Explorando a Potencialidade e a Expressividade do Modelo Reflexivo Tempo Real RTR na Programação de Aplicações Tempo Real

Olinto Furtado[*] , *Jean-Marie Farines e Joni Fraga*

Laboratório de Controle e Microinformática – LCMI
Departamento de Automação e Sistemas – Universidade Federal de Santa Catarina
Cx. Postal 476, Florianópolis, SC, Brasil. CEP 88040–900
E-mail: {olinto,farines,fraga}@lcmi.ufsc.br

Resumo *: Este artigo apresenta um modelo de programação tempo real, denominado Modelo RTR e explora sua potencialidade e sua expressividade na representação de situações típicas encontradas no desenvolvimento de aplicações tempo real. Fundamentado nos conceitos de orientação a objetos e reflexão computacional, o modelo RTR contribui para a solução de vários problemas atualmente encontrados na programação de sistemas tempo real, tais como representação e controle de restrições temporais, dificuldade de gerenciamento da complexidade e falta de flexibilidade.*

Palavras-chave*: Eng. de Software, Tempo Real, Orientação a Objetos e Reflexão Computacional.*

1. Introdução

Sistemas Tempo Real (STR) são sistemas cuja corretude deve ser temporal além de lógica; ou seja, são sistemas em que as restrições temporais impostas pelo ambiente devem ser satisfeitas. Além de atender requisitos tais como correção temporal (*timeliness*), previsibilidade (*predictability*) e segurança de funcionamento (*dependability*), os STR requerem adicionalmente suporte para distribuição, integração com sistemas não tempo real e habilidade para adaptar-se a mudanças no sistema e no próprio ambiente operacional.

Em função dessas especificidades dos STR, arquiteturas, sistemas operacionais, metodologias de desenvolvimento, modelos e linguagens de programação e ferramentas de apoio usadas tradicionalmente para o projeto, implementação e execução de sistemas computacionais, por não suportarem a noção de tempo, não se mostram adequadas ao desenvolvimento de STR.

Em particular, com relação a modelos e linguagens para programação de aplicações tempo real, embora tenham sido propostas várias extensões tempo real de linguagens convencionais (RTCC [Gehani 91], por exemplo) e mesmo novas linguagens especialmente projetadas para tempo real (RT-Euclid [Stoyenko 86], por exemplo), nenhuma destas abordagens tem se mostrado satisfatória frente as necessidades atuais da área.

Mais recentemente, novos modelos (RTO.k [Kim 94], DRO [Takashio 92] e R2 [Honda 94]) e linguagens (Flex [Lin 91], RTC++ [Ishikawa 90] DROL [Takashio 92], Ada95 [Burns 96] e Real-Time Java [Nilsen 96]) baseados em paradigmas como orientação a objetos e reflexão computacional, foram propostas com o objetivo de melhor integrar aspectos funcionais e temporais e atender mais satisfatoriamente os requisitos de flexibilidade, reuso, manutenção, concorrência e distribuição.

Inserido neste contexto, propõe-se neste trabalho um modelo que visa contribuir na solução de vários dos problemas atuais dos STR. O modelo proposto, denominado modelo RTR, é um modelo de programação orientado a objetos, reflexivo e de tempo real, que caracteriza-se por permitir a representação e o controle de aspectos temporais (restrições, escalonamento e exceções) de aplicações tempo real que seguem uma abordagem de melhor esforço, de forma simples e flexível.

Este artigo apresenta o Modelo RTR (seção 2) e explora sua potencialidade e sua expressividade (seção 3) na programação de aplicações tempo real. Complementarmente, na seção 4 é apresentada uma análise comparativa entre o modelo RTR e outros modelos existentes, e na seção 5 são apresentadas as conclusões do trabalho.

---

[*] Professor do Departamento de Informática da UFSC, afastado para doutoramento no LCMI

2 - Modelo RTR

A principal motivação para a concepção de um novo modelo é a defasagem existente entre modelos e linguagens para programação tempo real e questões tais como: dificuldade para gerenciar a complexidade dos STR atuais, falta de flexibilidade na representação e controle dos aspectos temporais da aplicação, existência de um "*gap*" semântico entre projeto e implementação e forte dependência de ambientes operacionais específicos. Assim sendo, o Modelo RTR (Modelo Reflexivo Tempo Real) foi concebido com o objetivo de contribuir para a redução dos problemas existentes, favorecendo a representação de restrições temporais e permitindo que STR sejam estruturados e programados de forma sistemática, confiável e flexível.

2.1 - Caracterização do Modelo RTR

O Modelo RTR é um modelo de programação para aplicações tempo real que se caracteriza como sendo uma extensão concorrente, reflexiva e tempo real do modelo de objetos convencional.

Orientação a objetos - Por ser uma extensão do modelo de objetos convencional, o modelo proposto herda seus mecanismos de estruturação e conseqüentemente sua potencialidade relativa ao entendimento, reuso, extensão e manutenção de sistemas; complementarmente, por ser orientado a objetos, o modelo proposto possibilita um tratamento adequado das questões de concorrência e distribuição inerentes a muitos dos STR atuais, além de facilitar o uso de reflexão computacional.

