Um Mecanismo de Adaptação para Aplicações Tempo Real Baseado em Computação Imprecisa e Reflexão Computacional

Rômulo Silva de Oliveira
I I - Univ. Fed. do Rio Grande do Sul
Caixa Postal 15064
Porto Alegre-RS, 91501-970, Brasil
romulo@inf.ufrgs.br

Olinto José Varela Furtado
INE - Univ. Fed. de Santa Catarina
Caixa Postal 476
Florianopolis-SC, 88040-900, Brasil
olinto@inf.ufsc.br

Resumo

Diversas aplicações com restrições de tempo real são disseminadas através da Internet na forma de applets Java ou componentes Active-X. Por exemplo, aplicações que lidam com áudio e vídeo, ferramentas para trabalho cooperativo e vídeogames. A satisfação das restrições temporais torna-se um problema pois estas aplicações devem executar em diferentes plataformas (processadores e sistemas operacionais), as quais apresentam os mais variados níveis de desempenho e ocupação. Um problema importante é como projetar os componentes do software de forma que eles apresentem desempenho satisfatório nestas diferentes plataformas. Uma das técnicas que possibilitam esta adaptação é a Computação Imprecisa, na medida em que ela flexibiliza o tempo de execução das tarefas. O uso de Reflexão Computacional facilita a implementação da Computação Imprecisa, separando as questões funcionais das questões de controle responsáveis pela adaptação da aplicação. O objetivo deste artigo é mostrar como variações da Computação Imprecisa, implementadas através de reflexão computacional, podem ser utilizadas para permitir a adaptação de aplicações de tempo real a diferentes plataformas no contexto da Internet. O modelo de programação RTR será utilizado para ilustrar a forma como esta adaptação pode ser realizada.

Abstract

Many applications with real-time requirements are disseminated through the Internet as Java applets or Active-X components. For example, applications that deal with audio and video, tools for cooperative work and video games. It is difficult to satisfy timing requirements since these applications must execute on very different execution environments (processors and operating systems), with different levels of performance and utilization. An important problem is how to design software components so they present an acceptable performance even when executing on different environments. One technique to provide adaptation is the imprecise computation, since it introduces flexibility into the execution time of tasks. Also, computational reflection facilitates the implementation of imprecise computation by separating functional aspects from controlling aspects that are responsible for the adaptation. The purpose of this paper is to show how variations of imprecise computation, implemented by using refletion, can be used to allow the adaptation of real-time applications to different execution environments in the context of the Internet. The programming model RTR is used to ilustrate how this adaptation may be programmed.

1. Introdução

Sistemas computacionais de tempo real são definidos como aqueles submetidos a requisitos de natureza temporal. Nestes sistemas, os resultados devem estar corretos não somente do ponto de vista lógico, mas também devem ser gerados no momento correto. Os aspectos temporais não estão limitados a uma questão de maior ou menor desempenho, mas estão diretamente associados com a própria funcionalidade do sistema.

Na literatura os sistemas de tempo real são, em geral, classificados conforme a criticalidade dos seus requisitos temporais. Nos sistemas tempo real críticos (*hard real-time*) o não atendimento de um requisito temporal pode resultar em consequências catastróficas tanto no sentido econômico quanto em vidas humanas. Para sistemas deste tipo é necessária uma análise de escalonabilidade ainda em tempo de projeto (*off-line*). Esta análise procura determinar se o sistema vai ou não atender os requisitos temporais mesmo em um cenário de pior caso, quando as demandas por recursos computacionais são maiores. Quando os requisitos temporais não são críticos (*soft real-time*) eles apenas descrevem o comportamento desejado. O não atendimento de tais requisitos reduz a utilidade da aplicação mas não a elimina completamente nem resulta em consequências catastróficas.

O desenvolvimento de aplicações tempo real críticas é dificultado por diversos fatores. A tecnologia empregada nos sistemas computacionais para o mercado de massa (*off-the-shelf*) evolui no sentido de prover um bom comportamento médio, sem prestar atenção ao comportamento de pior caso. Mecanismos como arquiteturas *pipelines*, memória *cache* e acesso direto a memória (*direct memory access - dma*) dificultam a análise de escalonabilidade. A mesma dificuldade está presente nos sistemas operacionais mais utilizados. Felizmente, a maioria das aplicações de tempo real não são críticas, o que permite sua implementação em arquiteturas convencionais, apesar das dificuldades citadas antes.

Uma das possibilidades abertas com o emprego da Internet e do WWW (*World-Wide Web*) é disseminação aplicações na forma, por exemplo, de *applets* Java ou componentes Active-X. Entre as aplicações disseminadas estão algumas que incluem restrições de tempo real. Por exemplo, aplicações que lidam com áudio e vídeo, ferramentas para trabalho cooperativo e jogos individuais ou coletivos onde o tempo de reação do jogador é importante.

A forma como a Internet opera atualmente não suporta, de uma maneira geral, aplicações tempo real críticas [8]. Mesmo para aplicações tempo real não críticas (*soft real-time*) o uso da Internet é problemático. Uma aplicação disseminada através do WWW vai encontrar, como sua plataforma de execução, os mais diferentes processadores e sistemas operacionais. Existe uma enorme dificuldade em construir uma aplicação tempo real capaz de apresentar um comportamento temporal aceitável tanto em uma estação de trabalho último tipo quanto em um microcomputador com vários anos de uso. No sentido de viabilizar a disseminação em larga escala de aplicações tempo real, busca-se mecanismos de adaptação para estas aplicações.

A questão da adaptação de aplicações tempo real ao seu ambiente de execução não está limitada ao contexto da Internet. Ela ocorre também quando aplicações possuem um comportamento dinâmico que impede uma reserva de recursos antecipada. Entretanto, nestes sistemas a adaptação é feita, em grande parte, pelo sistema operacional ou suporte de execução. No caso de uma aplicação tempo real disseminada via Internet pouco pode ser suposto a respeito do ambiente alvo. A adaptação deve ser provida pela própria aplicação.

Uma das técnicas existentes na literatura para flexibilizar o escalonamento tempo real é a Computação Imprecisa [9]. Na computação imprecisa as tarefas da aplicação são capazes de gerar resultados com diferentes níveis de qualidade ou precisão. Para isto, cada tarefa é dividida em parte obrigatória (*mandatory*) e parte opcional (*optional*). A parte obrigatória da tarefa é capaz de gerar um resultado com a qualidade mínima, necessária para manter o sistema

operando de maneira segura. A parte opcional refina este resultado, até que ele alcançe a qualidade desejada.

