêneo. O CORBA (*Common Object Request Broker Architecture*) é uma arquitetura padrão de ORB cujas interfaces foram especificadas pela OMG [OMG93]. Ele proporciona um ambiente que permite comunicação de forma transparente entre objetos, em um ambiente distribuído e heterogêneo. Nosso modelo é baseado na arquitetura CORBA e, em adição a suas características, possui a capacidade de representar e manipular o comportamento temporal de objetos.

Utilizamos o paradigma de reflexão computacional, introduzido em [Mae88] e utilizado em uma série de outros trabalhos [Hon92] [Hon94] [Mat91] [Tak92], com o propósito de facilitar a separação de problemas que devem ser abordados no desenvolvimento da aplicação. Desta forma, separaremos a funcionalidade (ou seja, procedimentos, métodos) do gerenciamento (escalonamento, verificação de restrições temporais e de sincronização, manipulação de exceções) da aplicação. Um conjunto de objetos base deve ser responsável pela resolução do primeiro problema, enquanto o segundo fica a cargo dos meta-objetos correspondentes. A reflexão computacional permite que o sistema analise seu próprio comportamento, podendo ajustá-lo e modificá-lo conforme necessário.

Outro dos benefícios proporcionados pela utilização de reflexão computacional no modelo consiste em permitir que seja escolhida dinamicamente a ordem de execução dos métodos de objetos de modo a satisfazer a política do melhor esforço. No modelo, um meta-objeto pode também selecionar, a partir do recebimento de uma requisição, entre múltiplas versões de um método, baseando-se nas disponibilidade de tempo para execução deste, numa estratégia de computação imprecisa [Liu94]. Antes do tempo de execução, o modelo permite que seja modificado o algoritmo de escalonamento, adotando uma política que seja mais adequada a uma aplicação específica.

Neste artigo discutiremos ainda a implementação de um protótipo do modelo proposto sobre uma plataforma de execução baseada no sistema operacional Solaris. Analisaremos a forma como os objetos do modelo são mapeados para elementos do sistema operacional, e a forma como o escalonamento a nível do modelo deve ser sobreposto ao escalonamento do sistema.

# 2 Análise do Contexto do Trabalho

## 2.1 Sistemas Abertos e Tempo-Real

O rápido crescimento das redes de computadores e os avanços obtidos na tecnologia das máquinas microprocessadas vem permitindo o processamento em tempo-real de grande quantidade de dados, inclusive com o tratamento simultâneo de sinais de áudio e vídeo. A distribuição e o controle da funcionalidade de tempo-real são requisitos intrínsecos deste tipo de sistema. Tais sistemas, por serem em geral muito grandes e naturalmente heterogêneos, nos levam a considerar uma arquitetura aberta na estruturação do modelo tempo-real. Além disso, o conceito de objetos distribuídos é implementado utilizando o modelo clássico de cliente -

servidor, que permite que clientes remotos interajam com objetos servidores através de invocações de métodos, independentemente da linguagem ou do compilador utilizado, ou da arquitetura da máquina alvo.

O processamento tempo-real deve incluir a noção de restrições de tempo de execução de tarefas do sistema, tanto no cliente quanto no servidor da aplicação. O tempo gasto na comunicação com o servidor é significante, e não pode ser ignorado pelo cliente. Desta forma, devemos analisar cuidadosamente aspectos de temporalidade tanto no cliente quanto no servidor da aplicação.

Em sistemas abertos temos as seguintes características relacionadas à temporalidade das tarefas do sistema [Tak92] :

- o tempo de comunicação não pode ser previsto;
- relógios locais não podem ser sincronizados;
- a carga dos nós do sistema é determinada dinamicamente, sendo portanto imprevisível.

Com isto, concluímos que os objetos envolvidos em uma transação devem examinar suas restrições baseados em seus relógios locais. Desta forma, o cliente que invoca um método em um objeto servidor deve controlar um tempo máximo de espera da resposta (*timeout*) baseado no seu relógio local, enquanto o objeto servidor possui um tempo para execução do método (*deadline*) menor, que será especificado na chamada e se tornará uma propriedade do método sendo executado, controlado com base no relógio do seu nó da rede.

## 2.2 CORBA

CORBA (*Common Object Request Broker Architecture*) é um conjunto de mecanismos padronizados e de conceitos, baseado em objetos distribuídos, propostos pela OMG (*Object Management Group*) para sistemas abertos. A arquitetura proposta pela OMG [OMG92] permite que objetos façam e recebam requisições de métodos (operações) de forma transparente em um ambiente distribuído e heterogêneo através do ORB (*Object Request Broker*). A especificação do CORBA [OMG93] define suas interfaces e seus componentes, determinando o papel de cada um destes no ambiente. Os componentes do CORBA são mostrados na figura 1.

