Mecanismos de Adaptação para Aplicações Tempo Real na Internet

Rômulo Silva Oliveira

Instituto de Informática - Univ. Fed. do Rio Grande do Sul
Caixa Postal 15064, Porto Alegre-RS, Cep 91501-970
romulo@inf.ufrgs.br

Resumo

Nos últimos anos a Internet vem sendo usada para disseminar aplicações na forma de *applets* Java ou componentes Active-X. Também aplicações distribuídas usam a Internet como meio de comunicação. Entre as aplicações a serem disseminadas ou distribuídas estão algumas que incluem restrições de tempo real, tais como aplicações que lidam com áudio e vídeo, ferramentas para trabalho cooperativo e vídeo *games*. Estas aplicações vão encontrar, como plataforma de execução, os mais diferentes processadores e sistemas operacionais. Este artigo identifica mecanismos que permitem a adaptação de uma aplicação tempo real ao seu ambiente de execução, quando disseminada através da Internet. Também são discutidos aspectos sobre como uma aplicação tempo real pode ser especificada e projetada para prover tal adaptação.

Abstract

In recent years the Internet has been used to disseminate applications as Java applets or Active-X components. Distributed applications also use the Internet as its communication medium. Some of these applications supposed to be disseminated or distributed present real-time requirements, such as applications that deal with audio and video, tools for cooperative work and videogames. These applications will have, as their execution environment, many different processors and operating systems. This paper identify mechanisms that allow a real-time application to adapt itself to its executon environment, when disseminated in the Internet. It also discusses aspects related to how a real-time application can be specified and designed to provide such adaptation.

1. Introdução

Sistemas computacionais de tempo real são identificados 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. Em outras palavras, os aspectos temporais não estão limitados a uma questão de maior ou menor desempenho, mas estão diretamente associados com a 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 o sistema inicia sua execução já deve estar garantido que todas as tarefas críticas serão capazes de executar corretamente do ponto de vista temporal. 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 uma série de fatores. Entre eles está a dificuldade de calcular o tempo máximo de computação dos componentes da aplicação. 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 no comportamento de pior caso. Mecanismos como execução de instruções em *pipelines*, memória *cache* e acesso direto a memória (*direct memory access - dma*) dificultam a análise. O mesmo pode ser dito dos sistemas operacionais mais utilizados.

É possível desenvolver arquiteturas especiais que facilitam a análise de pior caso do comportamento da aplicação. Entretanto, tais arquiteturas especiais não são economicamente atrativas e limitam o uso da aplicação às instalações que dispõem de tal arquitetura. O uso de arquiteturas especiais na prática está limitado aos sistemas que são críticos a ponto de justificar o custo. Por exemplo, sondas espaciais, controle de usinas nucleares e sistemas de defesa.

Nos últimos anos a computação foi revolucionada pela Internet e, mais recentemente, pelo World-Wide Web (WWW). Entre as inúmeras possibilidades abertas com a introdução destas tecnologias é possível destacar:

- O emprego da Internet e do WWW para disseminar aplicações na forma de *applets* Java ou componentes Active-X;

- A construção de aplicações distribuídas que usam a Internet como meio de comunicação entre os seus componentes.

Entre as aplicações a serem disseminadas ou distribuídas 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 (como vídeo *games*).

A forma como a Internet opera atualmente não suporta, de uma maneira geral, aplicações tempo real críticas [LAW 97]. 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. É possível imaginar a dificuldade de construir uma aplicação tempo real capaz de apresentar um comportamento temporal aceitável tanto em uma estação de trabalho executando Unix quanto em um microprocessador Intel 486 executando Microsoft Windows. As aplicações distribuídas enfrentam também o problema da enorme variabilidade nos atrasos associados com o envio de mensagens através da Internet.

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. Também ocorre quando um mesmo sistema operacional suporta várias aplicações tempo real simultaneamente e variações no comportamento de uma aplicação interferem na quantidade de recursos disponíveis para as outras aplicações. 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 nada pode ser suposto a respeito do sistema operacional alvo. A adaptação deve ser provida exclusivamente pela aplicação.

O objetivo deste artigo é identificar mecanismos disponíveis para permitir a adaptação de uma aplicação tempo real ao seu ambiente de execução, quando disseminada através da Internet. Também são discutidos aspectos sobre como uma aplicação tempo real pode ser especificada e projetada para prover tal adaptação. O restante do artigo está organizado da seguinte forma: a seção 2 detalha o problema a ser abordado. Na seção 3 são apresentados mecanismos de adaptação para aplicações tempo real que podem ser empregados em aplicações disseminadas pela Internet. A seção 4 discute a necessidade de introduzir o conceito de adaptação na especificação e no projeto da aplicação e como isto pode ser feito. As considerações finais aparecem na seção 5.