Em sistemas tempo real críticos (*hard real-time*) as partes obrigatórias são garantidas através de análise de escalonabilidade em tempo de projeto (*off-line*). Desta forma é possível maximizar a utilidade do sistema, através de partes opcionais, tendo o comportamento crítico garantido pelas partes obrigatórias. Quando os requisitos temporais não são críticos (*soft real-time*) é possível que determinadas tarefas não consigam executar sequer suas partes obrigatórias. Partes obrigatória e opcional passam a ser, na verdade, partes com precisão mínima e máxima, respectivamente. A motivação para usar computação imprecisa em sistemas deste tipo reside no fato dela permitir que a aplicação adapte sua utilidade dinamicamente, conforme a disponibilidade de recursos.

Ao mesmo tempo, a técnica de Reflexão Computacional pode facilitar a implementação da Computação Imprecisa, separando as questões funcionais das questões de controle responsáveis pela adaptação da aplicação. Entre os modelos de programação propostos na literatura que incluem reflexão está o Modelo Reflexivo Tempo Real RTR [4]. RTR é um modelo de programação reflexivo e de tempo real, que caracteriza-se por permitir, de forma flexível e sistemática, a representação e o controle de aspectos temporais de aplicações tempo real que seguem uma abordagem de melhor esforço.

O objetivo deste artigo é mostrar como a técnica de Computação Imprecisa, implementada através de reflexão computacional, pode ser utilizada para permitir a adaptação de aplicações de tempo real a diferentes plataformas no contexto da Internet. O modelo de programação RTR foi utilizado para ilustrar a forma como esta adaptação pode ser implementada. O restante do artigo está organizado da seguinte forma: a seção 2 discute a adaptação de aplicações tempo real na Internet; na seção 3 é descrita a técnica Computação Imprecisa; a seção 4 descreve a técnica de Reflexão Computacional e o modelo RTR; na seção 5 é discutido como Coputação Imprecisa e Reflexão Computacional podem ser utilizados na adaptação de aplicações tempo real, sendo o modelo RTR usado para exemplificar a solução proposta; a seção 6 contém as considerações finais.

2. Adaptação de Aplicações na Internet

No caso de uma aplicação tempo real disseminada via Internet nada pode ser suposto a respeito do sistema operacional alvo. Ao ser disseminada através da Internet, e chegar a uma nova plataforma para execução, uma aplicação tempo real terá que adaptar-se ao desempenho desta plataforma. Neste caso, a adaptação deverá ocorrer através de uma ação unilateral da aplicação. Atualmente a maioria dos programas ignora este problema e não oferece qualquer tipo de mecanismo de adaptação. Desta forma, o usuário fica sujeito ao desempenho de uma aplicação executando em uma plataforma diferente daquela para a qual ela foi projetada.

Esta seção trata de mecanismos para a adaptação de aplicações tempo real. É suposto que a aplicação tempo real possui requisitos temporais não críticos que deverão ser atendidos mesmo quando esta aplicação executa em plataformas com diferentes níveis de desempenho. É suposto ainda que o suporte de execução (*middleware*, sistema operacional, arquitetura) ignora os requisitos temporais de aplicação e fornece a mesma qualidade de serviço, não negociável, para todas as aplicações. Ainda, a qualidade do serviço que a aplicação tempo real recebe do suporte durante sua execução pode variar em função do início e término de outras

aplicações que compartilham os recursos existentes. Logo, existe a necessidade de uma adaptação contínua e não somente durante a etapa de inicialização.

2.1 Mecanismos de Adaptação para a Internet

Nesta seção serão examinadas mecanismos de adaptação descritos na literatura e que podem ser empregadas no contexto considerado. O termo aplicação será usado tanto para pequenos *applets* Java quanto para sistemas compostos por milhares de linhas de código. O termo tarefa será usado para segmentos do código cuja execução possui um atributo temporal próprio, por exemplo, sua execução deve respeitar um determinado deadline ou deve ser repetida conforme um determinado período. Logo, tarefas poderão ser implementadas tanto como métodos em objetos quanto como subrotinas ou trechos de processos.

Atrasar a conclusão da tarefa. A forma mais simples e freqüente de adaptação é simplesmente relaxar o conceito de deadline. Um dos primeiros trabalhos seguindo esta abordagem aparece em [6], onde é proposto que a conclusão de cada tarefa contribui para o sistema com um benefício e o valor deste benefício pode ser expresso em função do instante de conclusão da tarefa (*time-value function*). O conceito tradicional de deadline seria uma simplificação desta visão mais ampla. Qualquer aplicação que ignore os aspectos temporais está, de certa forma, implementando este mecanismo. Entretanto, o conhecimento dos deadlines e de suas respectivas importâncias permite uma degradação mais suave do que quando deadlines são perdidos de forma aleatória.

Variar o período da tarefa. Em uma aplicação tempo real muitas tarefas são executadas periodicamente. Por exemplo, a exibição dos quadros em uma animação, o processamento de informações de áudio e vídeo, o ciclo de execução de uma máquina de simulação suportando um vídeogame. Em geral estas tarefas possuem o seu período definido em tempo de projeto. Uma forma de prover adaptabilidade à aplicação é permitir que este período possa variar dinamicamente, durante a execução da aplicação. Desta forma, a qualidade da aplicação, representada aqui pelo período das tarefas, seria adaptada ao desempenho do suporte onde estiver executando. Por exemplo, a taxa de exibição dos quadros em um vídeo (*frame rate*) pode ser alterada conforme o desempenho do suporte onde a aplicação executa.

Cancelar a execução de uma tarefa. Uma forma mais radical de flexibilização é simplesmente não executar algumas tarefas quando o desempenho estiver abaixo do desejado. No caso de aplicações com tarefas repetitivas, é possível cancelar uma ativação específica da tarefa ou cancelar completamente a tarefa. Embora o cancelamento de ativações isoladas seja menos perceptível, existem situações onde uma tarefa repetitiva pode ser completamente cancelada. Em [3] é apresentada uma solução para situações de sobrecarga onde inicialmente são descartadas ativações individuais e, caso a sobrecarga persista, passam a ser descartadas tarefas completas.