No ambiente CORBA, interfaces de objetos são descritas em uma linguagem IDL (*Interface Definition Language*). Para fazer uma requisição de um método a um objeto, o cliente pode utilizar *stubs* geradas na compilação da descrição de interface da implementação do objeto, ou, alternativamente, pode construir a requisição usando a interface de invocação dinâmica - DII (*Dynamic Invocation Interface*). Para que isto seja possível, a interface do objeto deve ser adicionada a um depósito de interfaces (*Interface Repository*). A implementação do objeto (ou seja, o servidor) não conhece que tipo de interface do ORB - *stub* ou DII - foi utilizado na requisição, pois todos os aspectos que as diferenciam são tratados internamente pelo ORB.

O núcleo ORB provê os mecanismos de representação de objetos e de comunicação para que seja possível processar e executar as requisições; ele transmite a requisição e transfere o controle para a implementação do objeto utilizando esqueletos IDL, próprios para o adaptador de objetos utilizado para ativar o método chamado.

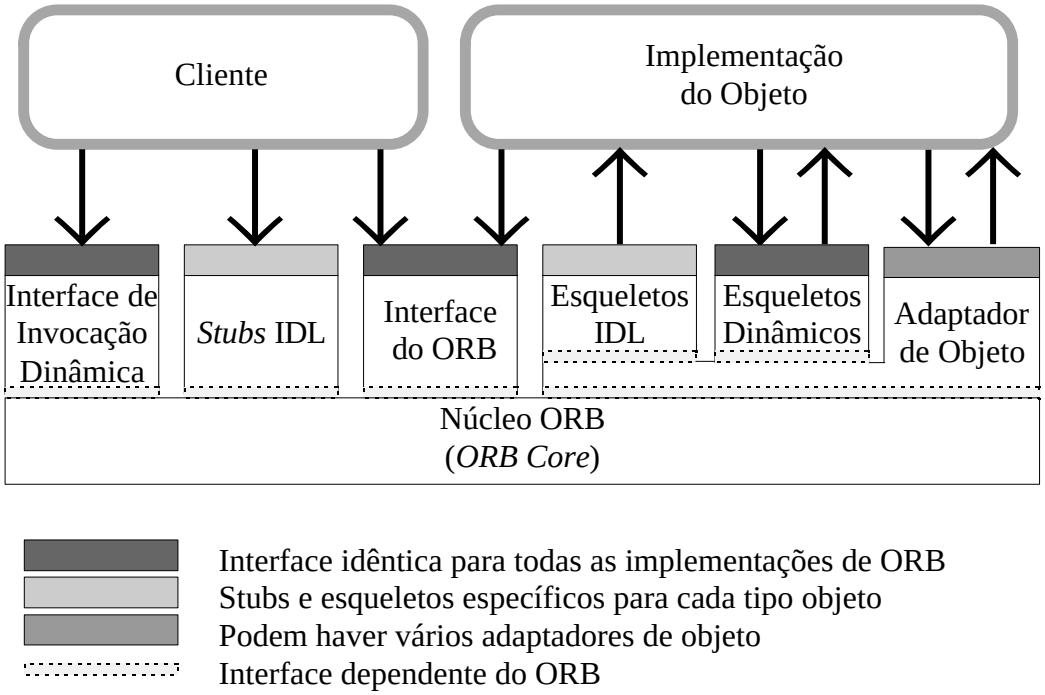


**Figura 1 - A Arquitetura CORBA**

Ao executar o serviço solicitado, a implementação do objeto pode acessar os serviços providos pelo ORB através de um adaptador de objeto. Podem haver vários adaptadores de objeto, e a implementação do objeto escolhe qual irá utilizar baseando-se no serviço que deseja obter do ORB. No entanto, toda implementação CORBA deve possuir um adaptador básico de objetos - BOA (*Basic Object Adapter*) - com serviços de propósito geral.

A interface do ORB é idêntica para todas as implementações CORBA, independentemente do adaptador de objetos utilizado. Através dela podem ser invocadas operações executadas pelo ORB úteis tanto para os clientes quanto para as implementações de objeto.

A interface de esqueletos dinâmicos - DSI [*Dynamic Skeleton Interface*] - acrescentada na versão 2.0 do CORBA, é utilizada para interconexão de ORBs. Quando um determinado ORB recebe uma requisição destinada a um objeto para o qual não possui esqueletos, este a transmite, através da interface DSI, para a interface DII de um outro ORB que possua esqueletos para o objeto.

