Simulador de Escalonamento para Sistemas de Tempo Real *

Gisélia Magalhães Cruz[1], **George Lima**[1]

[1]Departamento de Ciência da Computação – Universidade Federal da Bahia (UFBA)
Avenida Adhemar de Barros, s/n – 40.170-110 – Salvador – BA – Brasil

{giselia,gmlima}@dcc.ufba.br

Abstract. *Real-time systems are those whose correctness depends on both the produced logical results and the time within which such results are produced. One way to check the system temporal correctness is by means of simulation tools. This work presents a real-time scheduling simulator, developed in Java, which allows one to analyze several types of real-time systems since it takes into account three widely used scheduling policies. Moreover, due its modular structure, the simulator can be further extended to provide other functionalities.*
Keywords: *Simulation, Scheduling, Real-time Systems*

Resumo. *Sistemas de tempo real são aqueles cuja correção depende tanto do resultado lógico produzido quanto do tempo no qual tal resultado foi gerado. Uma forma de estudar a correção temporal de sistemas é utilizando ferramentas de simulação. Este trabalho apresenta um simulador de escalonamento para sistemas de tempo real, desenvolvido em Java, que contempla três políticas de escalonamento bastante conhecidas. Devido a sua estrutura modular, o simulador pode ser estendido para prover novas funcionalidades.*
Palavras-chave: *Simulação, Escalonamento, Sistemas de Tempo Real*

1. Introdução

Sistemas de tempo real são sistemas computacionais que possuem restrições especiais relacionadas a seu comportamento temporal [Liu 2000]. Nestes sistemas cabe ao escalonador garantir a seqüência adequada de ações a serem executadas tal que restrições temporais não sejam violadas, o que garante a correção temporal do sistema. A validação matemática de tais sistemas mostra-se adequada para estudar o comportamento dos sistemas no pior caso. No entanto, informações relacionados ao caso médio, igualmente importantes para a caracterização de sistemas de tempo real, não são contempladas por tal abordagem.

Neste contexto, o uso de simulação desponta como alternativa complementar para avaliação do comportamento dos sistemas em questão [Audsley et al. 1994] [Casile et al. 1998]. Por exemplo, através de simulação, pode-se verificar indiretamente o quão provável é que dado sistema viole suas restrições temporais. Outros aspectos de interesse contemplados pela simulação são tempo médio de resposta e a verificação de possíveis efeitos de sobrecarga no sistema.

Este trabalho descreve uma ferramenta projetada para simular escalonamento em sistemas de tempo real. Duas abordagens de escalonamento foram contempladas: escalonamento estático, baseado na atribuição de prioridades fixas, e escalonamento dinâmico,

*Esta ferramenta foi desenvolvida pelo primeiro autor como parte de um projeto final de conclusão de curso de graduação.

onde as prioridades das operações escalonadas variam em função do tempo. Com relação ao escalonamento estático, a ferramenta contém ainda a possibilidade de escalonar tarefas conhecidas apenas em tempo de execução. A ferramenta possui uma estrutura bastante modularizada, o que viabiliza posteriores extensões.

Este artigo está organizado em seis seções. A seção 2 traz uma contextualização na área, definindo os conceitos iniciais, os algoritmos envolvidos e os testes de escalonabilidade. A seção 3 apresenta alguns trabalhos relacionados ao tema. A seção 4 descreve o simulador desenvolvido. A seção 5 mostra um estudo de caso realizado. Por fim, na seção 6, as conclusões deste projeto e as perspectivas de trabalhos futuros são apresentadas.

2. Escalonamento em Sistemas de Tempo Real

O escalonador é componente básico para garantia de bom funcionamento em sistemas computacionais, pois é responsável por decidir qual tarefa irá executar quando houver mais de uma pronta para execução. O algoritmo utilizado para tal finalidade é denominado algoritmo de escalonamento, o qual segue um determinado critério para escolher uma tarefa em detrimento de outras. Em sistemas de tempo real, a relevância do escalonador é ainda mais evidente, visto que está instrinsecamente ligado à correção do sistema, já que a tarefa escalonada deve cumprir restrições temporais.