2. Cenários para a Adaptação de Aplicações Tempo Real

Mesmo em sistemas computacionais construidos especificamente para a execução de aplicações tempo real existem amplas oportunidades para adaptação. Isto pode ocorrer

porque a carga representada pela aplicação em questão não possui um limite conhecido que possa ser garantido em projeto. Mesmo quando a carga possui uma demanda por recursos bem caracterizada, pode não ser economicamente viável garantir o seu comportamento em um cenário de pior caso. Quando o sistema operacional suporta a execução simultânea de várias aplicações tempo real é necessário negociar a qualidade do serviço fornecido a cada uma delas.

A literatura de tempo real é farta em propostas onde componentes da aplicação e o suporte de execução negociam a qualidade do serviço a ser fornecido por este último. Por exemplo, propostas recentes podem ser encontradas em [ABD 97] e [RAJ 97]. Uma excelente taxonomia sobre adaptação em sistemas multimídia distribuídos aparece em [GEC 97]. Naquele artigo são discutidos aspectos sistêmicos, onde é dada ênfase às aplicações complexas, às redes heterogêneas e à computação móvel.

Embora a maior parte da literatura de tempo real trate de adaptação através da negociação da qualidade de serviço, a adaptação também pode ocorrer através de uma ação unilateral. No caso de uma adaptação unilateral existem quatro cenários:

Cenário A - Uma variação no comportamento da aplicação gera como resposta uma adaptação do suporte. Por exemplo, a redução no período de uma tarefa da aplicação é detectada pelo suporte que, em resposta, aumenta o percentual do tempo de processador reservado para esta tarefa.

Cenário B - Uma variação no comportamento do suporte gera como resposta uma adaptação do próprio suporte. Por exemplo, em um sistema distribuído a falha de um processador pode ser compensada pela redistribuição da carga.

Cenário C - Uma variação na aplicação gera como resposta uma adaptação da própria aplicação. Em sistemas onde a aplicação pode reservar recursos cabe a própria aplicação administrar os recursos reservados. Uma alteração no ambiente poderá exigir uma alteração no modo de operação (*mode change*) da aplicação e uma redistribuição interna dos recursos previamente reservados.

Cenário D - Uma variação no comportamento do suporte gera como resposta uma adaptação da aplicação. Esta é a situação encontrada na Internet. Variações no atraso associado com o envio de mensagens através da Internet devem ser tratadas pela própria aplicação, pois os suportes de execução existentes atualmente não são capazes de isolar a aplicação de tais variações. Mais importante ainda, 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. Atualmente a maioria dos programas ignora este problema e não oferece qualquer tipo de mecanismo de adaptação. Neste caso, a adaptação fica por conta do usuário, sujeito ao desempenho de uma aplicação executando em uma plataforma completamente diferente daquela para qual foi projetada.

Este artigo trata de mecanismos para a adaptação de aplicações tempo real na Internet. Apenas o cenário **D** será considerado. No restante deste artigo será 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, executando sobre o mesmo suporte, dividem os recursos existentes. Logo, existe a necessidade de uma adaptação contínua e não somente durante a etapa de inicialização.

3. Mecanismos de Adaptação para a Internet

Nesta seção serão examinadas técnicas de adaptação descritas na literatura e que podem ser empregadas no contexto descrito na seção anterior. Neste artigo, qualidade de serviço significa o tempo de processador que cada aplicação recebe ou o atraso que ela observa nas mensagens que envia e recebe. 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.

3.1 Flexibilização do Deadline

Embora deadlines não possam receber uma garantia determinista em aplicações disseminadas ou distribuídas via Internet, o conceito "deadline" é útil por duas razões:

- Uma aplicação pode usar os deadlines de suas tarefas para definir as prioridades dos seus fluxos de execução. Embora os sistemas operacionais em geral não suportem o conceito de deadline, a grande maioria deles suporta o conceito de prioridades. Por exemplo, é possível alterar a prioridade de uma *thread* na linguagem Java [LEA 97] com o método `Thread.setPriority()`. Desta forma, a aplicação pode usar os deadlines das tarefas como ponto de partida para a definição das respectivas prioridades. As políticas mais conhecidas neste sentido são o EDF (*Earliest-Deadline-First* [LIU 73]), o Taxa Monotônica (*Rate Monotonic* [LIU 73]) e o Deadline Monotônico (*Deadline Monotonic* [AUD 93]).

- A aplicação pode 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* [BUR 97]), isto é, não existe benefício em executar uma tarefa após o seu deadline. Uma variação deste modelo, mais apropriada para fluxos contínuos de dados (*streams*), é descrita em [HAM 97] com o nome de deadline *(m,k)-firm*. Neste modelo, cada fluxo de dados é implementado através de uma tarefa repetitiva de tal forma que devem ser concluidas dentro do deadline pelo menos **m** ativações em qualquer conjunto de **k** ativações consecutivas. Caso este modelo seja empregado, a taxa de deadlines perdidos deve ser adaptada para considerar os valores de **m** e **k** associados com cada fluxo de dados.

A forma mais simples e freqüente de adaptação é simplesmente relaxar o conceito de deadline. Um dos primeiros trabalhos seguindo esta abordagem aparece em [JEN 85], 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*). Desta forma, o conceito tradicional de deadline é uma simplificação desta visão mais ampla. O conceito de função tempo-valor também é utilizado nos trabalhos apresentados em [CHE 91], [JEN 93] e [CHE 96].