As interfaces do CORBA são especificadas como uma camada que mascara as diferenças entre as diferentes implementações.

# 3 O Modelo Tempo-Real para Sistemas Abertos

## 3.1 Características do Modelo

### Introdução da Noção de Tempo no Modelo

O modelo proposto possui mecanismos para verificação do cumprimento de restrições temporais impostas pela aplicação e pelo ambiente quando da ativação de um método.

Restrições temporais devem ser verificadas tanto no objeto cliente quanto no objeto servidor. O cliente deve especificar um *timeout* dentro do qual deve receber uma resposta do servidor para a chamada do método, baseado em suas restrições temporais. Neste tipo de sistema fica impossibilitada uma análise de pior caso, o que faz com que o *timeout* seja um valor estimado. No servidor, deve ser verificado o cumprimento de um *deadline*, determinado em função do *timeout* do cliente e do tempo estimado de propagação da mensagem na rede.

Além da imposição de restrições temporais, podem ser associadas restrições de sincronização a métodos de objetos base, que estabelecem a ordenação que deve ser obedecida na execução destes. Estas restrições podem ser descritas na forma de grafos de precedência, álgebra de processos ou *path-expressions*, por exemplo.

**Utilização da Reflexão Computacional no Modelo**

O modelo utiliza conceitos do paradigma reflexivo na sua estruturação. Na figura 2, mostramos objetos na forma como são estruturados no modelo. Objetos são dispostos em dois níveis : nível base, ao qual pertencem os objetos base; e nível meta, onde se encontram os meta-objetos. A cada objeto base corresponde um meta-objeto que modifica dinamicamente o comportamento do primeiro. No modelo, as chamadas direcionadas ao objeto base serão desviadas (*trapped*) para o meta-objeto correspondente. Objetos base implementam a funcionalidade de clientes e servidores, enquanto os meta-objetos são utilizados para controlar as interações entre objetos através de troca de mensagens e as execuções de métodos aos quais são impostas restrições temporais e de sincronização.

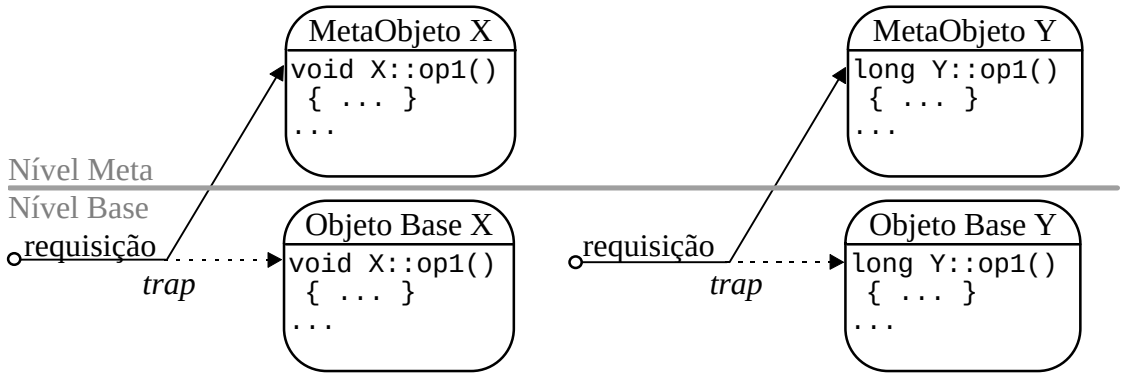


**Figura 2 : Desvio de Requisição de Método do Objeto Base para o Meta-Objeto**

**Caracterização de Métodos Segundo seu Comportamento Temporal**

Os métodos podem ainda ser classificados em três diferentes tipos, conforme as suas características de tempo :

- Periódicos : sua execução deve ser solicitada periodicamente; estes métodos podem ser ativados periodicamente por um objeto cliente; ou podem ser ativados por relógio, a partir de um tempo de inicio especificado explicitamente, ou a partir do instante de criação do objeto servidor. Em variantes deste estilo pode ser especificado um instante limite no qual deve ocorrer a última ativação do método, ou um intervalo de ativação durante o qual o método se comportará como periódico.
- Aperiódicos : suas requisições não obedecem nenhuma regra de periodicidade, podendo ocorrer a qualquer momento.

- Esporádicos : diferem de um método aperiódico basicamente por ser necessário observar um intervalo mínimo entre duas ativações sucessivas destes métodos, ou seja, existe um tempo mínimo entre ativações que deve ser respeitado.