Alterar o grau de paralelismo. Muitos algoritmos dividem o trabalho a ser feito em partes. As partes podem então ser executadas em paralelo, com o objetivo de diminuir o tempo de resposta da tarefa. Caso a arquitetura sendo usada não suporte execução em paralelo, as partes são executadas sequencialmente. O resultado é o mesmo, apenas o tempo de execução será maior. Na medida em que arquiteturas paralelas tornam-se mais comuns, este mecanismo de adaptação poderá ser utilizado para tirar proveito do paralelismo existente, sem impedir a execução da aplicação em máquinas monoprocessadas. Por exemplo, em [14] é descrito um

modelo cujo objetivo é permitir adaptabilidade em aplicações do tipo C^3I (*Command, Control, Communications and Intelligence*). Componentes são programados de tal forma que eles podem adaptar, entre outras coisas, o seu nível de paralelismo durante a execução.

Variar o tempo de execução da tarefa. Neste mecanismo de adaptação, as tarefas são escalonadas de forma a cumprirem os seus deadlines. Em caso de sobrecarga, o tempo de execução da tarefa é reduzido. Para isto, é necessário que cada tarefa possua opções do tipo "qualidade versus tempo de execução". Esta abordagem é chamada na literatura de Computação Imprecisa, e será discutida na seção 3 deste artigo.

2.2 Especificação da Adaptação

Qualquer mecanismo de adaptação empregado envolve o sacrifício de alguma propriedade funcional ou temporal da aplicação. Por exemplo, não executar uma tarefa significa abrir mão ou, pelo menos, diminuir a qualidade de alguma funcionalidade. O mesmo acontece com os outros mecanismos de adaptação descritos na seção anterior. Alterações no comportamento da aplicação em função da adaptação serão percebidas pelo usuário. Logo, elas devem fazer parte da própria especificação da aplicação.

Por exemplo, o método descrito em [7] é capaz de definir a importância relativa das propriedades de uma aplicação. O método foi criado para prioritizar propriedades durante o desenvolvimento de uma aplicação, isto é, definir quais propriedades devem aparecer nas primeiras versões da aplicação e quais propriedades podem ser implementadas mais tarde. Entretanto, é possível empregá-lo no contexto da adaptação temporal. A base deste método é chamada de Processo Hierárquico Analítico (*Analytic Hierarchy Process*, AHP). No AHP as propriedades da aplicação são listadas e, duas a duas, seus valores relativos são comparados. O resultado da comparação entre cada par de propriedades é um valor numérico representando a relação de importância entre elas. O fato das propriedades serem comparadas duas a duas implica na geração de informação redundante, a qual é usada para identificar inconsistências na especificação. Isto torna o processo menos sensível a erros de julgamento. Após uma etapa de normalização dos valores o AHP fornece, para cada propriedade, o percentual de valor da aplicação que ela representa. O somatório dos valores de todas as propriedades listadas resulta em 100%, ou seja, o valor total da aplicação.

Quando uma aplicação é especificada em termos de níveis de qualidade, os níveis de qualidade descritos podem ser usados diretamente na etapa de projeto. Cada componente de software pode ter comportamentos variados. O comportamento exibido é selecionado em tempo de execução, conforme o nível de qualidade requisitado. Por exemplo, em [3] é proposto um esquema para degradação do sistema baseado em níveis de qualidade. É suposto que a aplicação possui modos de degradação (*application degradation modes*) onde ocorre o cancelamento de tarefas periódicas completas. Os modos de degradação devem ser definidos pelo projetista para refletir a semântica da aplicação, isto é, sua especificação. Em [1] níveis discretos de qualidade são definidos em termos do período e do tempo de execução das tarefas da aplicação, cujos valores são estabelecidos em função da semântica da aplicação.

2.3 Gerência da Adaptação

No contexto deste trabalho, a adaptação é originada e executada pela própria aplicação. Existe a necessidade da aplicação monitorar o seu próprio desempenho e solicitar

aos seus componentes de software o nível de qualidade apropriado, conforme a situação no momento. Esta ação pode ser resumida nos seguintes ítens:

- Coleta de informações a respeito do seu próprio desempenho temporal;
- Identificação da situação, a qual pode ser:
  - -Indesejada, o comportamento não é satisfatório e exige redução no nível de qualidade;
  - -Equilibrada, os requisitos temporais do nível de qualidade atual estão sendo aproximadamente respeitados, indicando que o nível de qualidade deve ser mantido;
  - -Favorável, os requisitos temporais do nível de qualidade atual estão sendo respeitados com folga, indicando que o nível de qualidade da aplicação pode ser elevado;
- Seleção do novo nível de qualidade desejado;
- Comunicação aos componentes da aplicação do novo comportamento selecionado, caso este seja diferente do atual.

Uma forma simples de realizar a monitoração é comparar os deadlines das tarefas com o tempo de resposta efetivamente observado. Desta forma, é possível obter uma quantificação do desempenho da aplicação com respeito aos seus requisitos temporais. Existem duas quantificações básicas: o somatório dos atrasos de todas as tarefas e a taxa de tarefas que perderam o respectivo deadline. O somatório dos atrasos das tarefas fornece uma medida mais apropriada do desempenho quando as tarefas devem ser executadas mesmo com atraso. Por outro lado, a taxa de tarefas que perderam o deadline é uma medida útil quando tarefas possuem deadline firme (*firm deadline* [2]), isto é, não existe benefício em executar uma tarefa após o seu deadline.

A vantagem da gerência ser feita pela própria aplicação é a possibilidade de aproveitar o conhecimento semântico que ela tem do seu estado. Este conhecimento semântico permite uma gerência superior ao que seria conseguido, por exemplo, por um sistema operacional que ignorasse completamente o significado das restrições temporais da aplicação.

3. Computação Imprecisa

Aplicações de tempo real necessitam atender requisitos temporais, os quais geralmente correspondem a prazos definidos pelo ambiente do sistema. Logo, existe a preocupação de "concluir o trabalho no tempo disponível". A dificuldade encontrada para atender requisitos temporais levou alguns autores a proporem uma abordagem do tipo "fazer o trabalho possível dentro do tempo disponível". Em outras palavras, sacrificar a qualidade dos resultados para poder cumprir os prazos exigidos. Esta técnica, chamada Computação Imprecisa (*Imprecise Computation*, [9]), flexibiliza a execução de uma aplicação de tempo real. Uma descrição detalhada dos diversos aspectos desta técnica pode ser encontrada no livro [12].

