Adaptação em Aplicações Tempo Real: Um Protótipo Baseado em Computação Imprecisa e Reflexão Computacional

Sandro Luís Schmidtke+, Olinto José Varela Furtado+ e Rômulo Silva de Oliveira*
+ CPGCC – Dep. de Informática e de Estatística * LCMI – Dep. de Automação e Sistemas
Universidade Federal de Santa Catarina
Caixa Postal 476, Florianopolis-SC, 88040-900, Brasil
sandro@inf.ufsc.br, olinto@inf.ufsc.br e romulo@das.ufsc.br

Resumo

Diversas aplicações com restrições de tempo real são disseminadas através da Internet. 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 nessas diferentes plataformas. O objetivo desse artigo é descrever um protótipo onde Computação Imprecisa, através da Reflexão Computacional, é utilizada para adaptação de aplicações tempo real a diferentes plataformas.

Abstract

Many applications with real-time requirements are disseminated through the Internet. 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. The purpose of this paper is to describe a prototype where Imprecise Computation, implemented by using Refletion, can be used to allow the adaptation of real-time applications to different execution environments.

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.

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 descrevem o comportamento desejado. O não atendimento de tais requisitos reduz a utilidade da aplicação mas não resulta em conseqüências catastróficas.

Uma das possibilidades abertas com o emprego da Internet e do WWW (*World-Wide Web*) é a disseminação de aplicações em larga escala. Entre as aplicações disseminadas estão algumas que incluem restrições de tempo real. Por exemplo, aplicações que lidam com áudio

e vídeo, jogos individuais ou coletivos onde o tempo de reação do jogador é importante.

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 [6]. 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 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 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*). 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 [3]. RTR é um modelo de programação reflexivo e de tempo real, que permite a representação e o controle de aspectos temporais de aplicações tempo real que seguem uma abordagem de melhor esforço.

O objetivo deste artigo é descrever um protótipo onde a técnica de Computação Imprecisa, implementada através de Reflexão Computacional, é utilizada para permitir a adaptação de aplicações de tempo real a diferentes plataformas. A seção 2 discute a adaptação em aplicações tempo real; na seção 3 é descrita a Computação Imprecisa; a seção 4 descreve a técnica de Reflexão Computacional e o modelo RTR; na seção 5 é descrito um protótipo que emprega Computação Imprecisa e Reflexão Computacional para a adaptação, sendo o modelo RTR usado para exemplificar a solução proposta; a seção 6 contém as considerações finais.

2. Mecanismos de Adaptação

No caso de uma aplicação tempo real disseminada via Internet nada pode ser suposto a respeito do sistema operacional alvo. Ao 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 resume alguns mecanismos para a adaptação de aplicações tempo real. Uma

melhor descrição pode ser encontrada em [9] e [10]. 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.

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 [5], 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*). 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. Em geral estas tarefas possuem o seu período definido em tempo de projeto. Uma forma de prover adaptabilidade 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 [2] é apresentada uma solução para sobrecargas onde ativações individuais são descartadas e, caso a sobrecarga persista, passam a ser descartadas tarefas completas.

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.

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. 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.

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. A dificuldade encontrada para

atender requisitos temporais gerou a proposta de uma abordagem onde sacrifica-se a qualidade dos resultados para poder cumprir os prazos exigidos. Esta técnica é chamada Computação Imprecisa (*Imprecise Computation*, [6]).

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. 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 através da determinação do que não será executado na sobrecarga.

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.

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 [7], 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". 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 operacional ficam facilitadas.

Embora pouco explorada no domínio tempo real ([3], [4] e [8]), a abordagem de reflexão computacional é vista como uma abordagem promissora no que se refere a estruturação de sistemas tempo real complexos [11], apta a contribuir nas questões de flexibilidade e gerenciamento da complexidade dos sistemas tempo real atuais e futuros.

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 [8], para os quais a análise de pior caso ainda é uma questão de pesquisa em aberto. 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 tempo real *soft* é perfeitamente viável.

4.1 Modelo RTR

O Modelo RTR (Reflexivo Tempo Real) [3] 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.

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, pelo processamento de restrições temporais e pelo controle da execução de 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 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.

5. Protótipo

Visando testar a efetividade do uso de computação imprecisa e reflexão computacional na implementação de mecanismos de adaptação para aplicações Tempo Real (abordagem proposta em [10], com base em [3] e [9]), desenvolveu-se um protótipo para uma aplicação denominada Telejogo, inspirada em um produto da Philips do Brasil popular no Brasil no início dos anos 80. Tal aplicação consiste de vários jogos, dos quais o "jogo fechado" será usado neste artigo para descrever os resultados obtidos.