A especificação dos requisitos temporais de sistemas de tempo real é geralmente feita através dos atributos temporais de suas tarefas. Estes são usados tanto para caracterizar o sistema em questão quanto para impor o comportamento temporal que deve ser obedecido durante a execução das tarefas. Por exemplo, o limite de tempo para execução de determinada tarefa é comumente especificado pelo atributo *deadline* enquanto que o comportamento temporal relacionado à freqüência de execução pode ser especificado pelo atributo período.

Os atributos das tarefas são importantes parâmetros usados pelo algoritmo de escalonamento nos sistemas de tempo real. Por exemplo, suponha que uma tarefa não pode perder seus *deadlines*. Neste caso, é objetivo do escalonador fazer com que o tempo de resposta de cada instância da tarefa, tempo decorrido desde de seu lançamento até o instante em que ela é completada, seja menor ou igual a seus *deadlines*. A seguir estão definidos os atributos mais freqüentemente encontrados:

- Tempo de Lançamento: instante de tempo no qual a instância de tarefa se torna disponível para execução. Se as condições de dependência de dados e controle estiverem satisfeitas uma tarefa pode executar a partir do seu tempo de lançamento.
- Tempo Máximo de Execução (c_i): tempo máximo necessário para execução de uma instância de tarefa.
- *Deadline* ou *Deadline* Absoluto (d_i): instante de tempo no qual a execução de uma instância de uma tarefa deve estar completa. Diz-se que as instâncias de tarefa não têm *deadline* se seu valor é infinito.
- *Deadline* Relativo (D_i): tempo máximo de resposta disponível a partir do tempo de lançamento. A soma do tempo de lançamento com o *deadline* relativo é igual ao *deadline* absoluto.
- Período (p_i): periodicidade com que uma nova instância de tarefa será lançada.

Diz-se que uma tarefa é *periódica* quando cada uma de suas instâncias é lançada em intervalos de tempo regulares. Uma tarefa é chamada de *esporádica* quando conhece-

se apenas o intervalo mínimo de tempo entre lançamentos de instâncias consecutivas. Tarefas podem ainda ser *aperiódicas*. Neste caso não se conhece nada sobre a freqüência de ativação de suas instâncias. O atributo período é geralmente usado para especificar tarefas periódicas ou esporádicas.

O atributo *deadline* está associado à criticidade das tarefas. De forma geral, há tarefas críticas e não-críticas. O sistema operacional, através de seu escalonador, deve garantir que nenhum dos *deadlines* das instâncias das tarefas críticas seja perdido. Exemplos de tais tarefas podem ser encontrados em controle industrial, de avião ou automotivo. Perder o *deadline* da verificação de altitude de um avião, por exemplo, pode provocar catástrofes caso o avião esteja aterrissando. Para tarefas não-críticas, por outro lado, o não cumprimento ocasional de *deadlines* é permitido, apesar de não ser desejável. Por exemplo, num sistema multimídia, deixar de exibir algum quadro de imagem vez por outra é tolerável.

Existem várias classificações para tarefas críticas e não-críticas. A definição adotada neste trabalho baseia-se nos estudos de Jane Liu [Liu 2000]. Segundo esta definição, uma restrição temporal é crítica quando o usuário necessita de uma validação de que o sistema sempre atenda à esta restrição. Validação é a demonstração através uma prova correta, de um procedimento eficiente ou de exaustivos teste e simulações. Assim, deve haver uma validação eficiente de algoritmos e métodos, bem como de escalonamento e estratégias de administração de recursos. Caso não seja necessária nenhuma validação ou seja suficiente uma demonstração de que as instâncias de tarefas satisfazem algumas restrições estatísticas (como uma restrição temporal suficiente em termos de média estatística), a restrição temporal é não-crítica. Assim, nesta classificação, a diferença entre restrições críticas e não-críticas é equivalente à diferença entre “garantia” e “melhor esforço” de serviço. Geralmente, tanto tarefas periódicas como esporádicas têm restrições temporais críticas enquanto que tarefas aperiódicas têm restrições temporais não-críticas.

É interessante ressaltar que não há vantagem em completar cedo uma instância de tarefa crítica. Desde de que ela complete a execução num instante anterior ou igual a seu *deadline,* seu tempo de resposta não é importante. É freqüentemente vantajoso, ou até mesmo essencial, adiar a execução de instâncias de tarefas críticas em favor de outras não-críticas a fim de melhorar o tempo de resposta das tarefas não-críticas.