Neste mecanismo a adaptação ocorre de forma implícita, no momento em que os deadlines não são respeitados e as tarefas são concluídas com atraso. A vantagem desta adaptação é que os projetistas e programadores não precisam incluir código adicional para implementar o mecanismo. A desvantagem está no fato desta adaptação ser mais teórica do que prática, pois ela reside simplesmente na aceitação dos atrasos. Qualquer aplicação que ignore os aspectos temporais está, de certa forma, implementando este mecanismo. Entretanto, mesmo quando os aspectos temporais são devidamente analisados, existem situações onde a melhor adaptação é, simplesmente, atrasar a conclusão da tarefa. Além disto, o conhecimento dos deadlines e de suas respectivas importâncias permite uma degradação mais suave do que quando deadlines são perdidos de forma aleatória.

3.2 Flexibilização do Tempo de Execução

A dificuldade encontrada no escalonamento tempo real levou alguns autores a proporem uma abordagem do tipo "fazer o trabalho possível dentro do tempo disponível". Esta técnica, conhecida pelo nome de Computação Imprecisa (*Imprecise Computation* [LIU 94]), flexibiliza o escalonamento tempo real na medida em que sacrifica a qualidade dos resultados para poder cumprir os prazos exigidos. Uma descrição detalhada dos diversos aspectos desta técnica pode ser encontrada no livro [NAT 95].

Computação Imprecisa está fundamentada na idéia de que cada tarefa da aplicação possui 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 decompô-la em parte obrigatória e parte opcional. É importante observar que algumas tarefas podem possuir apenas parte obrigatória ou apenas parte opcional. A técnica não obriga que exatamente cada tarefa da aplicação possua as duas partes.

Em situações normais, tanto a parte obrigatória quanto a parte opcional são executadas e a qualidade máxima é obtida. Em situações de sobrecarga algumas partes opcionais são deixadas de lado. Este mecanismo permite uma degradação controlada do sistema, na medida em que pode-se determinar o que não será executado em caso de sobrecarga. Na Computação Imprecisa o tempo de computação das tarefas é negociado a partir da introdução do conceito de qualidade do resultado. A redução da qualidade do resultado produzido gera uma redução na demanda pelos recursos do sistema que permite o melhor atendimento dos deadlines. 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 do conceito de Computação Imprecisa. Em [HUA 95] Computação Imprecisa é aplicada à transmissão de imagens e vídeo. Em [DEY 96] são feitas referências a diversas aplicações descritas na literatura que empregam o conceito de Computação Imprecisa ou uma variação deste. Em [GAR 94] é feito um levantamento sobre o emprego da inteligência artificial em aplicações tempo real e Computação Imprecisa é apresentada como um mecanismo que viabiliza a implementação de soluções baseadas em inteligência artificial submetidas a restrições temporais.

Em sistemas tempo real críticos (*hard real-time*) as partes obrigatórias são garantidas através de análise de escalonabilidade em tempo de projeto (*off-line*). Quando o sistema inicia sua execução já está garantido que todas as tarefas serão capazes de executar, ao menos, suas partes obrigatórias. Esta abordagem não é possível no contexto deste trabalho, pois atualmente não é possível garantir em projeto o comportamento das tarefas na Internet. Quando os requisitos temporais não são críticos (*soft real-time*) é possível que determinadas tarefas não consigam executar sequer suas respectivas 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 a qualidade de seus resultados dinamicamente, conforme a disponibilidade dos recursos computacionais.

Existem três formas básicas de programar tarefas imprecisas: funções monotônicas, etapas de melhoramento e múltiplas versões.

As funções monotônicas (*monotone functions*) são aquelas cuja qualidade do resultado aumenta (ou pelo menos não diminui) na medida em que o tempo de execução da função aumenta. As computações necessárias para obter-se um nível mínimo de qualidade correspondem à parte obrigatória. Qualquer computação além desta refina progressivamente o resultado da tarefa e é tratada como parte opcional. É a forma de programação que fornece maior flexibilidade do ponto de vista do escalonamento, pois qualquer tempo de processador disponível a mais reflete na qualidade do resultado. Algoritmos deste tipo podem ser encontrados nas áreas de cálculo numérico, estimativa probabilista e pesquisa heurística. Por exemplo, algoritmos de inteligência artificial utilizados em vídeo *games* podem ser programados desta forma.

Etapas de melhoramento (*sieve functions*) são aquelas cuja finalidade é produzir saídas no mínimo tão precisas quanto as correspondentes entradas. O valor de entrada é melhorado de alguma forma. Este tipo de etapa corresponde a uma função que pode ser omitida completamente na execução de uma tarefa. Tipicamente, não existe benefício em executar uma etapa de melhoramento parcialmente. Por exemplo, algoritmos de processamento de imagens são capazes de produzir imagens razoáveis mesmo se algumas etapas forem descartadas. Também os cenários de fundo em jogos podem ser mais ou menos detalhados, dependendo do desempenho do sistema onde a aplicação executa.