Reflexão computacional - Reflexão computacional é introduzida no modelo RTR através da abordagem de meta-objetos [Maes 87], que estabelece a separação dos aspectos funcionais da aplicação (implementados através de objetos-base) do controle de seu comportamento (implementado através de meta-objetos). O uso de reflexão computacional flexibiliza o desenvolvimento de STR, na medida em que permite que as políticas de controle possam ser modificadas e 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. No modelo proposto, 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. Adicionalmente, a separação explicita entre questões funcionais e não funcionais, contribui para o entendimento do sistema, favorece o reuso e a manutenção de objetos-base e meta-objetos e incrementa a produtividade do programador.

Tempo real - O modelo de objetos convencional não possui suporte para representação e controle dos aspectos temporais típicos das aplicações tempo real, e portanto, deve ser estendido para que tais aspectos possam ser considerados. No modelo proposto, esta capacidade é obtida a partir da representação explicita das restrições temporais a nível de objetos-base (as quais são associadas a declaração e/ou ativação de métodos) e a verificação e controle destas restrições a nível de meta-objetos. Outra questão fundamental relativa a obtenção de correção temporal é a presença de um escalonador tempo-real operando no meta-nível da aplicação, o qual pode ser escolhido pelo usuário.

O esquema reflexivo usado na representação/controle das questões temporais, flexibiliza a definição e o uso de restrições temporais e algoritmos de escalonamento aumentando a possibilidade de que as restrições temporais sejam satisfeitas; restrições temporais e algoritmos de escalonamento podem ser escolhidos em função da especificidade de cada aplicação, de forma independente do suporte de execução. Contudo, em função de suas características e coerente com suas finalidades (modelagem e programação de STR *soft*), o modelo RTR segue uma abordagem dita de melhor esforço (*best-effort*), segundo a qual tenta-se evitar que as tarefas executem fora de suas especificações temporais e garante-se que toda violação temporal será detectada e que a ação alternativa (exceção temporal) correspondente a esta violação será executada.

Concorrência - Por ser concorrente, o modelo RTR permite que a concorrência inerente ao mundo real possa ser mapeada diretamente a nível de sistema computacional, favorecendo a obtenção de correção temporal, na medida em que as restrições temporais associadas às diferentes tarefas do sistema podem ser consideradas simultaneamente antes de uma decisão de escalonamento. O modelo proposto permite apenas concorrência

externa (isto é, concorrência entre objetos mas não entre métodos de um mesmo objeto) cujo controle (exclusão mútua na execução dos métodos de um objeto-base) é realizado de forma reflexiva a nível de meta-objetos; esta concorrência é provida pela existência de objetos ativos e pelo uso de mensagens (chamada de métodos) assíncronas. Outrossim, a sincronização condicional entre métodos de um objeto-base é tratada reflexivamente.

2.2 Estrutura geral e dinâmica de funcionamento

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. A figura 1 esquematiza a estrutura geral do modelo RTR e o inter-relacionamento entre seus componentes básicos.

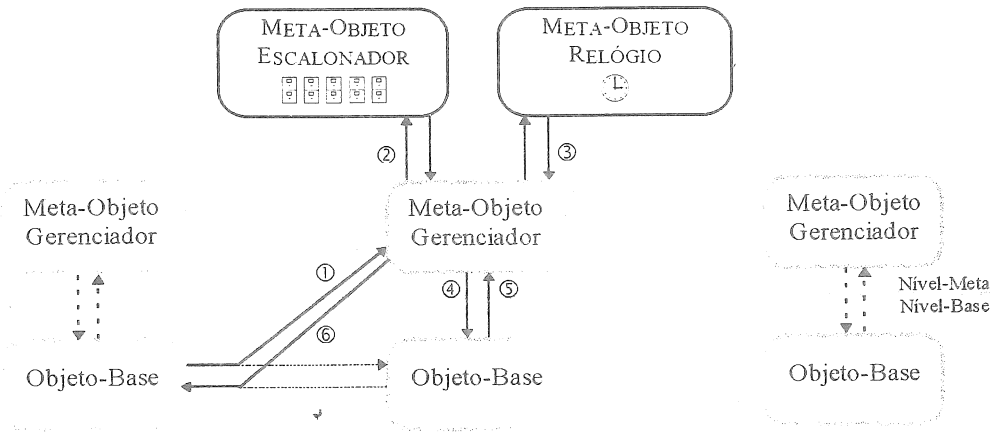

Figura 1 - Estrutura geral do modelo RTR

Tomando como base a figura 1, a dinâmica de funcionamento do modelo RTR, envolvendo a interação entre objetos-base e seus meta-objetos, pode ser assim descrita: A ativação de um método de um determinado objeto-base é desviado para seu meta-objeto gerenciador correspondente (ação ① da figura 1). O meta-objeto gerenciador interage com o meta-objeto escalonador (ação ②) e com o meta-objeto relógio (ação ③) para processar as restrições temporais associadas ao método solicitado e verificará as restrições de concorrência e de sincronização; caso estas restrições não sejam violadas, ativará o método solicitado no objeto-base (ação ④), retornando em seguida, via meta-objeto gerenciador para o objeto que deu origem a chamada (ações ⑤ e ⑥).