Os métodos podem possuir ainda múltiplas versões, com tempos de execução diferentes. Nestes métodos, chamados polimórficos, a versão a ser executada será escolhida em função do tempo disponível para execução.

**Tratamento de Exceções**

A violação de uma restrição temporal de um método deve levar o meta-objeto correspondente a um estado de exceção. Existem três tipos de exceções temporais :

- exceção do tipo `reject` : acontece quando a requisição do método chega ao meta-objeto servidor e este verifica que não pode executa-lo respeitando as restrições temporais e de sincronização impostas;
- exceção do tipo `abort` : quando o método já havia começado a se executar mas não foi possível conclui-lo respeitando o *deadline*;
- exceção do tipo `timeout` : a execução do método é abortada pois o *timeout* imposto pelo cliente se esgotou; esta exceção é detectada pelo cliente e comunicada ao meta-objeto servidor.

Exceções são tratadas por manipuladores de exceção (*exception handlers*) associados aos métodos. Manipuladores de exceção são métodos especiais que tem por função, nos casos em que um estado de exceção for atingido, executar um procedimento alternativo e recompor o estado do objeto base.

A recuperação do estado do objeto torna-se possível através do armazenamento do estado do objeto base, antes da execução de um método, no espaço de endereçamento do meta-objeto correspondente, na forma de variáveis de estado. O manipulador de exceção do método deve, quando ativado pela ocorrência de uma exceção temporal, utilizar a informação armazenada pelo meta-objeto para restaurar o estado do objeto base.

## 3.2 Definição da Estrutura dos Objetos do Modelo

**Estrutura do Objeto Base**

Um objeto base do modelo, como um objeto usual, encapsula um conjunto de dados e métodos. Adicionalmente, métodos de objetos base do modelo podem ser associados a restrições temporais e a manipuladores de exceção, utilizados quando este objeto se comporta como servidor. O objeto base pode ainda associar valores de *timeout* e procedimentos de exceção às chamadas de métodos de outros objetos do modelo que executa como cliente.

O modelo utiliza um conjunto de restrições temporais pré-definidas, que permitem a representação do comportamento de vários tipos de tarefas presentes em aplicações tempo-real. Além destas, o modelo permite que sejam declarados novos tipos de restrições temporais, as quais, em adição aos tipos pré-definidos, poderão ser associadas aos métodos do objeto básico. Cada novo tipo de restrição temporal introduzido no objeto base deve possuir uma implementação no meta-objeto correspondente.

Os valores de restrições temporais são especificados na criação do objeto base ou quando o método ao qual são associadas as restrições for ativado.

**Estrutura do Meta-Objeto**

Meta-objetos são responsáveis por manipular a fila de requisições de métodos enviadas ao objeto base correspondente, e verificar as restrições de tempo e de sincronização impostas sobre as execuções dos métodos.

Meta-objetos possuem métodos de gerenciamento onde serão especificadas as atividades executadas pelo meta-objeto. Sua funcionalidade básica deve ser composta pelas seguintes tarefas :

- receber as ativações de métodos enviadas ao objeto base, possibilitando a ordenação destes pedidos em uma fila de requisições de acordo com uma política de escalonamento predeterminada (FIFO, *deadline*, prioridades, etc.);
- verificar a conformidade das restrições temporais e de sincronização impostas ao método que está sendo executado;
- executar os tratadores de exceção associados aos métodos caso restrições tenham sido violadas.

Restrições de sincronização, declaradas no meta-objeto, especificam as relações existentes entre as execuções de métodos do objeto base, com o intuito de influenciar no processamento de requisições e, conseqüentemente, no escalonamento a nível de meta-objeto. Mecanismos como *path-expressions* e autômatos finitos podem ser utilizados para descrever este tipo de restrição.

O meta-objeto deve ainda conter o código dos manipuladores de exceção (*exception handlers*) que serão executados no caso de ocorrer uma quebra de restrição durante a execução do método. Cada manipulador de exceção utilizado no objeto base deve ser implementado no meta-objeto.

As restrições temporais definidas pelo usuário e associadas a métodos do objeto base devem ser implementadas no meta-objeto. As restrições pré-definidas também podem ser redeclaradas. Um procedimento de restrição temporal deve incorporar o seguinte trabalho de processamento :

- verificar a possibilidade de a requisição ser executada respeitando os limites impostos pela restrição temporal;
- solicitar o escalonamento da operação;
- informar ao objeto cliente caso uma exceção ocorra devido ao não cumprimento de alguma restrição imposta ao método;
- executar o tratador de exceção associado ao método caso uma restrição tenha sido violada.