2.1. Escalonamento de Tarefas Periódicas

Algoritmos de escalonamento para sistemas de tempo real podem basear-se em diversas abordagens conforme lidem com a escolha das prioridades. Este trabalho foca na abordagem orientada a prioridade, na qual o escalonamento é realizado em tempo de execução. Decisões de escalonamento são tomadas quando ocorrem eventos como lançamentos e encerramentos de instâncias de tarefas e estas possuem uma prioridade associada. As tarefas são colocadas numa fila de prontas, ordenada pela prioridade. No momento de selecionar uma tarefa, o escalonador atualiza a fila de prontas e então escalona a instância do início da fila. Diferentes algoritmos definem as prioridades segundo diferentes critérios.

Os algoritmos para escalonamento de tarefas periódicas baseados na abordagem orientada a prioridade podem ser classificados em dois tipos: prioridade fixa e prioridade variável. Um algoritmo de prioridade fixa define prioridades em tempo de projeto, as prioridades das instâncias de uma mesma tarefa são iguais, assim, a prioridade de cada

tarefa é fixa em relação as outras tarefas. Já um algoritmo de prioridade variável atribui prioridades em tempo de execução, instâncias de uma mesma tarefa podem receber prioridades diferentes, o que faz com que uma tarefa tenha sua prioridade modificada durante a execução do sistema.

Vários algoritmos baseiam-se na abordagem orientada a prioridade, entre eles *Rate-monotonic* (RM), *Deadline-monotonic* (DM) e *Earliest Deadline First* (EDF) [Liu 2000]. Os dois primeiros são algoritmos de prioridade fixa enquanto o segundo é de prioridade variável.

O algoritmo *RM* atribui prioridade para as tarefas usando como critério os seus períodos, quanto menor o período maior a prioridade da tarefa. O algoritmo *DM*, por sua vez, define a prioridade das tarefas de acordo com o seu *deadline* relativo. Quanto menor o *deadline* relativo maior a prioridade da tarefa. Já o algoritmo *EDF* determina a prioridade de cada instância de tarefa individualmente baseado em seu *deadline* absoluto.

2.2. Escalonamento de Tarefas Aperiódicas ou Esporádicas

Os algoritmos vistos anteriormente contemplam apenas o escalonamento de tarefas periódicas. Quando tarefas periódicas ou esporádicas coexistem com tarefas aperiódicas é necessário prover uma maneira para que se possa escalonar as últimas sem comprometimento das primeiras. Para tarefas aperiódicas não é possível fazer nenhuma previsão quanto intervalos de lançamento de suas instâncias ou sobre o tempo necessário para sua execução. Já no caso de tarefas periódicas e esporádicas, conhece-se alguma informação sobre o intervalo de tempo entre lançamentos de instâncias consecutivas.

Cabe aos algoritmos de escalonamento que consideram também as tarefas aperiódicas e esporádicas determinar em que momento deve-se executá-las. Este trabalho foca nos algoritmos que suportam escalonamento de tarefas aperiódicas. Nestes algoritmos o objetivo é escalonar as tarefas aperiódicas assim que possível sem causar perdas de *deadline* das tarefas periódicas e das tarefas esporádicas aceitas no sistema. Um bom algoritmo de escalonamento proporciona às tarefas aperiódicas o menor tempo de reposta possível e escalona tarefas esporádicas sem prejuízo as tarefas periódicas e as tarefas esporádicas aceitas.

Há diversas técnicas para se tratar o escalonamento de tarefas aperiódicas no modelo acima descrito. Notadamente, para algoritmos orientados à prioridade é comum usar um tarefa periódica especial chamada de servidor, que é usada para executar instâncias aperiódicas. O servidor é tratado pelo escalonador como uma tarefa periódica qualquer, no entanto, quando escalonado verifica a fila de tarefas aperiódicas. Se a fila não estiver vazia a instância do início da fila irá executar, caso a fila esteja vazia o servidor é suspenso. O servidor possui um período, p_s, e um tempo máximo de execução, que é chamado de *carga*, e_s. Ao ser escalonado e executar uma tarefa aperiódica a carga do servidor é consumida na taxa de um por unidade de tempo.