2.3 - Visão geral dos componentes básicos do modelo

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; tais restrições são implementadas nos meta-objetos gerenciadores correspondentes aos objetos base nos quais foram declaradas.

O exemplo a seguir ilustra a declaração de um método com restrição temporal associada,

void Animation(...). Periodic (Period, EndTime, TME=10). FrameIgnore()
{ ... }

onde o método "*Animation()*" é declarado como sendo um método periódico, onde os valores de *Period* e *EndTime* deverão ser especificados na ativação deste método; adicionalmente, esta declaração especifica que o tempo máximo de execução (TME) do método *Animation()* é de 10 ms e que *FrameIgnore()* é o

manipulador de exceções a ele associado. Semanticamente a ativação deste método fará com que os *frames* que compõem a animação sejam apresentados na freqüência (período) especificada ou então sejam ignorados.

Meta-objetos Manager (MOM) - Os meta-objetos manager são objetos *multi-threads* responsáveis pelo gerenciamento dos pedidos de ativação dos métodos de seus objetos-base correspondentes, pelo controle de concorrência, pelo gerenciamento das restrições de sincronização e pelo processamento das restrições temporais. O meta-objeto manager é estruturado em seções, as quais são descritas a seguir.

A seção de gerenciamento é responsável pelo recebimento de requisições dos métodos do objeto-base, pelo controle de concorrência na execução destes métodos e pela realização de controles específicos relativos a eles. Os pedidos de ativação recebidos pelo MOM são tratados concorrentemente, permitindo um tratamento mais efetivo das restrições temporais associadas. Ao receber uma requisição de um método com restrição temporal, o MOM ativa o método da seção de restrições temporais que implementa a restrição em questão e que será responsável pelo processamento da requisição a partir deste momento. Por outro lado, com base no estado do objeto-base, o MOM impede que mais de um método do objeto-base seja ativado simultaneamente.

Na seção de sincronização são especificadas as restrições de sincronização existentes entre métodos do objeto-base, ou seja, as relações de precedência que devem ser obedecidas na execução dos mesmos. Se um método liberado para execução pelo MOS, não puder executar imediatamente devido a restrições de sincronização, este método deve aguardar numa fila de pendências até que esteja apto a executar ou até ter suas restrições temporais violadas. A estrutura e o comportamento desta seção depende do mecanismo utilizado (*path-expressions*, *set-enables* ou máquina de estados) para expressar as restrições de sincronização.

A seção de exceções temporais implementa os manipuladores através de métodos; esta implementação é dependente da aplicação e portanto de responsabilidade do programador. Assim sendo, todos os manipuladores de exceção identificados no objeto-base devem ser implementados nesta seção; estes manipuladores serão ativados sempre que for prevista ou constatada a violação de uma restrição temporal.

A seção de restrições temporais é o local onde são definidos ou redefinidos os tipos de restrições temporais (um método para cada restrição) especificados nas declarações dos métodos do objeto-base. Para processar as restrições temporais, os métodos que as implementam interagem com o meta-objeto scheduler (MOS), o qual determina, de acordo com a política de escalonamento implementada, o momento a partir do qual o pedido de ativação que está sendo analisado pode ser liberado para execução. Neste momento, em função da disponibilidade de tempo e observados os aspectos de sincronização e concorrência (através da interação com as seções de sincronização e de gerenciamento), é decidido pela liberação da execução do método solicitado ou pelo levantamento de uma exceção temporal. Em seguida, após a execução, o controle é devolvido para a seção de gerenciamento, de onde retorna para o objeto que efetuou a requisição.

Meta-objeto scheduler (MOS) - este meta-objeto tem como função básica receber, ordenar e liberar pedidos de escalonamento de métodos dos objetos base que compõem a aplicação (on-line e de acordo com uma determinada política de escalonamento), visando influenciar o escalonamento realizado pelo suporte subjacente. A existência de um MOS no modelo possibilita que diferentes pedidos de escalonamento, originados do mesmo ou de diferentes meta-objetos manager de uma aplicação, possam ser analisados globalmente e ordenados de acordo com uma política de escalonamento (predefinida ou introduzida pelo programador) que melhor se adapte às especificidades da aplicação em questão.

Meta-objeto clock (MOC) - O MOC é uma abstração do relógio do sistema, estruturada na forma de objeto. Sua função básica é receber pedidos do meta-objeto manager (MOM) para ativar métodos num tempo futuro. Adicionalmente, o MOC é utilizado para detectar violação de *timeouts* e eventuais violações de restrições associadas com métodos que estão aguardando para serem escalonados (na fila de escalonamento do MOS) ou que foram atrasados (na fila de pendências do MOM) por motivo de concorrência ou sincronização.