Computação Imprecisa está fundamentada na idéia de que tarefas podem possuir uma parte obrigatória (*mandatory*) e uma parte opcional (*optional*). A parte obrigatória da tarefa é capaz de gerar um resultado com qualidade mínima. A parte opcional então refina este resultado, até que ele alcançe a qualidade desejada. O resultado da parte obrigatória é dito impreciso (*imprecise result*), enquanto o resultado das partes obrigatória mais opcional é dito preciso (*precise result*). Uma tarefa é chamada de tarefa imprecisa (*imprecise task*) se for possível decompo-la em parte obrigatória e parte opcional. É importante observar que algumas tarefas podem possuir apenas parte obrigatória ou apenas parte opcional.

Em situações normais, a aplicação gera resultados com a precisão máxima, pois tanto a parte obrigatória quanto a parte opcional são executadas. Em situações de sobrecarga algumas partes opcionais são descartadas. A vantagem deste mecanismo está em permitir uma degradação controlada do sistema, na medida em que é possível determinar o que não será executado em caso de sobrecarga.

Existem muitas situações onde esta abordagem é possível. Por exemplo, considere uma figura monocromática representada por um mapa de bits onde cada pixel é codificado com 16 bits. Em sobrecarga é possível ignorar os 8 bits menos significativos e considerar cada pixel representado apenas pelos 8 bits mais significativos. Técnica semelhante pode ser feita com amostras de áudio. A apresentação de vídeo na tela pode ter sua resolução e tamanho ajustados. Imagens geradas pelo computador podem ser mais ou menos refinadas. Cada aplicação oferece oportunidades específicas para a utilização da Computação Imprecisa.

Uma tarefa imprecisa pode ser implementada através de múltiplas versões (*multiple versions*). Na maioria das vezes são empregadas duas versões. A versão primária gera um resultado preciso, mas apresenta um tempo de execução desconhecido ou muito grande. A versão secundária gera um resultado impreciso, em um tempo de execução menor e conhecido. A cada ativação da tarefa deve ser escolhida qual das versões será executada. No mínimo, deve ser executada a versão secundária, que corresponde a parte obrigatória. A parte opcional é definida pela diferença entre os tempos de execução das versões primária e secundária. Esta técnica é a mais flexível do ponto de vista da programação, uma vez que os algoritmos usados nas duas versões podem ser completamente diferentes.

Funções de melhoramento (*sieve functions*) são aquelas cuja finalidade é melhorar os dados de entrada de alguma forma. Se o resultado recebido como entrada por uma função de melhoramento é aceitável como saída, então a função pode ser completamente omitida (não executada). As funções de melhoramento normalmente formam partes opcionais que seguem algum cálculo obrigatório. Tipicamente, não existe benefício em executar parcialmente uma função de melhoramento. Isto significa que o escalonador deve optar, antes de iniciar a tarefa, entre executa-la completamente ou descarta-la. Por exemplo, os cenários de fundo em jogos podem ser mais ou menos detalhados, dependendo do desempenho do sistema onde a aplicação executa.

As funções monotônicas (*monotone functions*) são definidas como aquelas cuja qualidade do resultado aumenta (ou pelo menos não diminui) na medida em que o tempo de execução da função aumenta. Tipicamente a função monotônica tem a forma de um laço onde cada iteração refina um pouco mais o resultado até então obtido. As computações necessárias para obter-se um nível mínimo de qualidade correspondem à parte obrigatória. As computações além deste mínimo são consideradas como parte opcional. Algoritmos deste tipo podem ser encontrados nas áreas de cálculo numérico, estimativa probabilista, pesquisa heurística, ordenação, entre outras. Por exemplo, algoritmos de inteligência artificial utilizados em vídeogames podem ser programados desta forma. Este tipo de função faz com que o escalonador tenha que decidir quanto tempo de processador cada parte opcional deve receber. É a forma de programação que fornece maior flexibilidade ao escalonador, pois qualquer tempo extra de processador que a função recebe melhora o resultado gerado.

A forma de programação empregada dita regras para o módulo responsável por determinar quanto tempo de processador cada tarefa recebe. Quando uma função monotônica

é empregada, o tempo de processador alocado à parte opcional pode ser qualquer valor entre zero e o tempo máximo de execução da parte opcional. Quando a parte opcional executa uma função de melhoramento o escalonador deve decidir se executa a função de melhoramento completamente ou a descarta. Esta característica é chamada na literatura de restrição 0/1 (*0/1 constraint*). Multiplas versões também criam uma restrição 0/1, pois requerem a execução completa da parte opcional (escolha da versão primária) ou o seu completo descarte (escolha da versão secundária).

No contexto das aplicações que utilizam a Internet, a flexibilização do tempo de execução parece ser o mecanismo de adaptação mais promissor. A possibilidade de empregar múltiplas versões permite ao projetista da aplicação determinar comportamentos diferentes, conforme o desempenho do sistema onde a aplicação executa. A maior desvantagem deste mecanismo é a necessidade de explicitamente projetar e programar a aplicação para suportar os conceitos da Computação Imprecisa.

4. Reflexão Computacional

Reflexão é a técnica pela qual um sistema pode raciocinar e atuar sobre si próprio. Os sistemas computacionais reflexivos contém dados que representam a estrutura e os aspectos computacionais do próprio sistema; desta forma, é possível monitorar e modificar a estrutura e o comportamento do sistema através de computações realizadas pelo próprio sistema.

A abordagem comumente empregada para implementação de sistemas reflexivos orientados a objetos, proposta inicialmente em [10], tem sido a utilização de meta-objetos. Segundo esta abordagem, a cada objeto "x" é associado um meta-objeto "^x", o qual representa os aspectos estruturais e comportamentais de "x". Desta forma, a estrutura e o comportamento de "x" podem ser ajustados dinamicamente através de computações realizadas em "^x".

A abordagem baseada em meta-objetos separa os aspectos funcionais dos aspectos não funcionais, permitindo assim que a solução do problema em si (com relação as suas funcionalidades básicas) seja expressa através de objetos-base e que o controle do comportamento desses objetos (adaptando-os a um domínio específico) seja expresso através de meta-objetos. Esta abordagem flexibiliza o desenvolvimento/programação de aplicações de tempo real, na medida em que permite que as políticas de controle sejam modificadas ou substituídas (inclusive dinamicamente), sem que os objetos base da aplicação e o suporte de execução necessitem ser modificados; assim sendo, a evolução do sistema bem como sua independência de ambiente opercional ficam facilitadas.