Na abordagem *Servidor com Preservação de Banda* a carga do servidor é preservada ao encontrar a fila de tarefas aperiódicas vazia. Desta forma, uma tarefa aperiódica lançada após o servidor ter encontrado a fila aperiódica vazia tem a oportunidade de ser escalonada no mesmo período, o que melhora o tempo de resposta das tarefas aperiódicas. Os algoritmos baseados nesta abordagem diferem um do outro quanto as regras definidas para consumo e recarregamento da carga. Seguem esta abordagem os algoritmos das

famílias *Servidor Deferrable* e *Servidor Esporádico*[Liu 2000].

2.3. Análise de Escalonamento

Para garantir que um sistema está temporalmente correto é necessário submetê-lo a uma prova de correção. O *teste de escalonabilidade* propõe-se a verificar se determinado sistema tem todos os seus *deadlines* críticos satisfeitos quando escalonado de acordo com determinado algoritmo. Geralmente este teste é feito em tempo de projeto e baseia-se no conhecimento sobre as tarefas periódicas e esporádicas do sistema. Tais testes são dados por meio de inequações que se satisfeitas podem informar se determinado sistema é escalonável. Seja qual for o teste de escalonabilidade usado, ele dará apenas um tipo de informação: há ou não chances de existir violação de *deadline*. Esta informação é útil, mas pode não ser suficiente.

O uso de simulações para verificar o comportamento do sistema, embora não seja prova de correção, complementa o uso dos testes de escalonabilidade. Enquanto os testes verificam apenas o pior caso, as simulações focam no caso geral, o que fornece uma idéia melhor do real comportamento do sistema. Quando o teste de escalonabilidade não fornece conclusões a respeito da escalonabilidade ou não de um sistema, pode-se analisá-lo a partir de sua simulação. Mesmo quando o teste aponta a não escalonabilidade de um sistema, o uso de simulação fornece uma análise mais criteriosa. A simulação pode demonstrar, por exemplo, que a não escalonabilidade apontada no teste restringi-se a uma perda de *deadline* que causa um prejuízo insignificante ao sistema, além de apontar quando esta perda realmente acontecerá.

3. Trabalhos Relacionados

Pode-se dividir os trabalhos sobre simulação para sistemas de tempo real em duas categorias: simuladores e *frameworks*. Estes últimos, a exemplo do *FORTISSIMO* [Kramp et al. 2000] e do *DRTSS* [Storch 1997], destinam-se ao desenvolvimento de simuladores. Já os simuladores propriamente ditos são voltados para o usuário que deseja testar determinados sistemas. Pode-se citar entre eles:

- *STRESS* [Audsley et al. 1994] é um ambiente geral para análise e simulação do comportamento de aplicações de tempo real críticas. Seu objetivo é ser uma ferramenta de avaliação de algoritmos de escalonamento e administração de recursos. Pode ser usado também para estudar o comportamento geral de aplicações e *kernels* de tempo real. É composto de uma linguagem genérica para descrição de arquitetura, *kernel* e aplicação; recursos de análise para *kernel* e aplicações; e recursos de simulação, além de exibir graficamente o escalonamento simulado. Permite escalonamento com prioridades fixas e variáveis, possibilitando comunicação entre os processos de forma síncrona ou assíncrona.
- *RTSIM* [Casile et al. 1998], *Real-Time system SIMulator*, é uma ferramenta para definição, simulação e análise de sistemas de tempo real, que contempla de simples cenários com poucos tipos de tarefas num único nó até várias arquiteturas complexas constituídas de múltiplos nós conectados com uma rede e compartilhando um conjunto de recursos de *hardware* e *software*. Esta ferramenta é baseada em uma biblioteca de simulação de eventos discretos chamada *MetaSim* [met 2005]. Portabilidade, flexibilidade e facilidade de uso foram características pretendidas no desenvolvimento da *RTSIM*. O código desta ferramenta foi escrito

em C++ e o de sua interface gráfica foi desenvolvida em Java. Pode-se especificar um sistema de três formas: através de sua interface gráfica, através de arquivos de descrição ou através de código em C++ (neste nível pode-se definir novos algoritmos de escalonamento).

4. Simulador

A ferramenta que será descrita nesta seção é um simulador de escalonamento para sistemas de tempo real. Esta ferramenta foi desenvolvida em Java e caracteriza-se por sua simplicidade. Sua estrutura modular facilita a extensão de novas funcionalidades.