3 - Explorando a potencialidade e a expressividade do modelo RTR

Além de suas facilidades inerentes, o modelo RTR suporta adicionalmente a representação e o controle de várias situações tipicamente encontradas no desenvolvimento de STR, sem que sua estrutura e sua semântica

de funcionamento tenham que ser modificadas. O suporte a tais funcionalidades, seguindo a filosofia do modelo, é realizado no meta-nível da aplicação, através da redefinição (extensão, alteração ou substituição) das funções básicas dos MOM e do MOS, que como previsto originalmente são acessíveis ao programador da aplicação. Nesta seção identificamos várias situações que podem ser representadas e controladas adequadamente pelo modelo RTR, demonstrando assim sua potencialidade, sua expressividade e seu alto grau de adaptabilidade; as situações consideradas estão relacionadas a: expressividade (3.1 a 3.3), ajuste dinâmico dos atributos das restrições temporais (3.4 e 3.5), escalonamento tempo real (3.6 a 3.8) e gerenciamento de memória (3.9).

3.1 - Refletindo aspectos de controle

Além dos aspectos temporais (restrições, escalonamento e exceções), de concorrência e de sincronização que são inerentemente reflexivos na filosofia RTR, outros aspectos comportamentais da aplicação também podem ser controlados reflexivamente; como exemplo, podemos citar : depuração, controle estatístico, persistência e qualidade de serviço. Para isso é necessário explicitar que tipos de controle devem ser realizados sobre quais métodos, ou seja é necessário classificar os métodos por categorias de controle.

Para obtermos este comportamento no modelo RTR (onde a operação de ativação de métodos já é reflexiva), basta adicionar uma cláusula "categoria" na declaração dos métodos dos objetos-base e providenciarmos para que, a nível de meta-objeto gerenciador, sempre que tal método for requisitado o comportamento correspondente a categoria de controle em questão (implementado através de métodos ou funções auxiliares) seja executado; tal controle poderá ser executado antes ou depois da execução do método requisitado, dependendo das especificidades de cada categoria.

3.2 - Polimorfismo Temporal

Segundo [Lin 91], em muitas situações práticas é preferível ter-se uma solução aproximada dentro do tempo especificado do que uma solução exata porém fora do tempo. A técnica usada na programação destas situações, denominada computação imprecisa, caracteriza-se pela realização do que é possível dentro do tempo disponível; esta técnica tem sido utilizada tanto na teoria de escalonamento quanto na prática de programação tempo real, como por exemplo nas linguagens Flex [Lin 91] e DROL [Takashio 92].

Da mesma forma que Flex e DROL, o modelo RTR também permite a realização de computação imprecisa baseada na técnica de programação denominada N-versões; segundo esta técnica, uma determinada tarefa (método) da aplicação pode possuir diferentes implementações (versões) com diferentes tempos de execução (onde a qualidade ou a precisão dos resultados obtidos é proporcional ao tempo de execução das versões disponíveis), sendo que em resposta a sua ativação, uma das versões disponíveis será escolhida para execução em função do tempo disponível no momento do escalonamento da tarefa solicitada.

Adicionalmente, a noção de polimorfismo temporal pode, por exemplo, ser usada para implementar o esquema de escalonamento "Task-Pair", o qual consiste em implementar uma tarefa tempo real com *deadline* "D" como sendo um par de tarefas, onde o primeiro componente, denominado *hard*, possui um tempo de execução de pior caso (*worst-case)* conhecido e o outro, denominado *soft*, possui um tempo de execução desconhecido ou estimado. Para tanto, uma tarefa (método) teria duas versões (*hard* e *soft*), sendo que a versão a ser executada seria escolhida dinamicamente em função da disponibilidade de tempo no momento da ativação. Alternativamente, esta situação pode ser implementada como proposto em [Mitchell 97], onde inicialmente a tarefa *soft* é liberada e executará até terminar ou até que o tempo disponível seja suficiente apenas para execução da tarefa *hard*;

3.3 -Representando restrições de sincronização multimídia

As características básicas do modelo RTR fazem com que ele seja particularmente atrativo para modelagem e programação de aplicações tempo real *soft*, incluindo aquelas que devem ser executadas em ambientes de propósito geral. Neste contexto destaca-se a classe de aplicações multimídia (tele-conferências,

ensino a distância, instrutores automáticos, entretenimento, aplicações médicas, etc.), cujos aspectos temporais podem ser adequadamente tratados através de abordagens do tipo melhor esforço.

Os aspectos temporais inerentes a aplicações multimídia estão diretamente relacionados com a questão de sincronização (intramídia e intermídias), a qual pode ser expressa através de diferentes modelos de especificação de sincronização [Blakowski 96]. Para demonstrar a potencialidade do modelo RTR na representação de sincronização multimídia, escolhemos o modelo de especificação de sincronização baseado em intervalos. Neste modelo, as 29 relações de intervalo consideradas relevantes para sincronização multimídia podem ser representadas por 10 operadores capazes de manipular tais relações [Blakowsky 96].