Desta forma, o modelo busca satisfazer uma política de ‘melhor esforço’ na execução dos métodos de objetos, procurando garantir que métodos sejam executados sem que haja tempo suficiente para concluí-los, otimizando o tempo de processamento utilizado pelo objeto servidor.

**Meta-Objetos Adicionais**

Para proporcionar a funcionalidade necessária, o modelo utiliza dois meta-objetos adicionais, responsáveis por atividades específicas.

O meta-objeto escalonador - *Meta Scheduler* - recebe requisições de métodos e as ordena segundo uma política bem definida, baseando-se para tal nas restrições temporais associadas ao método. No modelo proposto, cada nó deve ter seu meta-objeto escalonador. Cada meta-objeto escalonador trabalha independentemente na rede, influenciando apenas o escalonamento local de objetos.

O meta-objeto relógio - *Meta Clock* - é uma abstração do relógio local do sistema. Sua função no modelo é basicamente fornecer uma base de tempo local na qual se baseie a verificação de restrições temporais. Em uma requisição de método, tanto o objeto cliente quanto o servidor do método devem verificar o cumprimento de restrições temporais utilizando o meta-objeto relógio. O cliente deve verificar o cumprimento do *timeout* da chamada com base no meta-objeto relógio local, enquanto o objeto servidor deve controlar o *deadline* do método. Assim como o meta-objeto escalonador, cada nó do sistema deve ter um meta-objeto relógio local, que trabalhe independentemente dos outros relógios localizados nos demais nós da rede.

**Meta-Objetos de Interfaceamento com o ORB**

As interfaces CORBA envolvidas na comunicação entre objetos - *stubs*, DII e *skeletons* - são transparentes para o objeto base. Meta-objetos especiais inseridos pelo modelo modificam o comportamento destas interfaces, permitindo o controle de restrições temporais e de sincronização, e o tratamento de exceções no caso de violação de alguma restrição.

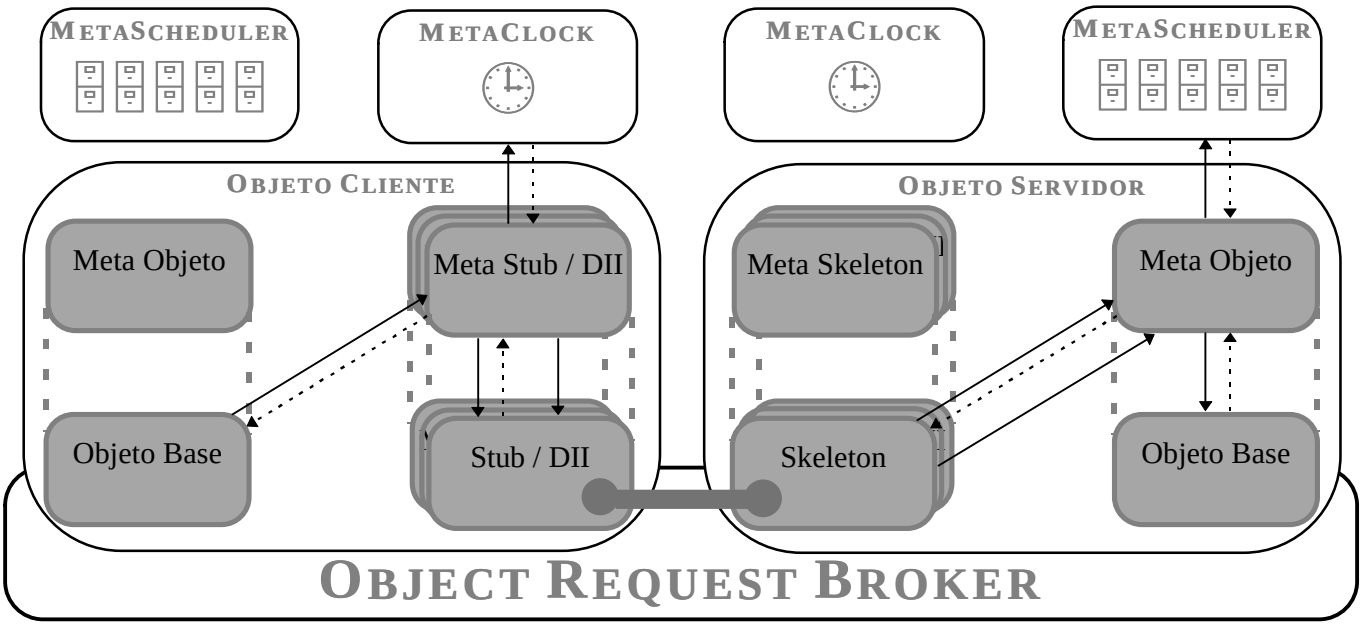


**Figura 3 - Modelo de Programação Tempo-Real para Sistemas Abertos**

Cada implementação de objeto CORBA inserida no modelo que deseje realizar interações como um cliente, deve possuir *stubs* e os correspondentes MetaStubs para comunicar-se com o servidor associado. Como alternativa à chamada por *stubs*, pode ser utilizada a interface de invocação dinâmica (DII) e o meta-objeto MetaDII presente no modelo tempo-real.