A proposta deste simulador é ser uma ferramenta de comparação da eficiência de algoritmos de escalonamento. Esta comparação é conseguida a partir da execução de um sistema composto por tarefas e pelo algoritmo a ser avaliado. O simulador exibe o escalonamento executado e acumula informações referentes aos critérios de avaliação. Perdas de *deadline* e tempo de resposta são critérios eficazes em termo de avaliação de algoritmos.

4.1. Modelo de Simulação

A simulação é realizada num intervalo de tempo que começa no instante inicial, dado pelo valor zero, até o instante final que é fornecido como entrada ao simulador.

O sistema a ser testado pode ser formado por tarefas periódicas e aperiódicas. O simulador implementa uma tarefa periódica servidor para ser usada no escalonamento das tarefas aperiódicas. O simulador escolhe de forma randômica o tempo de lançamento das tarefas aperiódicas.

No desenvolvimento do simulador foi necessário recorrer a algumas simplificações a fim de proporcionar o correto escalonamento do sistema. As instâncias das tarefas não executam código real. O código das instâncias é representado por um contador de tempo de execução que é incrementado a medida que a tarefa executa. A cada unidade de tempo o escalonador escolhe uma tarefa para ser escalonada baseando-se nas condições do sistema. Quando não há nenhuma tarefa para executar o escalonador escolhe uma tarefa especial chamada *idleTask*.

O tempo na simulação é uma entidade especial manipulada apenas pelas instâncias de tarefa. Em determinado instante o tempo só pode ser manipulado pela tarefa que foi escalonada e está sendo executada. Se não há nenhuma tarefa para executar, *idleTask*, que é escalonada, tem a responsabilidade de contar o tempo. As ações do escalonador são consideradas instantâneas. O simulador não utiliza o relógio da máquina pois o próprio simulador é um processo a ser executado pelo sistema operacional hospedeiro e portanto disputa processador com os outros processos do sistema. Desta forma, em determinados momentos, o simulador estaria executando e em outros não.

Apenas uma tarefa executa por vez, aquela que foi escolhida pelo escalonador. As tarefas que não estão executando estão prontas para executar, esperando apenas serem escalonadas, ou estão inativas. As mudanças de estado sofridas pelas instâncias das tarefas são mostradas na figura 1.

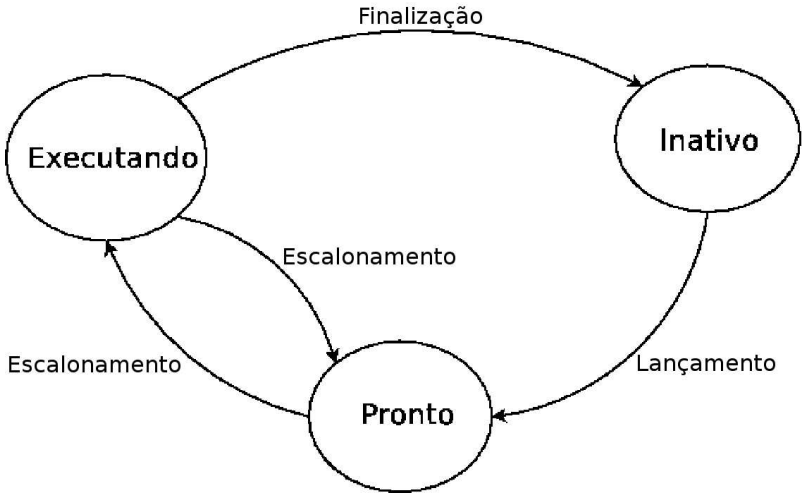

Figura 1. Estados das Instâncias

4.2. Modelo de Implementação

Este projeto foi desenvolvido dentro do paradigma de orientação a objetos. A linguagem utilizada foi Java, visto que esta oferece facilidades em sua especificação e provê portabilidade.

4.2.1. Programação *multithread*

A fim de oferecer maior realismo ao simulador foi utilizado o recurso de *threads*. Uma *thread* é uma seqüência de instruções que pertence a um programa mas é executada fora dele. *Threads* disputam o uso do processador com as demais tarefas do sistema. Cada *thread* possui uma determinada prioridade. Neste trabalho tanto o escalonador como as instâncias de tarefas foram implementados usando *threads*.