No modelo RTR, todos estes operadores (e conseqüentemente as relações de sincronização que eles representam) podem ser facilmente representados através de restrições temporais básicas, tais como "*Periodic*", "*Aperiodic*", "*Start-at*", e "*ActivationInterval*". Por exemplo, para representar uma animação parcialmente comentada por uma sequência de áudio, a relação de sincronização descrita em [Blakowski 96] como sendo "*Animation **while (d1, d2)** Audio*", pode ser expressa no modelo RTR pela declaração de dois métodos, um para cada mídia, como mostrado abaixo.

```
Void Animation ( ... ), ActivationInterval (StartTime, EndTime), AnimationException ( )
{ ... }
void Audio ( ... ), ActivationInterval (StartTime, EndTime), AudioException ( )
{ ... }
...
@Animation ( ... ), (T1, T2);
@Audio ( ... ), (T1 + D1, T2 - D2);
...
```

A sincronização desejada, representada graficamente na figura 2, será alcançada pela ativação assíncrona (indicada pelo símbolo "@") dos métodos *Animation()* e *Audio()*, onde *T1* , *T2*, *D1 e D2* são atributos temporais da relação de sincronização considerada. Os demais operadores são representáveis de forma similar ao operador "*while*".

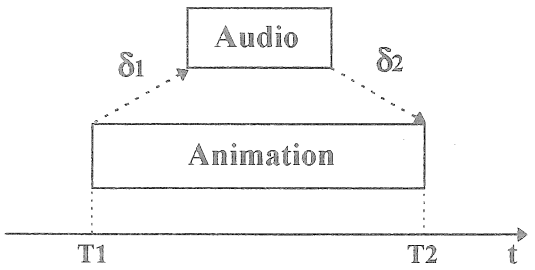

Figura 2 - Representação gráfica da relação "Animation while(δ1, δ2) Audio"

3.4 Ajuste dinâmico do tempo de execução das tarefas

Em função da complexidade do cálculo do tempo máximo de execução (*worstcase*) de uma tarefa (o qual é necessário para seu escalonamento), é comum, especialmente em sistemas tempo real *soft*, trabalhar-se com tempos médios, normalmente obtidos via medição; entretanto, como a abordagem de medição do tempo de execução não é exaustiva, os valores obtidos não são inteiramente confiáveis. Assim sendo, o ajuste dinâmico destes valores (baseado em novas medições realizadas durante a operação do sistema) pode contribuir para o incremento da confiabillidade dos mesmos e consequentemente para a confiabilidade do sistema como um todo.

No modelo RTR este ajuste pode ser realizado reflexivamente pelo meta-objeto manager, de forma transparente ao programador da aplicação. Para tanto, a partir de uma estimativa inicial (decorrente da experiência do programador, ou de medições *off-line*) o tempo de execução das tarefas (métodos) pode ser ajustado com base no tempo realmente gasto na execução de cada tarefa; os procedimentos de medição e ajuste

poderiam ser realizados como funções dos métodos responsáveis pelo gerenciamento de requisições ou daqueles que implementam as restrições temporais associadas aos métodos cujo tempo de execução está sendo ajustado. Visando reduzir a interferência do processo de medição e ajuste no tempo total de execução da aplicação, apenas métodos de determinadas categorias ou métodos com determinadas restrições temporais (periodicidade, por exemplo) associadas poderiam ter seus tempos de execução ajustados.

3.5 - Ajuste dinâmico dos valores dos atributos das restrições temporais

Embora normalmente os valores dos atributos das restrições temporais de uma aplicação tempo real dependam da própria aplicação, nem sempre esses valores são absolutos, e em muitos casos suportam uma certa faixa de variação sem comprometer o resultado da aplicação; além disso, existem casos em que os valores atribuídos inicialmente são meras estimativas que, por força do uso, acabam tornando-se padrão. Contudo, em função das limitações das construções utilizadas para representar as restrições temporais em muitos modelos e linguagens existentes, tais valores acabam sendo fixados como se fossem absolutos, não suportando nenhum ajuste dinâmico, independentemente da aplicação tolerar ou não determinadas variações.

No modelo RTR esta situação pode ser flexível e satisfatoriamente representada, uma vez que tanto as variações aceitáveis quanto estimativas totalmente intuitivas podem ser expressas naturalmente. No primeiro caso, podemos definir uma restrição temporal que represente explicitamente essa variação, fazendo com que o ajuste desejado (embutido no código do método que implementa a restrição) seja realizado dinamicamente a cada ativação do método ao qual a restrição estiver associada. No segundo caso, quando uma variação é aceitável porém não previsível, o ajuste poderá ser feito com base em informações obtidas dinamicamente, tais como folgas detectadas e número de violações temporais ocorridas.

3.6 - Análise de escalonabilidade dinâmica

Um número considerável de STR atuais caracterizam-se por serem flexíveis e dinâmicos, e geralmente não podem ter sua escalonabilidade analisada estaticamente (*off-line*); entretanto, é importante que as tarefas destes sistemas, tomadas individualmente, possam ser analisados dinamicamente quanto a sua escalonabilidade, objetivando antecipar a detecção de violações temporais e permitir que ações alternativas possam ser executadas mais cedo, contribuindo para um melhor resultado global do sistema. Por exemplo, uma ação alternativa poderia ser a realização de uma tarefa similar porém menos precisa ou ainda, no caso distribuído, a realização da mesma tarefa em outro processador; ação esta que poderia ser inviabilizada se a violação temporal não fosse prevista antecipadamente.