4.1 Reflexão Computacional e Tempo Real

Embora pouco explorada no domínio tempo real ([4], [5] e [11]), a abordagem de reflexão computacional é vista como uma abordagem promissora no que se refere a estruturação de sistemas tempo real complexos [16], apta a contribuir nas questões de flexibilidade e gerenciamento da complexidade dos sistemas tempo real atuais e futuros.

O uso de reflexão no domínio tempo real, permite:
- adicionar ou modificar construções temporais (via meta-objetos) específicas de um domínio (ou de uma aplicação);

- definir um comportamento alternativo para o caso de exceções temporais (não satisfação de *deadlines*, por exemplo);
- substituir/alterar o algoritmo de escalonamento, adequando-o à aplicação e/ou ao ambiente;
- mudar o comportamento do programa, em função de informações obtidas em tempo de execução, como por exemplo disponibilidade de tempo, carga do sistema e atributos de *QoS*.
- definir e/ou ajustar o tempo de execução das atividades do sistema, em função da experiência adquirida com a evolução deste;
- prover independência entre aplicação e ambiente operacional, através da implementação de controles (tipicamente realizados a nível de *runtime* ou sistema operacional) no meta-nível da aplicação;
- incrementar a portabilidade dos sistemas favorecendo sua utilização em ambientes abertos.

Entretanto, apesar do potencial da abordagem reflexiva, seu uso para tempo real pode ser questionado com relação aos aspectos de performance e previsibilidade. Com relação a performance, admite-se um *overhead* adicional devido ao processamento reflexivo. Com relação a previsibilidade, o problema não está na reflexão em si, mas sim no fato de que ela habilita a produção de sistemas tempo real muito mais flexíveis [11], para os quais a análise de pior caso ainda é uma questão de pesquisa em aberto; contudo, embora a questão de previsibilidade possa dificultar (ou mesmo impedir) o uso de reflexão na programação de sistemas de tempo real *hard*, sua utilização na programação de sistemas de tempo real *soft* (onde as exigências de previsibilidade são menos severas) é perfeitamente viável e promissora.

4.2 Modelo RTR

O Modelo RTR (Reflexivo Tempo Real) [4] é um modelo de programação para aplicações tempo real que se caracteriza como sendo uma extensão **reflexiva**, baseada na abordagem de meta-objetos, do modelo de objetos convencional. No modelo RTR todos os aspectos temporais (restrições, exceções e escalonamento) e mais os aspectos de concorrência e sincronização são tratados de forma reflexiva, possibilitando o uso de diferentes mecanismos e políticas no controle do comportamento dos aspectos refletidos.

Estruturalmente o modelo RTR é composto por objetos-base de tempo real, meta-objetos gerenciadores (um para cada objeto-base tempo real existente), um meta-objeto escalonador e um meta-objeto relógio, os quais interagem através de mensagens (ativações de métodos) síncronas e assíncronas, visando a realização das funcionalidades da aplicação de acordo com as restrições temporais associadas aos métodos dos objetos-base.

4.3 Visão Geral dos Componentes Básicos do Modelo RTR

Objetos-base - Os Objetos-Base de tempo real implementam a funcionalidade da aplicação e, em adição ao modelo de objetos convencional, podem ter restrições temporais e manipuladores de exceções temporais associados à declaração e a ativação de métodos. Além das restrições temporais pré-definidas (*Periodic, Aperiodic e Sporadic*), novos tipos de restrições temporais podem ser declarados e utilizadas pelo programador da aplicação

Meta-objetos gerenciadores (MOG) - Os meta-objetos gerenciadores são objetos *multi-threads* responsáveis pelo controle de concorrência e sincronização, pelo tratamento de exceções e pelo processamento das restrições temporais e seus objetos-base correspondentes. Para prover a semântica de execução correspondente a cada tipo de restrição temporal, o

MOG interage com o meta-objeto escalonador (MOE), o qual determina, de acordo com a política de escalonamento implementada, o momento a partir do qual o pedido de ativação sendo analisado pode ser liberado para execução. Neste momento, em função da disponibilidade de tempo e observados os aspectos de sincronização e concorrência, é decidido pela liberação da execução do método solicitado ou pelo levantamento de uma exceção temporal.

Meta-objeto escalonador (MOE) - Este meta-objeto tem como função implementar uma política de escalonamento (escolhida pelo programador) que melhor se adapte às especificidades da aplicação e do ambiente em questão.

Meta-objeto relógio (MOR) - O MOR é uma abstração do relógio do sistema, estruturada na forma de objeto. Sua função básica é ativar métodos num tempo futuro e controlar a passagem de tempo visando a detecção de violações temporais.

Por ser um modelo abstrato, independente de linguagem de programação, a utilização do Modelo RTR depende de uma implementação concreta que estabeleça a disciplina de programação necessária para obtenção do comportamento especificado. Neste sentido, foram realizadas duas implementações experimentais baseadas em simulação [13] e [15] e definida uma extensão reflexiva e tempo real da linguagem Java, denominada Java/RTR [4], a qual permite a implementação explícita das funcionalidades do modelo.

5. Computação Imprecisa no Modelo RTR

O suporte à computação imprecisa no modelo RTR envolve tanto a especificação da imprecisão, através de construções que representem a estrutura inerente a cada uma das formas de programação imprecisa usuais, quanto o controle relativo a execução dessas construções. O uso de reflexão computacional no modelo RTR possibilita a separação explícita entre estes dois aspectos, sendo que a questão da representação é tratada no nível base enquanto que o controle é tratado no nível meta, de forma independente tanto do programador dos objetos-base quanto do suporte de execução.

A representação da imprecisão dá-se através da definição de novos tipos de restrições temporais (representando as diferentes formas de programação de computação imprecisa) as quais são associadas a declaração dos métodos dos objetos-base que representam tarefas imprecisas. Por outro lado, o controle destas restrições, da mesma forma que os demais tipos de restrições temporais suportados pelo modelo, é implementado através de métodos dos meta-objetos gerenciadores, os quais interagindo com o meta-objeto escalonador (MOE) proverão a semântica de execução correspondente a cada restrição considerada.