No ambiente de execução Java há um algoritmo de prioridade fixa, simples e determinístico, para escalonar as *threads* a partir de suas prioridades [thr 2005a]. Ele escalona a *thread* de mais alta prioridade e caso haja várias *threads* com a maior prioridade escalona-se uma delas arbitrariamente. Não há garantia, entretanto, que em determinado instante a *thread* em execução é necessariamente a de maior prioridade. Por esta razão o uso de prioridades não é uma garantia eficaz sobre o comportamento das *threads*.

A especificação Java possui uma classe chamada *Thread* [cla 2005] usada para programação com *threads*. Esta classe possui um método *run()* que representa a execução propriamente dita. Para fazer uso da classe *Thread* deve-se implementar uma classe que herde dela e nesta definir em seu método *run()* a seqüência de instruções a ser executada quando a *thread* ganhar o processador.

Ao implementar o simulador de escalonamento é necessário bloquear e desbloquear *threads* específicas mas os recursos oferecidos pela classe *Thread* não atendem a esta necessidade. Para contornar esta limitação foi usado um artifício que será detalhado na próxima seção.

4.2.2. Monitores

Para garantir a alternância entre escalonador e tarefa no uso do processador e garantir que em um instante qualquer não há mais de uma instância de tarefa em atividade no sistema foi usado o recurso *monitor*. Um monitor é um conjunto de procedimentos, variáveis e estruturas de dados, agrupadas num módulo especial, usadas para garantir exclusão mútua e bloqueio de processos [Tanenbaum 1999].

Em Java, monitores suportam dois tipos de sincronização de *threads*: exclusão mútua e cooperação [thr 2005b]. Neste trabalho foi usado o recurso de cooperação, que se dá via uso dos métodos *wait* e *notifyAll* da classe *Object*, na implementação de uma classe chamada *Monitor*, que é usada para bloquear e desbloquear *threads*.

4.3. Descrição das Classes

A figura 2 ilustra a estrutura do simulador.

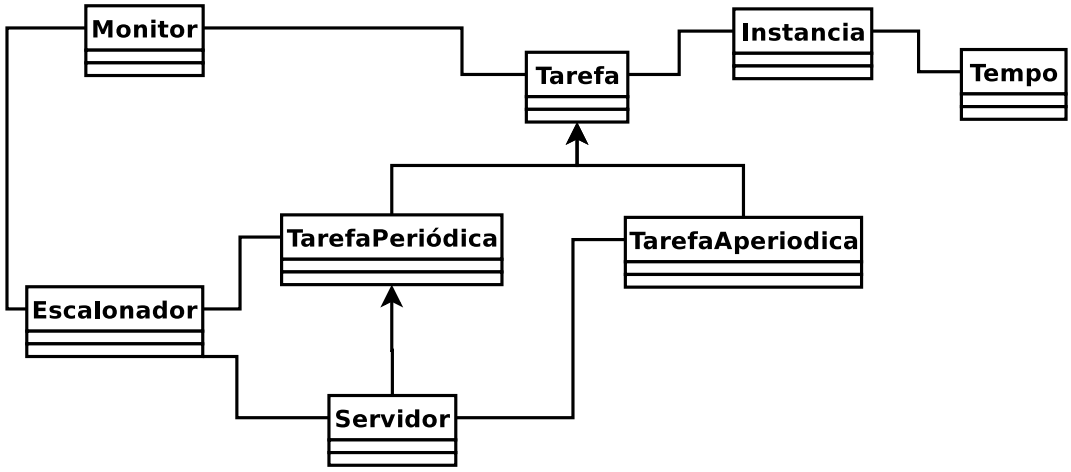

Figura 2. Estrutura do Simulador

O simulador é composto das seguintes classes:

- Tarefa: contém as propriedades básicas de uma tarefa, são elas: nome, tempo máximo de execução, estado e instância. A partir dela pode-se especificar a tarefa em periódica ou aperiódica.
- Instância: um dos atributos da classe *Tarefa*. Seu método *run()* comporta o código que será realmente executado ao escalonar determinada tarefa. Esta classe comunica-se com o monitor do sistema para liberar o processador quando não for sua vez de executar. Responsabiliza-se também por incrementar o tempo do sistema.
- TarefaPeriodica: herda os atributos da classe *Tarefa*. Possui atributos como: período, *deadline* absoluto, *deadline* relativo e número de execuções.
- TarefaAperiodica: herda os atributos da classe *Tarefa*. Possui como atributo o tempo de lançamento da instância da tarefa.
- Escalonador: possui como atributos as listas de tarefas periódicas e aperiódicas do sistema, além do servidor e de uma tarefa vazia que é escalonada sempre que não há nenhuma outra pronta para isto. Comunica-se com o monitor do sistema a fim de permitir a execução de instâncias de tarefas. Em seu método *run()* é feito o escalonamento do sistema.