Para apresentar a funcionalidade de servidor CORBA, o objeto tempo-real deve possuir *skeletons* para as interfaces suportadas, e MetaSkeletons que modificam a funcionalidade dos esqueletos gerados pelo compilador IDL.

## 3.3 Descrição do Protocolo para Chamada de Métodos Tempo-Real

**Comportamento do Objeto Cliente**

O cliente invoca métodos de forma transparente, especificando um valor de *timeout* para o método. O protocolo proposto pode utilizar para requisição de métodos as *stubs* geradas pelo compilador IDL do CORBA, ou a interface de invocação dinâmica (DII). No primeiro caso será utilizado o objeto MetaStub correspondente às *stubs* que serão utilizadas na chamada do objeto servidor. No segundo caso, será utilizado o objeto MetaDII.

A chamada do método, que conforme a arquitetura CORBA ativaria o objeto *stub* (ou DII), será desviada para o objeto MetaStub (ou MetaDII) correspondente. Este meta-objeto comunica-se com o objeto MetaClock, criando um temporizador com o valor de *timeout* especificado. Simultaneamente, será invocado o método através da *stub* gerada pelo compilador IDL (ou através da DII). A partir de então, o cliente fica aguardando uma resposta de MetaClock ou do objeto servidor. Caso o servidor responda primeiro, a resposta retornada por este será repassada ao cliente. Esta resposta pode indicar a execução do método normalmente, ou uma exceção temporal ou de sistema (do ORB). Já se o MetaClock for o primeiro a responder, indicando que o *timeout* especificado se esgotou, será retornada ao cliente uma exceção temporal do tipo *timeout*. MetaStub (ou MetaDII) deve enviar ao servidor uma indicação de *timeout*, fazendo com que o meta-objeto servidor aborte a execução do método. Caso a chamada resulte em exceção temporal, são previstos na chamada do método no cliente procedimentos de tratamento de exceção.

Em uma chamada assíncrona (ou *oneway*, como é chamada no CORBA), não faz sentido controlar o *timeout*, já que o cliente não fica bloqueado à espera do fim da execução do método no servidor.

**Comportamento do Objeto Servidor**

O *skeleton* CORBA recebe invocações através do ORB, e chama o método correspondente no meta-objeto servidor. Este interage com o meta-objeto escalonador daquele nó para inserir a requisição na fila de pedidos. Quando liberado pelo escalonador para executar o método, o meta-objeto verifica as restrições temporais e de sincronização, se existirem. Se não puder executar o método sem violar as restrições impostas, retorna uma exceção temporal do tipo *rejected* e executa o tratador de exceção associado ao método. O meta-objeto servidor deve verificar ainda se não recebeu do cliente uma indicação de violação de *timeout*. Se tiver recebido, deve executar o tratador de exceção. Caso contrário, o meta-objeto pede a execução do método ao objeto base.

A partir do momento que iniciar a execução do método no objeto base, pode acontecer o não-cumprimento de uma restrição de *deadline*, o que leva a uma exceção temporal do tipo *aborted*. O objeto servidor deve ainda verificar se o cliente lhe envia uma indicação de *timeout*,

que deve leva-lo a abortar a execução do método e executar o tratador de exceção. Concluído o método e cumpridas as restrições temporais impostas pelo servidor, ainda poderão ocorrer exceções de sistema (do ORB) devido a problemas ocorridos na comunicação ou na desserialização (*unmarshalling*) dos dados, que serão da mesma forma enviadas ao cliente.

Caso a chamada proceda normalmente, sem que nenhum dos tipos de exceção citados anteriormente tenha acontecido, o servidor retorna ao cliente o resultado da operação executada.

# 4 Mapeamento do Modelo sobre a Plataforma de Execução

## 4.1 O Sistema Operacional Solaris

Adotaremos como plataforma de execução para um protótipo do modelo o sistema operacional Solaris versão 2.3 ou superior. O sistema operacional Solaris [*Sun*93] foi escolhido como plataforma de execução por suas características de tempo-real e por ser uma implementação completa do sistema UNIX.