Uma tarefa imprecisa também pode ser implementada através de múltiplas versões (*multiple versions*). Normalmente são empregadas duas versões. A versão primária gera um resultado preciso, porém possui um tempo de execução maior. A versão secundária gera um resultado impreciso em um tempo de execução menor. A cada ativação da tarefa é necessário escolher qual versão será executada. Esta técnica é a mais flexível do ponto de vista da programação, pois as duas versões podem empregar algoritmos completamente diferentes. Também é possível considerar a versão primária como o algoritmo desejado e a versão secundária como um tratador de exceção temporal associado com o algoritmo.

Um esquema semelhante à Computação Imprecisa com múltiplas versões é descrito em [GER 96] e [STR 97] com o nome de *TaskPair* e empregado na construção de aplicações distribuídas de tempo real. A única diferença está no fato do *TaskPair* sempre iniciar executando a versão principal. Caso a versão principal não seja concluída até um certo tempo antes do deadline ela é abortada e o tratador de exceção executado completamente. A vantagem do *TaskPair* está em conseguir lidar melhor com algoritmos cujo tempo de execução é muito variável. A desvantagem está no desperdício de tempo do processador sempre que a versão principal é iniciada e abortada sem gerar resultados.

A forma de programação empregada interfere no tipo de decisão de escalonamento a ser tomada. Quando uma função monotônica é empregada qualquer tempo a mais de processador fornecido para a tarefa resulta em uma (pequena) melhora na qualidade do resultado. Quando a parte opcional executa uma etapa de melhoramento, não existe benefício em executa-la parcialmente. Assim, é necessário decidir se a etapa de

melhoramento será executada completamente ou descartada. Esta característica é chamada na literatura de restrição 0/1 (*0/1 constraint*). De forma semelhante, multiplas versões criam uma restrição 0/1. O emprego de múltiplas versões obriga uma escolha entre a versão primária e a versão secundária. Obviamente esta escolha deverá ser feita antes da tarefa iniciar sua execução.

No contexto das aplicações que utilizam a Internet, a flexibilização do tempo de execução parece ser o 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 ou conforme os atrasos observados na rede enquanto 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.

3.3 Flexibilização do Período

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 vídeo *game*. 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 a aplicação 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. Uma transmissão de áudio em tempo real é capaz de apresentar melhor qualidade quando pedaços menores são transmitidos com maior frequência. Em [SET 96] é apresentado um algoritmo para escolher, em tempo de projeto (*off-line*), os períodos das tarefas em função de curvas do tipo "índice de desempenho versus período da tarefa". O objetivo do artigo é usar o algoritmo na síntese de controladores, isto é, em sistemas tempo real *hard*. Entretanto, a abordagem ilustra como o período das tarefas pode ser manipulado para adaptar a aplicação aos recursos existentes.

Existem trabalhos na literatura que analisam a situação onde os períodos das tarefas podem ser alterados em tempo de execução (*on-line*). Em [KUO 97] é definido o conceito de processo periódico ajustável (*adjustable periodic process*), o qual é caracterizado por um conjunto de pares (tempo de computação, período). Cada um destes pares corresponde a uma opcão para o comportamento do processo em questão. A adaptação ocorre através da seleção de um dos pares oferecidos para cada um dos processos, de maneira a tornar o sistema escalonável. Esta seleção é feita em tempo de execução e pode ser revista quando necessário.

De certa forma este mecanismo já está presente em muitas aplicações onde as tarefas são ativadas em sequência a partir de um laço principal (executivo cíclico) e este laço principal é reiniciado tão logo a última tarefa seja concluida. Neste esquema as tarefas são executadas periodicamente, sendo o período de cada tarefa determinado pelo desempenho do suporte e pela necessidade de recursos das demais tarefas. Embora possível, esta implementação do mecanismo possui como grande desvantagem a total falta de controle sobre os períodos das tarefas. Por exemplo, pode não haver ganho em executar uma determinada tarefa com um período menor que certo limiar. Ao fazer isto, a aplicação estará desperdiçando recursos que poderiam ser utilizados para aumentar a qualidade de outras tarefas.

3.4 Flexibilização da Execução

Uma forma mais radical de flexibilização é simplesmente não executar algumas tarefas quando o desempenho estiver abaixo do desejado. Este mecanismo pode ser considerado um caso extremo dos anteriores, quando o deadline é relaxado a tal ponto que a tarefa somente será concluida em um tempo infinito, quando a qualidade é sacrificada a ponto do tempo de execução da tarefa chegar a zero ou, ainda, quando a tarefa passa a ser executada com período infinito. No caso de aplicações com tarefas repetitivas (periódicas ou esporádicas), 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. Por exemplo, em animações compostas por diversos elementos independentes, alguns elementos podem ser excluídos da animação. Em [DEL 96] 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.