- Servidor: guarda a carga do servidor
- Monitor: provê a preempção do sistema através de métodos que garantem as instâncias de tarefas o direito de executar.
- Tempo: implementa o objeto responsável pela contagem do tempo na simulação.

5. Estudo de Caso

Como estudo de caso, o simulador desenvolvido foi utilizado com um conjunto de tarefas inspirado em um pequeno estudo de caso de sistema de aviação encontrado em [Audsley et al. 1993]. A tabela 1 mostra o conjunto de tarefas usado neste caso de teste. Todas as tarefas são periódicas. Na primeira coluna tem-se um identificador de cada tarefa (*i*), na segunda coluna o tempo máximo de execução de cada uma (c_i), na terceira o período (p_i), na quarta os *Deadlines* Relativos (D_i) e nas seguintes tem-se, respectivamente, o número de *deadlines* perdidos (D_i^P), o tempo médio de resposta (R_i^S) e o tempo de resposta no pior caso (R_i^M) para cada tarefa. O teste foi realizado utilizando-se o algoritmo *Deadline-monotonic*.

Tabela 1. Experimento 1: Tarefas Periódicas do Sistema

i	c_i	p_i	D_i	D_i^P	R_i^S	R_i^M
1	51	1000	1000	0	51.000	51
2	3000	200000	5000	1	3204.000	3204
3	2000	25000	25000	0	2547.125	5306
4	5000	25000	25000	0	7802.125	10561
5	1000	40000	40000	0	3675.400	11612
6	3000	50000	50000	0	11874.750	14765
7	5000	50000	50000	0	17142.500	20071
8	8000	59000	59000	0	20448.500	35836
9	9000	80000	80000	0	30502.666	46397
10	2000	80000	80000	0	32621.666	48499
11	5000	100000	100000	0	67968.000	97998
12	1000	200000	200000	0	99100.000	99100
13	3000	200000	200000	0	140191.000	140191
14	1000	200000	200000	0	141242.000	141242
15	1000	200000	200000	0	142293.000	142293
16	3000	200000	200000	0	145446.000	145446
17	1000	100000	1000000	0	146497.000	146497
18	1000	100000	1000000	0	147548.000	147548

A tabela 2 mostra as mesmas tarefas periódicas da tabela anterior, novamente escalonadas pelo algoritmo *Deadline-monotonic*. No entanto, nesta nova simulação foram acrescentadas uma tarefa servidor do tipo *deferrable* (tarefa 19) e algumas tarefas aperiódicas. Os tempos de resposta e lançamento (*l*) das tarefas aperiódicas da simulação são mostrados na tabela 3.

Como pode-se perceber nestes dois casos de teste, o simulador é capaz evidenciar características como tempo médio e máximo de resposta e número de *deadlines* perdidos por cada tarefa. Estas informações, que não seriam obtidas por meio de um teste de escalonabilidade, podem ser usadas em avaliações mais precisas de sistemas de tempo

Tabela 2. Experimento 2: Tarefas Periódicas do Sistema

i	c_i	p_i	D_i	D_i^P	R_i^S	R_i^M
1	51	1000	1000	0	52.000	71
2	3000	200000	5000	1	3204.000	3204
3	2000	25000	25000	0	2587.125	5306
4	5000	25000	25000	0	7842.125	10561
5	1000	40000	40000	0	3731.400	11612
6	3000	50000	50000	0	11889.750	14825
7	5000	50000	50000	0	17157.500	20131
8	8000	59000	59000	0	20814.000	36927
9	9000	80000	80000	0	31100.000	47488
10	2000	80000	80000	0	33202.000	49590
11	5000	100000	100000	0	69049.500	99790
12	1000	200000	200000	0	139150.000	139150
13	3000	200000	200000	0	142303.000	142303
14	1000	200000	200000	0	143354.000	143354
15	1000	200000	200000	0	144405.000	144405
16	3000	200000	200000	0	147558.000	147558
17	1000	100000	1000000	0	148609.000	148609
18	1000	100000	1000000	0	149660.000	149660
19	20	100	100	0	-	-

real, complementando a análise de escalonamento. Por exemplo, mesmo sabendo que a tarefa 2 perde seu *deadline* em algum momento, com os dados da simulação o projetista pode decidir se a taxa de *deadlines* perdidos é admissível para o sistema desenvolvido.