5.1 Representação da Imprecisão nos Objetos-Base

No modelo RTR a representação de tarefas imprecisas dá-se através da associação de restrições temporais aos métodos dos objetos-base. São definidos novos tipos de restrições temporais, uma para cada forma de programação da computação imprecisa. Quando associados aos método dos objetos-base, estas restrições temporais farão com que os mesmos comportem-se como tarefas imprecisas. A representação da imprecisão, a nível de objetos-base, compreende:

- A definição de um novo tipo de restrição temporal através da declaração RT-Type ou a disponibilização via herança da restrição desejada;

- A associação da restrição ao método que deverá comportar-se como uma tarefa imprecisa;
- A especificação dos valores dos atributos da restrição, quando da ativação de métodos representando tarefas imprecisas.

A definição dos atributos associados com cada restrição temporal pode variar de uma aplicação para outra, e deverá ser decidida conjuntamente com a definição da semântica de execução da restrição e com o esquema de escalonamento escolhido para a aplicação.

Múltiplas Versões (*Multiple Versions*) – Nesta forma de programação uma determinada tarefa da aplicação pode possuir diferentes versões com diferentes tempos de execução. Em resposta a ativação de uma tarefa, uma destas versões será escolhida dinamicamente para execução de acordo com a semântica de execução e o esquema de escalonamento implementados no meta-nível da aplicação.

No modelo RTR a noção de múltiplas versões pode, por exemplo, ser representada através da restrição temporal *TimingPolymorphic( )* [4] :

```
(* Declaração de um novo tipo de restrição temporal *)
RT-Type TimingPolymorphic = (Deadline, <MethodList>);
```

Nesta declaração, o atributo *Deadline* especifica o limite máximo de tempo dentro do qual a execução do método ao qual esta restrição vier a ser associada deverá ser concluída, enquanto que o atributo <MethodList> especifica os identificadores dos métodos que representam as múltiplas versões (neste caso uma quantidade variável) que poderão vir a ser executadas em resposta à ativação de um método polimórfico-temporal.

```
(* Associação da restrição temporal a um método do objeto-base *)
void DisplayImagem( ... ), TimingPolymorphic (D, Met1="DI-Versao1",
                                    Met2="DI-Versao2", Met3="DI-Versao3")
begin  ...  end;
. . .
(* Implementação das múltiplas versões *)
void DI-Versao1 ( ... )  // Apresenta imagem de ótima qualidade
begin  ... end;
void DI-Versao2 ( ... )  // Apresenta imagem de boa qualidade
begin ... end;
void DI-Versao3 ( ... )  // Apresenta imagem com qualidade minima
begin ... end;
```

Neste exemplo, as diferentes versões do método "DisplayImagem()" foram especificadas estaticamente na declaração deste método, enquanto que o atributo deadline ("D") permaneceu variável, devendo ser especificado explicitamente a cada ativação do método "DisplayImagem()".

```
(* Ativação da tarefa polimórfica-temporal *)
Objeto.DisplayImagem(),(40);
```

Alternativamente, as versões a serem utilizadas podem ser especificadas dinamicamente em cada ativação do método em questão. Entretanto, em qualquer caso o objeto-base deverá conter métodos representando as diferentes versões utilizáveis.

Funções Melhoramento (*Sieve Functions*) - Esta forma de computação imprecisa consiste em uma tentativa de aprimorar um resultado disponível aproveitando uma disponibilidade adicional de recursos de processamento. No modelo RTR, funções melhoramento podem ser

representadas através da restrição temporal “SieveFunction” exemplificada a seguir. Sua função é sinalizar para o nível meta o caráter opcional da tarefa invocada.

```
(* Declaração da restrição temporal *)
RT-Type SieveFunction = (Deadline);
```

Além da especificação do deadline, conforme a estratégia de escalonamento em uso, outros atributos, tais como valor do “melhoramento” para o sistema, podem ser levados em conta e compor a definição deste tipo de restrição temporal.

```
(* Associação da restrição a um método do objeto-base *)
void AprimoraPrecisao ( ... ), SieveFunction  (D)
begin  ...  end;
. . .
(* Ativação da tarefa imprecisa *)
Objeto.AprimoraPrecisao(), (30);
```

Funções Monotônicas (*Monotone Functions*) - Os aspectos de controle relativos a esta forma de programação estão intimamente relacionados com os aspectos algoritmicos da tarefa e portanto não há como separá-los completamente como ocorre com as outras formas de programação de computação imprecisa.

Assim sendo, uma tarefa representando uma função monotônica deverá ser programada de forma que o tempo permitido para sua execução seja explicitamente considerado na lógica desta tarefa, além de poder ser definido dinamicamente pelo escalonador da aplicação. Para tanto, estas funções devem ser explicitamente diferenciadas através da definição e do uso de uma restrição temporal, capaz de garantir a semântica de execução desejada.

```
(* Declaração da Restrição temporal *)
RT-Type MonotoneFunction = (Deadline, MinExecTime);
(* Associação da restrição a um método do objeto-base *)
void DefinePosition(AvaibleET, ...), MonotoneFunction(D, MinET), ExceptionDP()
begin  ...  end;
. . .
(* Ativação da tarefa imprecisa *)
Objeto.DefinePosition (10,…), (50,10);
```

O valor correspondente ao tempo disponível para execução (Atributo AvaiableET) será redefinido no momento da ativação do método DefinePosition() por parte do meta-objeto gerenciador (via interação com o meta-objeto escalonador). Este valor não poderá ser inferior ao tempo mínimo de execução (valor associado ao atributo temporal MinET) estabelecido no nível-base quando da ativação do método representando uma função monotônica.

5.2 Controle e Monitoração das Tarefas Imprecisas nos Meta-Objetos Gerenciadores

De acordo com a abordagem RTR, as restrições temporais representando as diferentes formas de programação de computação imprecisa devem ser implementadas através de métodos (um para cada restrição) nos meta-objetos gerenciadores. Assim sendo, sempre que o meta-objeto gerenciador (MOG) interceptar a ativação de um método ao qual esteja associada uma dessas restrições, o fluxo de execução será desviado para o método que implementa a restrição em questão, o qual controlará a execução do método do objeto-base solicitado originalmente.