Em certos fluxos contínuos de dados implementados por tarefa periódica algumas instâncias da tarefa podem ser descartadas sem prejuízo para a aplicação. Neste sentido, em [KOR 95] é proposto um modelo composto por tarefas periódicas onde ativações podem ser ocasionalmente descartadas (*skipped*), desde que a distância temporal entre dois descartes seja, no mínimo, igual a um parâmetro **s** (*skip parameter*) definido para cada tarefa. A aplicação somente apresenta uma redução sensível de qualidade quando o parâmetro **s** não é respeitado para alguma tarefa. Os autores sugerem o emprego deste modelo para tarefas que manipulam sinais de radar ou realizam apresentação de vídeo.

3.5 Flexibilização do Nível 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. Esta

organização do algoritmo permite sua adaptação a diferentes níveis de paralelismo, isto é, ele é capaz de adaptar-se para tirar proveito do paralelismo real quando este existir.

Por exemplo, no livro [AND 91] uma abordagem para a programação concorrente chamada de "trabalhadores replicados" (*replicated workers*) consiste em um conjunto de *threads* e um conjunto de trabalhos. Inicialmente o conjunto de trabalhos é composto por um único trabalho, isto é, o trabalho total a ser feito. Cada *thread* executa um laço onde retira um trabalho do conjunto, divide o trabalho em subtrabalhos, executa ou devolve alguns subtrabalhos ao conjunto. A execução termina quando o conjunto de trabalhos estiver vazio e nenhuma *thread* estiver executando. Como exemplo, a abordagem é usada em integração numérica, geração de números primos e problema do caixeiro viajante.

Em [ROS 97] é descrito um modelo para aplicações de tempo real 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 suas necessidades de recursos durante a execução, inclusive, o seu nível de paralelismo.

Atualmente ainda são poucos os sistemas que suportam paralelismo real. Entretanto, a tendência é que este número cresça. Computadores com dois ou quatro processadores já são encontrados com certa frequência. 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.

4. Especificação e Projeto Visando Adaptabilidade Temporal

Nesta seção serão discutidos aspectos ligados a especificação e ao projeto de software visando incluir um certo grau de adaptabilidade temporal. A definição de uma metodologia detalhada está além do escopo deste trabalho. O objetivo desta seção é descrever alguns elementos chave a serem considerados na alteração das metodologias existentes para o desenvolvimento de software adaptativo e indicar como tais alterações poderiam ser feitas.

4.1 Níveis de Qualidade

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. É importante que a especificação inclua quais propriedades podem ser sacrificadas quando o suporte não fornecer a quantidade de recursos necessária para executar a aplicação completa, com qualidade máxima.

Uma aplicação pode oferecer diversas oportunidades para adaptação. Por exemplo, algumas funcionalidades secundárias podem ser eliminadas, outras podem ter a qualidade reduzida em função do emprego de algoritmos mais simples, do menor processamento de dados ou ainda de uma menor frequência de execução. Durante a execução é necessário decidir quais oportunidades para adaptação serão efetivamente usadas. Em outras palavras, quais propriedades da aplicação são menos importantes e devem ser sacrificadas antes.

Por exemplo, o método descrito em [KAR 97] 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. Ela é usada para identificar inconsistências na especificação dos valores através do cálculo de um índice de consistência (*consistency index*). Isto torna o processo menos sensível a erros de julgamento. Após alguns cálculos para 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.

Em muitas propostas encontradas na literatura é suposto que funções contínuas definem a utilidade ou o valor da aplicação a partir das propriedades exibidas. Propostas que utilizam funções contínuas de valor são difíceis de construir na prática. Freqüentemente não existe na especificação uma precisão tal que permita a definição de equações matemáticas para descrever a utilidade da aplicação. Se, durante o projeto, são criadas funções matemáticas precisas a partir de uma especificação superficial, isto significa que escolhas arbitrárias foram feitas, impondo ao projeto restrições que não estavam presentes na especificação original.

Aplicações definidas em termos de níveis de qualidade são mais simples de especificar e projetar. A nível de projeto não é necessário criar equações matemáticas de utilidade em função das propriedades exibidas pela aplicação. São usados diretamente os níveis de qualidade presentes na especificação. 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 [DEL 96] 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 e não apenas ativações individuais. Estes modos devem ser definidos pelo projetista para refletir a semântica da aplicação, isto é, sua especificação. Em [ABD 97] 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.

4.2 Monitoração

No contexto deste trabalho, a adaptação é originada e executada pela própria aplicação. O suporte (sistema operacional, rede de comunicação) ignora este comportamento. Existe portanto a necessidade da aplicação monitorar 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 como indesejada, na medida em que o comportamento resultante não é satisfatório e exige uma redução no nível de qualidade;
- Identificação da situação como favorável, na medida em que 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, se for o caso, do novo nível de qualidade desejado;
- Comunicação, se for o caso, aos componentes da aplicação o novo comportamento selecionado.