O protótipo foi desenvolvido em Java e Javaassist (versão 8.0) [1], implementado em um K6-III (500mhz, 128Mb) e testado em microcomputadores com diferentes configurações. A

aplicação foi modelada segundo a abordagem proposta pelo modelo RTR. Como RTR é um modelo de programação abstrato, independente de linguagem de programação, a estrutura e o comportamento da aplicação foram simulados através das potencialidades reflexivas de Javassist. Na implementação realizada o mecanismo de adaptação utilizado foi a variação do tempo de execução das tarefas, enquanto que múltiplas versões foi a forma de programação de computação imprecisa utilizada.

5.1 Computação Imprecisa no Modelo RTR

O suporte à computação imprecisa no modelo RTR envolve tanto a representação quanto o controle da imprecisão, sendo que a representação é tratada no nível base enquanto o controle é tratado no nível meta. A representação de tarefas imprecisas dá-se através da associação de restrições temporais aos métodos dos objetos-base. Para tanto, são definidas restrições temporais específicas para cada forma de programação de computação imprecisa. Quando associadas aos método dos objetos-base, estas restrições temporais fazem com que os mesmos comportem-se como tarefas imprecisas.

O controle das restrições temporais representando as diferentes formas de programação de computação imprecisa é implementado através de métodos (um para cada restrição) nos meta-objetos gerenciadores (MOG). Assim sendo, sempre que o MOG intercepta a ativação de um método ao qual esteja associada uma dessas restrições, o método que implementa a restrição em questão é executado, garantindo a semântica de imprecisão desejada.

A seguir, é mostrado, de forma simplificada, a representação e o controle da implementação de Computação Imprecisa na forma de múltiplas versões no modelo RTR.

Múltiplas Versões – No modelo RTR a noção de múltiplas versões pode, por exemplo, ser representada através da restrição temporal *TimingPolymorphic( )* [3] :

```
(* 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 <MethodList> especifica as diferentes versões desse método.

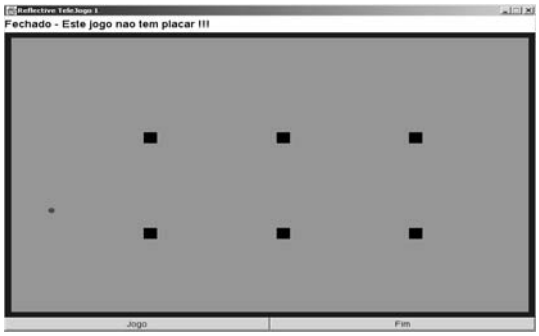
```
(* Associação da restrição temporal a um método do objeto-base *)
void DisplayImagem( ... ), TimingPolymorphic (D, Met1="DI-Versao1",
                                    Met2="DI-Versao2", Met3="DI-Versao3")
begin  ...  end;
(* Implementação das múltiplas versões *)
void DI-Versao1 ( ... ) begin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