Este controle será feito a partir da informação "nível de qualidade escolhido", o qual é decidido pelo meta-objeto escalonador (MOE) e comunicado a cada MOG da aplicação sempre que for alterado. O "nível de qualidade escolhido" permite ao MOG determinar o comportamento de cada método no seu respectivo objeto base. A tabela 1 ilustra como uma tabela simples, definida em tempo de projeto, pode ser usada para tanto.

	Método M1 *(função monotonica)*	**Método M2** *(função melhoramento)*	**Método M3** *(múltiplas versões)*	**Método M4** *(múltiplas versões)*
Nível Máximo	sem limite	sim	primária	primária
Nível Médio	100 iter.	sim	secundária	primária
Nível Mínimo	50 iter.	não	secundária	secundária

Tabela 1 - Tabela usada por um meta-objeto gerenciador.

O objeto ilustrado pela figura 1 possui 4 métodos. O método M1 foi programado como função monotônica, o método M2 é uma função melhoramento, enquanto os métodos M3 e M4 possuem múltiplas versões. Para a aplicação em questão o "nível de qualidade escolhido" pode assumir três valores: máximo, médio e nível. Durante a execução o MOG é informado pelo MOE qual nível deve ser utilizado. A tabela 1 define, para cada possível nível de qualidade, qual deve ser o comportamento de cada método, com respeito a sua precisão. Por exemplo, quando o "nível de qualidade escolhido" é o nível máximo, o método M1 não possui um limite de iterações e a versão primária do método M4 é executada. Se o MOE decide passar a aplicação para o nível médio, então o método M1 passa a executar no máximo 100 iterações, enquanto o método M4 continua empregando a versão primária. Finalmente, se o MOE comunica ao MOG que o "nível de qualidade escolhido" é o nível mínimo, então o método M1 pode executar apenas 50 iterações e a versão secundária do método M4 será empregada.

Os próximos parágrafos discutem a semântica de execução associada a cada uma das restrições temporais introduzidas na seção anterior (ou seja, a função dos métodos que as implementam), destacando que esta semântica pode variar de uma aplicação para outra (exigindo diferentes implementações) e em cada caso dependerá da estratégia de escalonamento considerada.

Múltiplas versões - A implementação da restrição “TimingPolymorphic” dá-se através da definição de um método, também denominado TimingPolymorphic(), cuja função consiste em escolher dinamicamente qual das versões deverá ser executada em resposta à ativação de um método declarado como sendo polimórfico temporal. Esta escolha envolve o "nível de qualidade escolhido" atual e dependerá da estratégia de adaptação da aplicação.

Funções melhoramento – A semântica associada a restrição temporal que representa esta forma de programação de computação imprecisa é implementada através do método SieveFunction() do meta-objeto gerenciador, cuja função básica será interagir com o meta-objeto escalonador para verificar a possibilidade ou não de execução da tarefa solicitada antes do esgotamento do deadline especificado. Outros controles podem ser implementados no

contexto desta restrição. Por exemplo, controles para forçar a execução da tarefa em questão quando o número de violações atingir um patamar previamente estabelecido.

Funções monotônicas – Tendo em vista o fato de que nesta forma de programação de computação imprecisa é impossível isolar-se completamente os aspectos de controle dos aspectos algoritmos da aplicação, a capacidade reflexiva do modelo RTR será usada para definição dinâmica do tempo durante o qual o método (aqui representando uma função monotônica) ativado deverá executar. Assim sendo, a função principal do método MonotoneFunction() que implementa a restrição em questão, será determinar este tempo e comunicá-lo ao método do objeto-base. A comunicação deste tempo para o método do objeto-base poderá ser feita explícita ou implicitamente. No caso explícito o valor obtido pode ser transmitido na forma de um parâmetro funcional quando da efetiva ativação do método do objeto-base, enquanto que no caso implícito, uma exceção temporal (possivelmente levantada pelo meta-objeto relógio) poderá ser sinalizada pelo MOG e detectada pelo objeto-base.

Além de controlar o comportamento do objeto-base, o MOG deve também monitorar o desempenho temporal daquele. Isto é feito através da medição do tempo de resposta de cada método e comparação com o respectivo deadline. O meta-objeto relógio é usado como objeto auxiliar na manipulação das informações temporais. As métricas típicas a serem obtidas são o número de deadlines perdidos e o atraso médio dos métodos que não atenderam o deadline. Estes valores são comunicados ao MOE periodicamente. A iniciativa da comunicação pode tanto partir do MOG (informe periódico) quanto do MOE (consulta periódica).

5.3 Seleção do Nível de Qualidade pelo Meta-Objeto Escalonador

A função do MOE é definir o "nível de qualidade escolhido" para aplicação. Esta informação será comunicada aos vários MOGs e será usada para definir o nível de precisão de cada método. A escolha do nível de qualidade é feita a partir do comportamento atual da aplicação, monitorado pelos MOGs e comunicado periodicamente ao MOE. O nível de resposta atual da aplicação é comparado com valores estabelecidos em projeto e poderá disparar uma mudança global de comportamento. A mudança entre níveis de qualidade é determinada por gatilhos definidos também em projeto. Por exemplo, o número de deadlines perdidos dividido pelo número de ativações de métodos.

A figura 1 apresenta um diagrama de estados ilustrando o papel dos gatilhos. O MOE assume a forma de uma máquina de estados onde cada estado representa um possível "nível de qualidade escolhido" enquanto as transições entre estados representam os gatilhos definidos em tempo de projeto. Por exemplo, caso o "nível de qualidade escolhido" atual seja o máximo e a quantidade de deadlines perdidos supere 70%, o MOE passará o "nível de qualidade escolhido" para médio e informará os diversos MOGs.

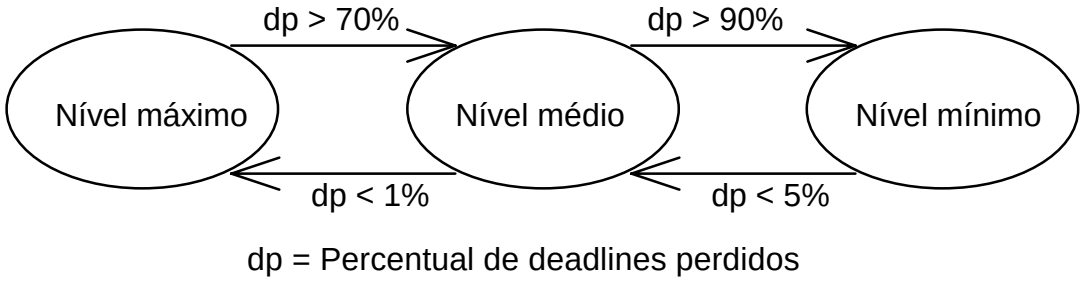

Figura 1 - MOE estruturado como uma máquina de estados.