Uma forma simples de realizar monitoração é comparar os deadlines das tarefas com o respectivo tempo de resposta medido. Sempre que a diferença média entre os dois passar de um limiar, a qualidade da aplicação é alterada para mais ou para menos. Por exemplo, podem ser usados gatilhos incluidos na própria aplicação. Para cada nível de qualidade possível, existe um conjunto de gatilhos. Quando um dos gatilhos associados com o nível de qualidade corrente é acionado, ocorre uma mudança (aumento ou redução) do nível de qualidade da aplicação. Após esta mudança, um novo conjunto de gatilhos passa a ser utilizado. Como a detecção é feita pela própria aplicação, é possível aproveitar o conhecimento semântico que ela tem do seu estado. Este conhecimento semântico permite uma detecção superior ao que seria conseguido, por exemplo, por um sistema operacional que ignorasse completamente o significado das restrições temporais da aplicação.

Em [ROS 97] são descritos dois algoritmos para detectar mudanças nos níveis de qualidade medidos de uma aplicação. Embora os algoritmos sejam propostos para um contexto onde o suporte é responsável por realizar a adaptação, eles podem ser adaptados para um ambiente onde a própria aplicação detecta a necessidade de mudanças e as realiza. Os detetores foram baseados em limiar (*threshold-driven*) e em variação (*variation-driven*). As experiências mostraram que detecção baseada em limiar é mais rápida, porém mais sensível ao ruído (quando o limiar é ultrapassado apenas por um breve momento). Detecção baseada em variação é mais confiável, no sentido de ser menos

sensível ao ruído, porém mais lenta. Ela detecta variações no comportamento da aplicação que não violam imediatamente o limiar, mas aumentam as chances disto acontecer. Por outro lado, uma variação lenta nas medidas pode fazer com que o limiar aceitável seja ultrapassado sem que uma adaptação seja disparada.

Existem na literatura propostas de escalonamento tempo real que exigem a detecção de sobrecarga. Por exemplo, em [DEL 96] é proposta uma solução que utiliza as propriedades temporais médias das tarefas no lugar de uma análise de pior caso. Cada tarefa é caracterizada por um tempo médio de execução, um período médio, um deadline e por uma medida de sua importância. Em situações normais as tarefas são escalonadas segundo EDF (*Earliest Deadline First* [LIU 73]). Quando uma sobrecarga (*overload*) é detectada, a execução das tarefas menos importantes é cancelada para diminuir a carga no sistema, isto é, adaptação através de flexibilização da execução. As tarefas remanescentes são novamente escalonadas segundo EDF. A detecção de sobrecarga é feita através de um algoritmo que considera o tempo médio de execução das tarefas, seus deadlines e supõe EDF como o escalonador empregado pelo sistema. Infelizmente esta solução está limitada a sistemas que executam uma única aplicação e cujo suporte utiliza EDF.

Uma forma alternativa é permitir que o próprio usuário defina qual nível de qualidade ele deseja, através de uma lista com alguns níveis estipulados ainda na especificação. Em [VOG 85] é discutido um método (*Quality Query by Example*) que utiliza uma lista de exemplos com qualidades diferentes para o usuário escolher. Por exemplo, imagens com diferentes tamanhos e resoluções ou sons com qualidade de telefone ou CD. É importante neste caso que os exemplos retratem a capacidade ou não da aplicação em executá-lo. Por exemplo, uma imagem maior poderá implicar em uma taxa menor na exibição dos quadros, em função do desempenho possível no suporte em questão.

Esta seleção do nível de qualidade feita pela usuário é mais apropriada para decisões de longo prazo (válidas por parcela considerável da sessão). Ajustes rápidos de curto prazo (válidos por alguns segundos), em função de oscilações na quantidade de recursos disponibilizados pelo suporte, devem ser feitos pela própria aplicação.

4.3 Alteração do Comportamento Durante a Execução

Alguns problemas importantes podem aparecer no momento em que uma aplicação altera o seu comportamento durante a execução. Por exemplo, no momento em que a decisão de alterar o comportamento é tomada, algoritmos associados com o nível de qualidade a ser abandonado podem estar no meio de sua execução. Simplesmente abortar estes algoritmos em execução poderá ser pior do que esperar sua conclusão, mesmo que isto signifique atrasar a adaptação do comportamento. O mesmo acontece com variações no período. É importante projetar as mudanças no comportamento da aplicação de tal forma que ela seja suave, sem mudanças abruptas que gerariam desconforto para o usuáiro.

Se o algoritmo que decide por uma mudança no comportamento da aplicação for muito sensível, podem ocorrer instabilidades no sistema. Por exemplo, o algoritmo detecta uma certa folga nos deadlines e solicita um aumento na qualidade da aplicação. Este aumento na qualidade provoca um imediato aumento na carga do sistema e deadlines começam a ser perdidos. O mesmo algoritmo decide então diminuir o nível de qualidade e volta a situação original, entrando em um ciclo infinito. Dependendo de como a mudança no comportamento da aplicação for feita, esta alteração constante do nível de qualidade pode gerar desconforto para o usuário ou até mesmo paralisar a aplicação. A paralisação da aplicação acontece quando ela fica todo o tempo realizando adaptações sem jamais executar as funções para as quais foi realmente construida. Esta é uma situação semelhante ao *thrashing*, possível em sistemas de memória virtual.