Quanto a sua adequação ao ambiente tempo-real, o Solaris aproxima-se muito do especificado nos *drafts* da norma POSIX (*Portable Operating System Interface*) IEEE P1003.4 [Cor __], que especifica as interfaces do sistema operacional que tratam de escalonamento e de suporte para tempo real.

O núcleo do sistema operacional Solaris escalona processos segundo uma política baseada em prioridades. Tal política suporta o conceito de classes de escalonamento, dentro das quais vigoram políticas de escalonamento distintas. Determinada a classe que detém o processador, será estabelecido qual o processo que deve se executar, de acordo com a política de escalonamento adotada internamente pela classe. A figura 4 mostra as classes de escalonamento nas quais cada um dos processos do sistema é enquadrado.

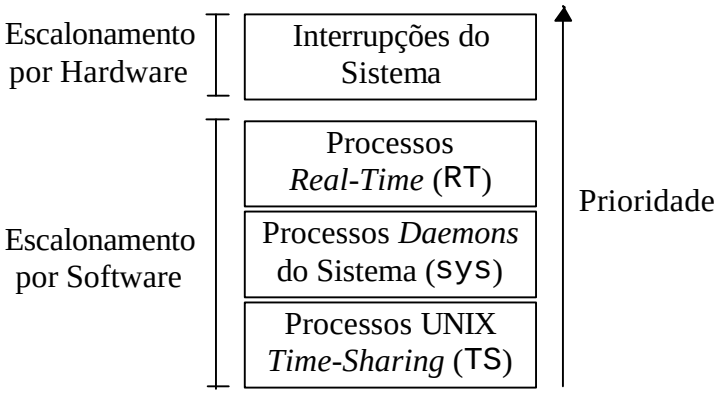


**Figura 4 - Prioridades de Escalonamento para as Classes de Processos**

Em um nível de prioridade mais alto, encontram-se as interrupções do sistema, tratadas por *hardware* e às quais o usuário do sistema não tem acesso. Em seguida vem os processos escalonados pelo núcleo do sistema operacional. Os mais prioritários são os processos tempo-real, pertencentes à classe `RT`. Em um nível intermediário encontram-se os processos do sistema operacional, os *daemons*, que fazem parte da classe `sys`. E por último estão os

processos UNIX comuns, que seguem a política de escalonamento interna por divisão de tempo (*time-sharing*), pertencentes à classe `TS`.

Um processo criado através de uma chamada `fork()` herda a classe de escalonamento do seu processo pai e seus atributos.

Os processos pertencentes à classe `RT` são associados a, além de um valor de prioridade, um *quantum* de tempo, durante o qual o processo deve se executar. As prioridades não se modificam, a não ser que uma aplicação o faça explicitamente. Com esta política de prioridades fixas, os processos `RT` se executam em uma ordem predeterminada até que se completem e abandonem a fila de processos do núcleo.

Um processo `RT` pode se executar enquanto não expirar o seu *quantum* de tempo, não terminar seu processamento, não se bloquear esperando por um evento de entrada e saída, e enquanto não perder o direito de utilizar o processador para um processo de mais alta prioridade.

O *quantum* de tempo de um processo pode ser finito ou infinito. Caso o *quantum* seja finito, o processo deve se executar no máximo durante este tempo, ao final do qual perderá o direito de utilizar o processador. Um *quantum* de tempo infinito significa que o processo não deve ser interrompido quando em execução, a não ser por processos de prioridade mais alta ou por interrupções do sistema.

## 4.2 Modelo de Execução sobre o Solaris

Objeto base e meta-objeto, em conjunto com os objetos de comunicação, devem compartilhar do mesmo espaço de endereçamento, ou seja, residir em um mesmo processo do Solaris. O meta-objeto deve utilizar *threads*, responsáveis pelo recebimento de requisições através do ORB, utilizando *skeletons* IDL e MetaSkeletons, pela verificação do cumprimento das restrições temporais e de sincronização associadas ao método, e pela interação com o meta-escalonador para obtenção de autorização para executar o método. O meta-objeto deve fazer ainda o controle de concorrência interna no objeto base, impondo a exclusão mútua na execução de métodos.

Cada meta-objeto possui uma *thread* principal de gerenciamento, que receberá através do ORB as requisições de métodos do objeto base. Ao receber uma requisição, esta é enviada a uma *thread* despachante (*dispatcher*), que verifica as restrições impostas para o método e interage com o meta-escalonador. Para cada chamada de método recebida pelo meta-objeto tempo-real, deve ser criada uma *thread* despachante. Assim que o meta-escalonador permitir a execução do método, será criada uma nova *thread* para executá-lo em prioridade mais alta. O processo Solaris no qual residem o objeto base e o meta-objeto deve ter prioridade *real-time*.