Esta abordagem pode ser representada no modelo RTR através da redefinição do meta-objeto escalonador, adicionando-se a política de escalonamento existente, uma política de admissão de tarefas, segundo a qual só seriam aceitos pedidos de escalonamentos com chances de serem executados dentro de suas especificações temporais e sem comprometer a escalonabillidade das tarefas admitidas anteriormente. Para tanto, além das informações temporais das tarefas (disponíveis via reflexão), o meta-objeto escalonador teria que ter uma noção do tempo total de CPU disponível para a aplicação (nos casos onde diferentes aplicações são executadas simultaneamente) e gerenciar esta disponibilidade em função do tempo previsto e do tempo efetivamente gasto na execução das tarefas admitidas. Uma vantagem na implementação desta abordagem segundo a filosofia RTR, é a possibilidade de se poder combinar diferentes políticas de admissão com diferentes políticas de escalonamento, de maneira a melhor satisfazer os requisitos temporais da aplicação.

3.7 - Mudança dinâmica do algoritmo de escalonamento

Além da proposta básica do modelo RTR que prevê a realização de um meta-escalonamento a nível de aplicação, permitindo que o projetista defina estaticamente o meta-objeto escalonador (e conseqüentemente a política de escalonamento) a ser usado na aplicação, o modelo RTR também suporta a mudança da política de escalonamento durante a execução da aplicação.

Para tanto, o meta-objeto escalonador deve ser programado de forma a suportar duas ou mais políticas de escalonamento e possuir uma função auxiliar capaz de analisar o comportamento da aplicação durante a vigência de uma determinada política e, em função desta análise, chavear o escalonador para outra política disponível; esta análise poderia ser realizada em função de uma qualidade desejada (*default* ou especificada pelo usuário no *start-up* da aplicação), com base, por exemplo, no percentual de violações temporais ocorridas em um determinado intervalo de tempo.

A mudança de política de escalonamento durante a execução do sistema, é particularmente atrativa para STR que devem ser executados em ambientes de propósito geral (convivendo com outras aplicações tempo real e mesmo com aplicações convencionais), onde nada pode ser antecipado com relação a carga total do sistema computacional no qual a aplicação está sendo executada. Além disto, este esquema é apropriado para a fase de testes do sistema, onde diferentes políticas de escalonamento podem ser testadas e a escolha da política definitiva a ser usada na operação do sistema seja menos intuitiva e melhor fundamentada.

3.8 - Uso simultâneo de diferentes políticas de escalonamento

A escolha de uma política de escalonamento adequada é fundamental para que uma aplicação tenha suas restrições temporais satisfeitas. Entretanto, nem sempre uma única política pode ser adequada para todas as tarefas que compõem uma aplicação; neste caso a solução é o uso simultâneo de diferentes políticas para o escalonamento de diferentes classes de tarefas, segundo suas características temporais e/ou de acordo com diferentes níveis de garantia desejados. Esta abordagem tem sido usada nas linguagens RTCC [Gehani 91] e Flex [lin 91], e considerada no RT-MOP (Real-Time Meta-Object Protocol) proposto em [Mitchell 97].

No modelo RTR, esta diversidade de políticas pode ser obtida através do meta-objeto escalonador, o qual pode ser programado de forma a suportar simultaneamente diversas políticas; uma forma de se obter isto, seria sobrecarregar o método responsável pelo recebimento de pedidos de escalonamento através de uma implementação para cada política desejada e usar diferentes filas de escalonamento. A política a ser usada para o escalonamento de um método (tarefa) particular poderia ser deduzida dinamicamente em função das características temporais ou de outros atributos tais como prioridade ou categoria do método; alternativamente, a política a ser usada poderia ser definida na declaração do método, por exemplo através da cláusula "categoria". Adicionalmente, no modelo RTR esta funcionalidade pode ser combinada com aquelas introduzidas nas subseções 3.6 e 3.7, resultando em esquemas de escalonamento extremamente flexíveis e adaptáveis.

3.9 - Controlando reflexivamente a disponibilidade de memória

Embora o controle do tempo de CPU seja de importância fundamental para sistemas tempo real, a disponibilidade de tempo não é suficiente para que os requisitos temporais de uma aplicação sejam satisfeitos, é necessário que os demais recursos utilizados também estejam disponíveis no momento e na quantidade desejados. Dentre tais recursos, naturalmente se destaca o recurso memória, que assim como o tempo de CPU deve estar disponível para que as tarefas possam ser executadas corretamente.

No modelo RTR, este controle pode ser feito dinamicamente usando a capacidade reflexiva do modelo; ou seja a operação de criação de um objeto tempo real pode ser considerada uma operação reflexiva, a qual só será executada se houver disponibilidade de memória para alocação do objeto que está sendo criado. Esta condição deve ser verificada reflexivamente, e se for o caso um coletor de lixo deve ser ativado para obtenção da quantidade de memória necessária. Naturalmente para que a operação de criação de objetos mantenha-se previsível o coletor de lixo usado deve ser determinístico (como por exemplo aquele proposto em [Nilsen 96]) e o tempo possivelmente gasto com a coleta de lixo (na alocação de objetos) deve ser considerado.