Se várias aplicações compartilham o mesmo suporte (computador, sistema operacional) existe uma disputa por recursos entre elas. Suponha duas aplicações **A** e **B** que procuram adaptar-se ao suporte de execução. Suponha agora que a aplicação **A** percebe que seus deadlines são perdidos e diminui o seu nível de qualidade. Esta redução diminui a carga no sistema, provendo uma folga maior para as duas aplicações. A aplicação **B** poderá entender isto como uma capacidade do suporte não aproveitada e aumentar o seu nível de qualidade. Com o aumento do nível de qualidade de **B**, a aplicação **A** volta a perder deadlines e reduz ainda mais o seu nível de qualidade. Este ciclo prossegue até que a aplicação **A** se declara incapaz de executar por falta de recursos enquanto a aplicação **B** apresenta sua qualidade máxima. Como lidar com este cenário ainda é uma questão a ser pesquisada.

4.4 Reflexão Computacional

Em [STA 95] é discutido como a "reflexão computacional" (*reflection*) pode ser empregada para introduzir flexibilidade em sistemas de tempo real complexos. Um sistema é considerado reflexivo quando ele raciocina sobre o seu estado corrente e o do ambiente e, em função disto, atua sobre seu próprio comportamento. As políticas e os parâmetros que definem o comportamento da aplicação formam um meta-nível (*meta-level*) onde a reflexão acontece. Informações sobre o estado da aplicação são mantidas e usadas para tomar decisões a respeito de seu comportamento, isto é, a respeito das políticas e dos parâmetros em vigor. Segundo [STA 95] a reflexão computacional permite uma reação mais flexível às alterações que venham a ocorrer na dinâmica da aplicação e do ambiente.

Os mecanismos de adaptação discutidos na seção 3 envolvem alterações no comportamento da aplicação em função de informações coletadas sobre o estado atual da aplicação, do ambiente e do suporte de execução. A reflexão computacional parece ser o caminho ideal para incorporar tais mecanismos na construção de aplicações com ênfase na adaptabilidade temporal. Alguns passos já estão sendo dados nesta direção. Por exemplo, uma proposta de extensão para a linguagem Java onde reflexão é usada para controlar os

aspectos temporais da aplicação aparece em [FUR 96]. Embora o aspecto adaptação não seja abordado naquele artigo exatamente como foi apresentado aqui, o mecanismo de reflexão descrito em [FUR 96] para a linguagem Java pode ser utilizado na implementação dos mecanismos de adaptação descritos na seção 3.

5. Conclusões

Neste artigo foram identificados mecanismos para permitir a adaptação de uma aplicação tempo real ao seu ambiente de execução, quando disseminada ou distribuída através da Internet. Cinco mecanismos foram descritos, baseados na flexibilização dos deadlines, dos tempos de execução, dos períodos, da execução e do nível de paralelismo. Também foram discutidos aspectos importantes da especificação e projeto de aplicações visando adaptabilidade temporal.

Na maioria das vezes os aspectos temporais de uma aplicação são ignorados nas fases iniciais do desenvolvimento e considerados apenas depois que o código já está escrito. O resultado desta postura é a necessidade de empregar remendos de última hora com o objetivo de tornar a aplicação aceitável para os usuários. Embutido no trabalho apresentado neste artigo está a idéia de que os aspectos temporais das aplicações devem ser considerados desde o primeiro momento, isto é, desde a especificação e o projeto. Desta forma, seus requisitos temporais poderão ser melhor atendidos com um menor custo de desenvolvimento.

Uma questão não abordada neste artigo é a descrição e avaliação de algoritmos apropriados para controlar aplicações que empregam os mecanismos aqui descritos. O objetivo de tais algoritmos seria selecionar, para cada componente a cada momento, o nível de qualidade a ser buscado. Como a abordagem é do tipo melhor esforço (*best-effort*) por natureza, a avaliação dos algoritmos que vierem a ser propostos será probabilista, feita através de simulações ou de experiências na própria Internet. A questão básica neste tipo de algoritmo é como obter um compromisso entre a qualidade das decisões e o tempo de processamento (*overhead*) necessário para chegar às decisões. Este é um tema para pesquisas futuras.

Agradecimentos

Este trabalho foi parcialmente financiado pela FAPERGS, processo 97/0741.0. O autor agradece a colaboração, na forma de comentários e discussões, dos professores Joni da Silva Fraga (LCMI-DAS-UFSC) e Olinto Furtado (INE-UFSC) e do doutorando Carlos Montez (LCMI-DAS-UFSC).

Referências

[ABD 97] 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.

[AND 91] G. R. Andrews. Concurrent Programming. The Benjamin/Cummings Publishing Company, Inc., 1991.

[AUD 93] N. C. Audsley, A. Burns, M. F. Richardson, K. Tindell, A. J. Wellings. Applying New Scheduling Theory to Static Priority Pre-emptive Scheduling. Software Engineering Journal, vol. 8, no. 5, pp.284-292, 1993.