Este mecanismo de escalonamento, que implica em um certo grau de concorrência no meta-objeto, aumenta o paralelismo da aplicação e a justiça (*fairness*) do escalonamento de requisições de métodos. Caso não utilizássemos *threads*, como em grande parte dos suportes para tempo-real utilizando meta-objetos encontrados na literatura [Hon92] [Hon94] [Li94], poderíamos impedir que um método de mais alta prioridade se executasse pelo simples fato de o meta-objeto correspondente estar aguardando autorização do meta-escalonador para executar um método menos prioritário, mas que foi requisitado alguns instantes antes.

A execução do método do objeto base deve ocorrer dentro da *thread* criada para tratar a requisição de método correspondente. Como neste instante a *thread* assume prioridade mais

alta que as outras, e o processo ganha prioridade RT, não haverá ‘preempção’ do método, a não ser nos casos em que este chamar de modo síncrono um outro método com restrições de tempo associadas, e residente em outro objeto. Neste caso, será processada a chamada, o método aguardará a execução desta, mas o meta-objeto deve impedir que aconteça a execução de um outro método do mesmo objeto base.

# 5 Conclusões e Perspectivas

Consideramos que o modelo proposto atende os requisitos necessários para um sistema aberto e tempo-real. Tais requisitos foram atendidos utilizando técnicas de programação orientada a objetos e reflexão computacional, aliadas à estruturação do sistema segundo o paradigma cliente - servidor. Foi também analisado o trabalho inicial de mapeamento do sistema sobre um suporte de execução baseado no sistema operacional Solaris.

Concluído o trabalho de mapeamento do modelo sobre o suporte de execução, deve ser construído um protótipo do modelo sobre a plataforma de execução proposta. Um exemplo de aplicação com requisitos de tempo-real *soft*, possivelmente relacionada a multimídia, deve ser implementado sobre este protótipo.

# Bibliografia


**[Cor __]** Corwin. W.M. ; Locke, C.D. ; Gordon, K.D. "**Overview of the IEEE P1003.4 Realtime Extension to POSIX**", _____

**[Fra95]** Fraja, J.S. ; Farines, J-M. ; Varela, O.V. ; Siqueira, F. A. "**A Programming Model for Real-Time Applications in Open Distributed Systems**", V Workshop on Future Trends in Distributed Systems, Korea, 1995.

**[Hon92]** Honda, Y.; Tokoro, M. "**Soft Real-Time Programming through Reflection**", Proceedings of the IMSA'92 - International Workshop on New Models for Software Architecture, November 1992.

**[Hon94]** Honda, Y. ; Tokoro, M. "**Reflection and Time-Dependent Computing : Experiences with the $R^2$ Architecture**", Sony Computer Science Laboratory Inc., Technical Report, Tokio, Japan, July 1994.

**[Li94]** Li, G. "**Distributing Real-Time Objects : Some Early Experiences**", ANSA External Paper APM.1231.00.01-, Cambridge, UK, 1994.

**[Liu94]** Liu, J. S. et al. "**Imprecise Computations**", Proceedings of the IEEE, Vol. 82, No. 1, pp. 83-94, January 1994.

**[Mae88]** Maes, P. "**Issues in Computational Reflection**", Proceedings of the Workshop "Meta-Level Architectures and Reflection", Elsevier Science Publishers, 1988.

**[Mat91]** Matsuoka, S.; Watanabe, T.; Ichisugi, Y.; Yonezawa, A. "**Object-Oriented Concurrent Reflective Architectures**", Proceedings of the OOPSLA'91 Workshop on Reflection and Metalevel Architectures in Object-Oriented Programming, October 1991.

**[OMG92]** OMG - Object Management Group "**The Object Management Architecture Guide**", OMG Document Number 92-11-1, September 1992.

**[OMG93]** OMG - Object Management Group "**The Common Object Request Broker : Architecture and Specification**", Revision 1.2, OMG Document Number 93-__-__, December 1993.

**[Sun93]** Sun Microsystems, Solaris Operating System Manuals and AnswerBook, 1993.

**[Tak92]** Takashio, K. ; Tokoro, M. "**DROL: An Object-Oriented Programming Language for Distributed Real-Time Systems**", OOPSLA'92 - Object-Oriented Programming Systems, Languages, and Applications - Conference Proceedings, ACM SIGPLAN Notices, Vol. 27, No. 10, pp. 276-294, October 1992.