6. Considerações Finais

Sistemas computacionais compõem-se essencialmente de blocos de instruções, chamados de processos, a serem executados. Em sistemas onde há mais de um processo para executar em determinado espaço de tempo, há a necessidade de um escalonador de processos para definir a ordem adequada de execução dos mesmos. Sistemas de tempo real distinguem-se de sistemas computacionais convencionais porque precisam satisfazer restrições sobre seu comportamento temporal para serem considerados corretos. Assim, cabe ao escalonador grande responsabilidade sobre a correção de tais sistemas. As restrições impostas nestes sistemas podem ser críticas, quando precisam ter garantia de atendimento, e não-críticas, quando satisfazem-se com a tentativa de melhor esforço do sistema. Neste contexto, simular as ações executadas por um escalonador usado em sistema de tempo real é importante para melhorar o conhecimento do projetista sobre o sistema antes do mesmo entrar em funcionamento.

Este trabalho descreveu um simulador projetado para analisar o comportamento de sistemas de tempo real. Tal ferramenta contempla o escalonamento de tarefas periódicas, que se repetem intervalo definido, e aperiódicas, sobre as quais não se sabe o momento em que chegam ao sistema. A análise obtida a partir de simulação demonstra o desempenho e as deficiências dos sistemas sendo simulados.

A ferramenta é simples. Porém, pela sua estrutura modular, novas funcionali-

Tabela 3. Experimento 2: Tarefas Aperiódicas do Sistema

i	c_i	l	R_i^S
20	10	14082	10.0
21	50	14082	220.0
22	300	21709	4811.0
23	550	21709	3311.0
24	30	21709	601.0
25	100	21709	411.0
26	650	82721	3189.0
27	300	120854	20.0
28	20	120854	1466.0

dades podem ser incorporadas com certa facilidade. Exemplos de possíveis extensões são a incorporação de tarefas esporádicas no modelo, o que envolve a programação do teste de aceitação para tais tarefas, e o acréscimo de métodos para definir prioridades para as tarefas aperiódicas, o que possibilitaria uma ordenação da fila de instâncias de tarefas aperiódicas conforme o critério desejado. Pode-se também disponibilizar a opção de compartilhamento de recurso entre tarefas, dotando-as de opções de bloqueio e desbloqueio sobre determinado recurso, acrescentar novos critérios de avaliação e gerar a representação da distribuição de determinado critério de avaliação.

Referências

(2005). Api specification. `http://java.sun.com/j2se/1.5.0/docs/api/`, último acesso em 13 de Dezembro de 2005.

(2005). Metasim. `http://metasim.sssup.it/`, último acesso em 13 de Dezembro de 2005.

(2005a). Thread scheduling. `http://java.sun.com/docs/books/tutorial/essential/threads/priority.html`, último acesso em 13 de Dezembro de 2005.

(2005b). Thread synchronization. `http://www.artima.com/insidejvm/ed2/threadsynch.html`, último acesso em 13 de Dezembro de 2005.

Audsley, N., Burns, A., Richardson, M., Tindell, K., and Wellings, A. J. (1993). Applying new scheduling theory to static priority pre-emptive scheduling.

Audsley, N. C., Burns, A., Richardson, M. F., and Wellings, A. J. (1994). Stress: a simulator for hard real-time systems.

Casile, A., Buttazzo, G., Lamastra, G., and Lipari, G. (1998). Simulation and tracing of hybrid task sets on distributed systems.

Kramp, T., Adrian, M., and Koster, R. (2000). An open framework for real-time scheduling simulation.

Liu, J. W. S. (2000). *Real Time Systems*. Prentice Hall.

Storch, M. F. (1997). A framework for the simulation of complex real-time systems.

Tanenbaum, A. S. (1999). *Sistemas Operacionais Modernos*. Prentice-Hall, Inc.