[BUR 97] A. Burns, A. Wellings. Real-Time Systems and Programming Languages. Addison-Wesley, 2nd edition, 1997.

[CHE 91] K. Chen. A Study on the Timeliness Property in Real-Time Systems. The Journal of Real-Time Systems, vol. 3, pp. 247-273, 1991.

[CHE 96] K. Chen, P. Muhlethaler. A Scheduling Algorithm for Tasks Described by Time Value Function. The Journal of Real-Time Systems, vol. 10, pp. 293-312, 1996.

[DEL 96] J. Delacroix. Towards a Stable Earliest Deadline Scheduling Algorithm. Real-Time Systems, vol . 10, pp. 263-291, 1996.

[DEY 96] J. K. Dey, J. Kurose, D. Towsley. On-Line Scheduling Policies for a Class of IRIS (Increasing Reward with Increasing Service) Real-Time Tasks. IEEE Transactions on Computers, vol. 45, no. 7, pp. 802-813, july 1996.

[FUR 96] O. Furtado, J.-M. Farines. Java/RTR - Uma Linguagem Reflexiva para Programação de Aplicações Tempo Real. Anais do Simpósio Brasileiro de Linguagens de Programação, setembro 1996.

[GAR 94] A. Garvey, V. Lesser. A Survey of Research in Deliberative Real-Time Artificial Intelligence. Real-Time Systems, vol. 6, pp. 317-347, 1994.

[GEC 97] J. Gecsei. Adaptation in Distributed Multimedia Systems. IEEE MultiMedia, pp. 58-66, april-june 1997.

[GER 96] M. Gergeleit, H. Streich. TaskPair-Scheduling with Optimistic Case Execution Times - An Example for an Adaptive Real-Time System. Proceedings of the WORDS, Laguna Beach, California, february 1996.

[HAM 97] M. Hamdaoui, P. Ramanathan. Evaluating Dynamic Failure Probability for Streams with (m, k)-Firm Deadlines. IEEE Transactions on Computers, vol. 46, no. 12, pp. 1325-1337, december 1997.

[HUA 95] X. Huang, A. M. K. Cheng. Applying Imprecise Algorithms to Real-Time Image and Video Transmission. Proceedings of the IEEE Workshop on Real-Time Applications. Chicago, may 1995.

[JEN 85] 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.

[JEN 93] E. D. Jensen. A Timeliness Model for Asynchronous Decentralized Computer Systems. Proc. of the IEEE Real-Time Systems Symposium, pp.173-182, december 1993.

[KAR 97] J. Karlsson, K. Ryan. A Cost-Value Approach for Prioritizing Requirements. IEEE Software, pp.67-74, september/october 1997.

[KOR 95] G. Koren, D. Shasha. Skip-Over: Algorithms and Complexity for Overloaded Systems that Allow Skips. Proceedings of the IEEE Real-Time Systems Symposium, pp. 110-117, december 1995.

[KUO 97] T.-W. Kuo, A. K. Mok. Incremental Reconfiguration and Load Adjustment in Adaptive Real-Time Systems. IEEE Transactions on Computers, vol. 46, no. 12, pp. 1313-1324, december 1997.

[LAW 97] G. Lawton. In Search of Real-Time Internet Service. IEEE Computer, vol. 30, no. 11, pp.14-16, november 1997.

[LEA 97] D. Lea. Concurrent Programming in Java: Design Principles and Patterns. Addison Wesley Longman, 1997.

[LIU 73] C. L. Liu, J. W. Layland. Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment. J. of the ACM, vol. 20, no. 1, pp. 46-61, jan/1973.

[LIU 94] 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. 83-94, january 1994.

[NAT 95] S. Natarajan (editor). Imprecise and Approximate Computation. Kluwer Academic Publishers, 177 pages, 1995.

[RAJ 97] R. Rajkumar, C.Lee, J.Lehoczky, D.Siewiorek. A Resource Allocation Model for QoS Management. Proceedings of the 18th IEEE Real-Time Systems Symposium, december 1997.

[ROS 97] 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.

[SET 96] D. Seto, J. P. Lehoczky, L. Sha, K. G. Shin. On Task Schedulability in Real-Time Control Systems. Proceedings of the IEEE Real-Time Systems Symposium, pp. 13-21, december 1996.

[STA 95] J. A. Stankovic, K. Ramamritham. A Reflective Architecture for Real-Time Operating Systems. Chapter 2 in "Adavances in Real-Time Systems", edited by S. H. Son, Prentice-Hall, 1995.

[STR 97] H. Streich, M. Gergeleit. On the Design of a Dynamic Distributed Real-Time Environment. Proceedings of the WPDRTS, Geneva, Switzerland, april 1997.

[VOG 95] A. Vogel, B. Kerhervé, G. Bochmann, J. Gecsei. Distributed Multimedia and QOS: A Survey. IEEE Multimedia, vol. 2, no. 2, summer/1995.