void DI-Versao2 ( ... ) begin ... end;  // Apresenta imagem de boa qualidade

void DI-Versao3 ( ... ) begin ... end;    // Apresenta imagem com qualidade minima
(* Ativação da tarefa polimórfica-temporal *)
Objeto.DisplayImagem( ... ),(40);
```

O Controle da restrição "TimingPolymorphic" dá-se através da definição de um método no MOG, 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 pode, por exemplo, ser dependente do nível de qualidade corrente, o qual é determinado pelo MOE a partir de métricas tais como o número de deadlines perdidos e o atraso médio dos métodos que não atenderam o deadline.

5.2 Descrição do protótipo

A aplicação usada no protótipo foi o Jogo Fechado. Neste jogo os jogadores (6 quadrados)

são colocados na janela do jogo e movimentam-se fazendo um vai-e-vem. Quando a bola toca uma borda ou um dos jogadores, sua trajetória é automaticamente desviada.



A implementação deste jogo pode, simplificadamente, ser vista como a ativação repetida do método “Calcula_e_Apresenta”, o qual determina uma nova posição para a bola, fazendo um display da mesma na nova posição calculada (nete protótipo, considerou-se os jogadores estáticos). A noção de Computação Imprecisa neste caso, consistiu em evitar o display da imagem em pontos intermediários (reduzindo assim a qualidade da apresentação), quando tal tarefa não pudesse ser executada dentro do deadline especificado. Para tanto criou-se uma nova versão do método “Calcula_e_Apresenta”, denominado “So_Calcula”, que elimina o display da bola. Cada vez que o método “Calcula_e_Apresenta” é chamado, esta chamada é desviada para o MOG e este, interagindo com o MOE, decide qual versão será executada.

Como resultado desta implementação é possível que a trajetória da bola não seja percebida de forma contínua, mas, em contra-partida, a dinâmica do jogo não será afetada pela performance/carga da máquina onde a aplicação estiver sendo executada.

Este protótipo foi executado em máquinas com diferentes performances e sob diferentes condições de carga, tendo sido constatado que quanto mais lenta/sobrecarregada a máquina, maior é o benefício do uso da abordagem proposta; ou seja, em máquinas mais lentas/sobrecarregadas a trajetória da bola não foi apresentada de forma contínua, mas em todas elas o tempo consumido em cada trajetória foi o mesmo, garantindo assim que o tempo total do jogo (considerando-se os mesmos parâmetros) fosse o mesmo.

Quanto ao desempenho, constatou-se que overhead devido a presença da reflexão computacional não é signativo, pois a diferença de tempos observada entre a execução da aplicação sem controle temporal (implementada em Java) e com controle temporal usando reflexão computacional (implementada em Javassist), resumiu-se praticamente ao tempo gasto no controle temporal (caso o mesmo fosse acoplado ao corpo do proprio método).

6. Conclusões

Este artigo descreveu uma experiência preliminar relativa ao uso de Computação Imprecisa, implementada através de reflexão computacional, como forma de permitir a adaptação de aplicações de tempo real a plataformas distintas ou sob diferentes condições de carga. O mecanismo de adaptação utilizado foi a variação no tempo de execução das tarefas, enquanto que múltiplas versões foi a forma de programação utilizada.

A principal motivação para este trabalho, está na dificuldade de atendimento de 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 resultados obtidos, ainda que incipientes, são bastante animadores e nos permitem concluir que:

- A abordagem proposta é efetiva; ou seja as restrições temporais são satisfeitas independentemente do desempenho/carga em que a aplicação estiver sendo executada. Na experiência realizada isto foi constatado pelo fato de que que o tempo gasto em uma trajetoria da bola será sempre o mesmo, embora isso custe uma perda na qualidade visual da aplicação.
- A implementação de Computação Imprecisa através de Reflexão Computacional, também correspondeu ao esperado; ou seja: diminui a complexidade de desenvolvimento (na medida em que todo o controle temporal é programado no meta nível, separadamente das funcionalidades da aplicação); permite o reuso de classes base e meta-classes; permite que a política de controle seja alterada independentemente da aplicação e do ambiente de execução.
- A modelagem da aplicação segundo um modelo que suporte reflexão e tempo real (no caso o modelo RTR) permite que os aspectos temporais sejam considerados já na concepção da aplicação; desta forma os requisitos temporais da aplicação são melhor entendidos e podem ser atendidos com menor custo de desenvolvimento.
- No desenvolvimento, o custo adicional é a necessidade de projetar métodos flexíveis para as funcionalidades da aplicação e métodos de controle para as restrições temporais. Se considerarmos que o controle temporal é necessário, esse custo será menor que o associado com a implementação desse controle embutido na funcionalidade da aplicação.
- Quanto ao desempenho do protótipo, os testes realizados demonstraram que o overhead devido a reflexão computacional, desconsiderando o tempo adicional gasto na carga/instanciação dos meta-objetos, não é significativo.

Com base nos resultados obtidos neste experimento, o trabalho deverá ter continuidade com testes de aplicações de maior complexidade, envolvendo todas as formas de programação de computação imprecisa. Em um segundo momento deveremos avaliar a abordadgem proposta em aplicaçõoes distribuídas, em redes locais e no contexto da Internet.

Referências

[1] S. Chiba. Load-time structural reflection in Java. ECOOP2000 – Oriented Programming LNCS 1850, Springer Verlag, pp 313- 336, 2000.

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

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

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

[5] E. D. Jensen, C. D. Locke, H. Tokuda. A Time-Driven Scheduling Model for Real-Time Operating Systems. Proc. of the IEEE Real-Time Systems Symp., pp.112-122, dec 1985.

[6] 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.

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

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

[9] R. de Oliveira. Mecanismos de Adaptação para Aplicações Tempo Real na Internet. Congresso da Soc. Brasileira de Computação - SEMISH'98. Belo Horizonte-MG, 1998.

[10] R. de Oliveira, O. Furtado. Um mecanismo de Adaptação para aplicações Tempo Real Baseado em Computação Imprecisa e Reflexão Computacional. Simpósio Brasileiro de Engenharia de Software, SBES'99, pp 239-254, Florianópolis-SC, 1999.

[11] 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.