Existe uma série de decisões importantes a serem tomadas no momento do projeto da aplicação. É possível destacar as questões: "quantos níveis de qualidade a aplicação poderá exibir", "que tipo de métrica será usada na construção dos gatilhos" e "quais os valores da métrica em questão serão associados com cada gatilho em particular". Embora estas questões não possam ser respondidas facilmente de forma genérica, elas ficam mais simples quando uma aplicação em particular é considerada. A resposta para estas e outras questões de projeto estão diretamente ligadas à semântica da aplicação.

6. Conclusões

Este artigo mostrou como a técnica de Computação Imprecisa, implementada através de reflexão computacional, pode ser utilizada para permitir a adaptação de aplicações de tempo real a diferentes plataformas no contexto da Internet. Inicialmente foram considerados diversos aspectos relacionados com a adaptação de aplicações de tempo real. Em seguida, as técnicas Computação Imprecisa e Reflexão Computacional foram discutidas. Finalmente, o modelo de programação RTR foi utilizado para ilustrar a forma como esta adaptação pode ser implementada.

A principal motivação para o artigo está na dificuldade de atender restrições temporais quando aplicações devem executar em diferentes plataformas (processadores e sistemas operacionais), as quais apresentam os mais variados níveis de desempenho e ocupação. Embora este seja um problema geral, ele torna-se mais relevante na medida em que aplicações são disseminadas através da Internet, em escala global.

Os mecanismos de adaptação discutidos neste artigo envolvem alterações no comportamento dos objetos em função de informações coletadas sobre o comportamento temporal apresentado pela aplicação. A reflexão computacional aparece como uma técnica promissora para incorporar tais mecanismos na construção de aplicações com ênfase na adaptabilidade temporal. Embora a necessidade de projetar métodos flexíveis aumente o custo do desenvolvimento, a automatização do ajuste fino do software, com respeito ao comportamento temporal, diminui este mesmo custo. É importante destacar que nem todos os objetos da aplicação precisam empregar reflexão. Mesmo nos objetos que empregam reflexão, nem todos os métodos precisam apresentar alternativas de comportamento. A flexibilização de alguns métodos em alguns objetos cruciais da aplicação já será capaz de permitir um certo nível de adaptação.

Na maioria dos projetos de software os aspectos temporais da aplicação são ignorados nas fases iniciais do desenvolvimento e considerados apenas depois que o código já está escrito e os aspectos funcionais depurados. O resultado desta atitude é a necessidade de empregar remendos de última hora com o objetivo de tornar a aplicação aceitável para os usuários, no que diz respeito aos seus aspectos temporais. Implícita na abordagem apresentada por este artigo está a idéia de que os aspectos temporais devem ser considerados desde as fases iniciais de especificação e projeto. Desta forma, os requisitos temporais da aplicação serão melhor entendidos e poderão ser atendidos com menor custo de desenvolvimento.

Referências

[1] T. F. Abdelzaher, E. M. Atkins, K. G. Shin. QoS Negotiation in Real-Time Systems and Its Application to Automated Flight Control. IEEE Real-Time Applications Symposium, Montreal, Canada, 1997.

[2] A. Burns, A. Wellings. Real-Time Systems and Programming Languages. Addison-Wesley, 2nd edition, 1997.

[3] J. Delacroix. Towards a Stable Earliest Deadline Scheduling Algorithm. Real-Time Systems, vol . 10, pp. 263-291, 1996.

[4] O. Furtado. RTR - Uma Abordagem Reflexiva para Programação de Aplicações Tempo Real. Tese de doutorado, LCMI-DAS, Univ. Federal de Santa Catarina, 1997.

[5] Y. Honda, M. Tokoro. Reflection and Time-Dependent Computing: Experiences with the R2 Architecture. Sony Comp. Science Lab., Tokio, Japan, july 1994.

[6] E. D. Jensen, C. D. Locke, H. Tokuda. A Time-Driven Scheduling Model for Real-Time Operating Systems. Proceedings of the IEEE Real-Time Systems Symposium, pp.112-122, december 1985.

[7] J. Karlsson, K. Ryan. A Cost-Value Approach for Prioritizing Requirements. IEEE Software, pp.67-74, september/october 1997.

[8] G. Lawton. In Search of Real-Time Internet Service. IEEE Computer, vol. 30, no. 11, pp.14-16, november 1997.

[9] J. W.-S. Liu, W.-K. Shih, K.-J. Lin, R. Bettati, J.-Y. Chung. Imprecise Computations. Proceedings of the IEEE, Vol. 82, No. 1, pp. 68-82, january 1994.

[10] P. Maes. Concepts and Experiments in Computational Reflection. Proc. of OOPSLA'87, pp. 147-155, october 1987.

[11] S. E. Mitchell, A. Burns, A. J. Wellings. Developing a Real-Time Metaobject Protocol. WORDS'97, Newport Beach, California, USA, february 5-7, 1997.

[12] S. Natarajan (editor). Imprecise and Approximate Computation. Kluwer Academic Publishers, 177 pages, 1995.

[13] D. Nishida, O. Furtado, J. Fraga, J-M. Farines. Um Protótipo do Modelo Reflexivo Tempo Real RTR sobre a Linguagem Java. XXIII Conferência Latinoamericana de Informática, Vol. 2, pp. 589-598, Valparaiso - Chile, novembro de 1997.

[14] D. I. Rosu, K. Schwan, S. Yalamanchili, R. Jha. On Adaptive Resource Allocation for Complex Real-Time Applications. Proceedings of the 18th IEEE Real-Time Systems Symposium, december 1997.

[15] F. Siqueira, O. Furtado, J. Fraga, J-M. Farines. Implementação Distribuída de um Modelo Reflexivo Tempo Real. I Workshop de Sistemas Distribuídos – WOSID, Salvador- BA, maio de 1996.

[16] J. A. Stankovic, et al. Strategic Directions in Real Time and Embedded Systems. ACM Computer Surveys, Vol. 28, no. 4, pp. 751-763, december 1996.