4 - Análise comparativa entre o modelo RTR e outros modelos existentes

Vários modelos de programação tempo real baseados em objetos e/ou reflexão computacional tem sido propostos nos últimos anos, alguns definidos explicitamente (RTO.k [Kim 94], DRO [Takashio 92], R2 [Honda 94] e o RT-MOP [Mitchell 97]) e outros embutidos implicitamente em linguagens de programação tempo real

(Flex [Lin 91], RTC++ [Ishikawa 91] e RT-Java [Nilsen 96]). Todos estes trabalhos visam simplificar o projeto e/ou a programação de STR, oferecendo facilidades relativas a estruturação dos sistemas e a representação explicita de restrições temporais.

Comparativamente com os modelos RTO.k, Flex, RTC++ e RT-Java, que não são reflexivos, o modelo RTR é no mínimo tão expressivo quanto eles (pois todas as facilidades destes podem ser representadas no modelo RTR), com vantagens adicionais devidas ao uso de reflexão. Em geral estes modelos apresentam um conjunto fixo de restrições temporais e políticas de escalonamento, as quais, via de regra, são dependentes do suporte de execução e/ou de ambientes operacionais específicos, impedindo a definição e o uso de novos tipos de restrições temporais e algoritmos de escalonamento a nível da aplicação; aspectos estes que no modelo RTR podem ser livremente escolhidos, definidos e modificados de acordo com as necessidades das aplicações.

Com relação aos modelos DRO, R2 e o RT-MOP, que como RTR são reflexivos, existem diferenças básicas relativas a filosofia de programação, aos aspectos tratados de forma reflexiva e a abrangência e efetividade dos modelos propostos; tais diferenças são comentadas nos parágrafos subseqüentes.

DRO é um modelo de objetos tempo real distribuído que suporta as propriedades de melhor esforço e menor prejuízo através de um mecanismo de invocação polimórfica. DRO utiliza meta-objetos para controle de sincronização e para definição de protocolos de comunicação com comportamento tempo-real; aspectos estes que no modelo RTR podem ser tratados, respectivamente, via seção de sincronização e restrições temporais especiais. Por outro lado, o conjunto de restrições temporais de DRO é fixo e não existe a noção de escalonamento a nível de aplicação (ambos são dependentes do suporte subjacente, ao contrário de RTR).

R2 é uma arquitetura reflexiva baseada em objetos, que como RTR usa reflexão para definir e controlar restrições temporais e algoritmos de escalonamento a nível de aplicação. Entretanto, apesar das similaridades de propósito entre R2 e RTR, os modelos diferem em vários aspectos, dentre os quais destacamos : escopo das restrições temporais (no modelo RTR as restrições são associadas a métodos, enquanto em R2 elas são associadas a segmentos de códigos); tipos de restrições temporais que podem ser utilizados (R2 não suporta restrições temporais *time-trigger*); e abrangência do meta-escalonamento (enquanto no modelo RTR todos os métodos com ou sem restrição temporal são considerados, no modelo R2 apenas os segmentos de código com restrições associadas são submetidas a tal escalonamento), Além disso, ao contrário de R2, o modelo RTR trata as questões temporais, de concorrência e de sincronização de forma integrada e também permite que controles não temporais sejam tratados reflexivamente. Conjuntamente, estas diferenças fazem com que o modelo RTR seja mais expressivo na representação e mais efetivo no controle dos aspectos temporais das aplicações..

O RT-MOP (Real-Time Meta-Object Protocol) proposto em [Mitchell 97], tem como objetivo permitir que o comportamento temporal das aplicações possa ser controlado e modificado dinamicamente. Segundo o RT-MOP proposto, as aplicações são estruturadas em grupos e meta-grupos de escalonamento, onde cada meta-grupo possui um meta-objeto responsável pelas decisões de escalonamento relativas aos objetos que compõem o grupo correspondente; tais meta-objetos podem implementar diferentes políticas de escalonamento e oferecer níveis diferenciados de garantias. Comparativamente, o modelo RTR possui expressividade suficiente para suportar diferentes políticas de escalonamento em uma mesma aplicação (como visto na seção anterior), com a vantagem de permitir que diferentes métodos de um mesmo objeto sejam escalonados segundo diferentes políticas de escalonamento, aumentando a flexibilidade do modelo; outros aspectos tais como representação e controle de restrições temporais não são explicitados na referência citada.

5 - Conclusão

Neste artigo apresentamos as principais características do modelo RTR e exploramos seu potencial e sua expressividade relativos a representação de situações típicas encontradas em aplicações tempo real. O modelo apresentado, através da integração de orientação a objetos, reflexão computacional e tempo real, contribui para a solução de vários problemas encontrados na programação de STR. Neste sentido destacamos o potencial do modelo com relação a estruturação destes sistemas (reduzindo a complexidade de gerenciamento), a flexibilidade provida (facilitando a representação e o controle de restrições temporais e a adaptação dos sistemas

às especificidades das aplicações) e a habilidade em suportar evoluções do sistema e mesmo do ambiente operacional (sem que o nível-base da aplicação ou o suporte de execução tenham que ser modificados).

Com relação aos resultados disponíveis, várias implementações experimentais do modelo proposto foram realizadas: uma extensão distribuída do modelo RTR para ambientes abertos baseados na plataforma CORBA [Siqueira 96],[Furtado 96a] e [Fraga 97]; uma extensão do modelo RTR para o suporte de aplicações de trabalho cooperativo [Fritzke 97]; e um mapeamento do modelo RTR sobre a linguagem de programação Java [Nishida 96]. A partir destas implementações, foi possível constatar que o desenvolvimento de sistemas tempo real seguindo a filosofia do modelo RTR, contribui de fato na redução dos problemas encontrados atualmente na prática de programação destes sistemas. Esta constatação nos motivou a realizar uma implementação explicita do modelo RTR através da linguagem Java/RTR [Furtado 96b], uma extensão da linguagem Java que permitirá a programação de aplicações tempo real combinando os benefícios do modelo RTR com as facilidades de Java (Sun Microsystems Inc.).

Na continuidade do trabalho, estão previstos a implementação de um tradutor de Java/RTR para Java, o desenvolvimento de ferramentas para cálculo e medição dos tempos de execução de programas Java/RTR e a proposição de uma metodologia para desenvolvimento de sistemas tempo real baseada no modelo RTR.

Referências bibliográficas

[Blakowski 96] - Blakowski, G. and Steinmetz, R. "A Media Synchronization Survey : Reference Model, Specification, and Case Studies". IEEE Journal in Selected Areas in Communications, 14(1) : 5-35, 1996.

[Burns 96] - Burns, A. "Concurrent and Real-Time Programming in ADA95", Notes of the talk presented at WRTP'96, Gramado, RS, Brasil, november 4-6, 1996.

[Fraga 97] - Fraga, J., Farines, J-M. and Furtado, O. "RTR Model : An Approach for Dealing with Real-Time Programming in Open Distributed Systems", WORDS'97, Newport Beach, Ca, USA. February 5-7, 1997.

[Fritzke 97] - Fritzke Jr., U. e Farines, J-M. "Modelagem e Implementação de Aplicações de Computação Cooperativas em Ambientes Distribuídos Heterogêneos"; SEMISH'97, Brasília, DF, Brasil, agosto 1997 (a ser apresentado).

[Furtado 96a] - Furtado, O., Siqueira, F., Fraga, J. and Farines, J-M. "A Reflective Model for Real-Time Applications in Open Distributed Systems". In: WRTP'96 Proceedings, November 1996, Brazil.

[Furtado 96b] - Furtado, O. and Farines, J-M. "Java/RTR - Uma Linguagem Reflexiva para Programação de Aplicações Tempo-Real". In SBLP'96 *Proceedings*, September 1996, Brazil.

[Gehani 91] - Gehani, N. and Ramanritham, K. "Real-Time Concurrent C: A Language for Programming Dynamic Real-Time Systems", The Journal of Real-Time Systems, n. 3, pp 377-405, 1991.

[Honda 94] - Honda, Y. and Tokoro, M. "Reflection and Time-Dependent Computing: Experiences with the R2 Architecture", Sony Computer Science Laboratory Inc., Tokio, Japan, July 1994.

[Ishikawa 90] Y. Ishikawa, Y., Tokuda, H. and Mercer, C. W. "Object-Oriented Real-Time Language Design: Constructs for Timing Constraints", ECOOP/OOPSLA'90, pp. 289-298, outubro 1990.

[Kim 94] - Kim, K., Bacellar, L., Kim, Y. and Choi, D. "Distinguishing Features and Potential Roles of the RTO.k Object Model". In: WORDS'94, Proceedings, IEEE Pub, 1994.

[Lin 91] - Lin, K.-J., Liu, J.W.S., Kenny, K.B. and Natarajan, S. "FLEX: A Language for Programming Flexible Real-Time Systems", chapter 10 of "Foundations of Real-Time Computing: Formal Specifications and Methods", edited by Tilborg, A. and Koob, G.. Kluwer Academic Publishers, USA, 1991.

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

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

[Nilsen 96] - Nilsen, K. "Real-Time Java (draft 1.1)", Iowa State University, Ames, Iowa, 1996.

[Nishida 96] - Nishida, D. "Estudo e implementação do Modelo Reflexivo Tempo Real RTR sobre a Linguagem Java", Dissertação de Mestrado - LCMI/EEL/UFSC, Florianópolis, SC, Brasil, dezembro 1996.

[Siqueira 96] - Siqueira, F. "Programação de Aplicações com Requisitos Temporais em Sistemas Abertos: O modelo de Objetos Reflexivo Tempo Real Distribuído", Dissertação de Mestrado , LCMI/EEL/UFSC, Florianópolis, SC, Brasil, Junho 1996.

[Stoyenko 86] - Stoyenko, A. D. and Kligerman, E. "Real-Time Euclid : A Language for Reliable Real-Time Systems", IEEE Transactions in Software Engineering, vol. SE-12, n.9, pp. 941-949, september 1986.

[Takashio 92] - Takashio, K. and Tokoro, M. "DROL : An Object-Oriented Programming Language for Distributed Real-Time Systems", OOPSLA'92 Conference Proceedings, October 1992.