Universidade Federal de Pernambuco

Centro de Informática

ALGORITMOS HÍBRIDOS DE ESCALONAMENTO

TRABALHO DE GRADUAÇÃO EM SISTEMAS DE TEMPO REAL

Curso de Engenharia da Computação

Aluno: Gustavo André Fernandes Braga de Melo (gafbm@cin.ufpe.br)

Orientador: Sérgio Vanderlei Cavalcante (svc@cin.ufpe.br)

Recife, Junho de 2008

Assinaturas

Este Trabalho de Graduação é resultado dos esforços do aluno Gustavo André Fernandes Braga de Melo, sob a orientação do professor Sérgio Vanderlei Cavalcante, conduzido no Centro de Informática da Universidade Federal de Pernambuco. Todos abaixo estão de acordo com o conteúdo deste documento e os resultados deste Trabalho de Graduação.

_____

Gustavo André Fernandes Braga de Melo

_____

Sérgio Vanderlei Cavalcante

Dedicatórias

Dedico este trabalho aos meus pais, Fernando e Mercêdes, que desde o início têm juntado forças para que eu pudesse realizar meus almejos. Uma pequena retribuição ao incodicional amor que os dois sempre me deram.

Agradecimentos

Agradeço primeiramente a Deus que me fez conquistar mais esta vitória, e sem ele nada teria conseguido.

Agradeço aos meus pais e irmãos, que sempre me apoiaram e me ajudaram a chegar até aqui, seja o apoio financeiro, seja o apoio afetivo, ou até mesmo o apoio psicológico.

Ao meu orientador Sérgio Cavalcante por ter aceitado me orientar, pela paciência e confiança. Sempre com humor me dando o apoio necessário.

Ao professor Aluizio Araújo, que foi peça fundamendal na minha formação e maturação científico-acadêmica.

Agradeço aos meus amigos, que sempre estiveram do meu lado me ajudando a suportar a distância da família, mesmo tendo que aguentar os meus afastamentos devido às obrigações acadêmicas.

A todos aqueles que direta ou indiretamente contribuíram para que eu concluísse meu curso com sucesso.

Um sincero obrigado.

Resumo

Este trabalho de graduação visa estudar um dos campos mais importantes dos Sistemas de Tempo Real, escalonamento de tarefas. Um tipo específico de escalonamento é abordado, o escalonamento híbrido de tarefas. O foco do estudo é em sistemas críticos, que exigem garantias das restrições temporais em tempo de projeto. O trabalho apresenta diversos algoritmos de escalonamento híbrido, que garantem a execução e conformidades temporais das tarefas periódicas e conseguem tratar as tarefas esporádicas. Também, um método de tolerância a falhas foi proposto para permitir o tratamento seguro de interrupções para escalonamento híbrido, evitando o problema de ativação indevidamente adiantada de tarefas esporádicas.

Índice

Índice de figuras

Índice de tabelas

1. Introdução

Os sistemas computacionais estão cada vez mais presentes no cotidiano das pessoas. Grande parte dos equipamentos como eletrodomésticos, automóveis, câmeras fotográficas e MP3 players têm incorporado a utilização desses sistemas para facilitar a utilização, melhorar a qualidade e baratear os custos. Hoje em dia a dependência é tão grande que é quase impossível abrir mão desta tecnologia.

Alguns desses sistemas são utilizados em ocasiões nas quais certas restrições temporais devem ser obedecidas. Estes sistemas são chamados Sistemas de Tempo Real (STR). Controle de plantas industriais, monitoramento de ambientes, controle de tráfego aéreo, DVD player são exemplos deste tipo de aplicação. Alguns são mais complexos, como é o caso de freios ABS, e outros são mais simples, como o controle de temperatura de um condicionador de ar.

Muitas vezes o papel desses sistemas é confundido com o requisito de melhoria de desempenho. Na verdade, sistemas de tempo real garantem a execução de suas funcionalidades em prazos bem definidos. O não cumprimento de tais restrições temporais pode acarretar conseqüências catastróficas. Sistemas com essas características são conhecidos como sistemas de tempo real críticos. Por exemplo, o monitoramento de pacientes hospitalares e controle de funcionamento de turbina de um avião.

Um dos principais pontos deste tipo de sistema é o problema de escalonamento, no qual para garantir que todas as tarefas do sistema atendam suas restrições temporais e ao mesmo tempo dividam um processador é preciso ordená-las da melhor forma possível. Em situações muito críticas e com *deadlines* curtos a previsibilidade do sistema é algo importantíssimo ou até essencial, e por isso a determinação da ordem das tarefas em tempo de projeto, escalonamento *pre-runtime* é normalmente a melhor abordagem.

Em contrapartida, existem sistemas que são sensíveis a eventos do ambiente cujos tempos de ocorrência não são tão bem definidos. O escalonamento *on-line* fornece um melhor suporte para este tipo de sistema, mas elimina a previsibilidade inerente à outra abordagem. Para sistemas muito críticos isso pode se tornar um problema.

Partindo deste problema, várias técnicas visando à abordagem híbrida, que mescla algoritmos *pre-runtime* com algoritmos *on-line,* têm sido desenvolvidas. Muitos resultados

têm se mostrado bastante eficientes na garantia temporal dos sistemas e na flexibilidade do tratamento de resposta do ambiente.

1.1. Objetivo

O objetivo deste trabalho é introduzir o conceito de escalonamento híbrido de tarefas para sistemas de tempo real críticos e fazer uma análise das abordagens híbridas existentes. Neste trabalho será apresentada também uma proposta de solução híbrida para tratamento seguro de interrupções em sistemas de criticidade muito alta.

1.2. Estrutura do Documento

O capítulo 2 desta monografia introduz os conceitos básicos para o entendimento geral do trabalho e apresenta alguns termos comuns na área de escalonamento de tarefas. O capítulo 3 coloca melhor a ideia, aparentemente contrastante, de modelos *time-triggered* e *event-triggered* associados como abordagem híbrida. Apresenta também o estado da arte de algoritmos híbridos e faz uma análise dos mesmos. No capítulo 4 uma solução para o problema de ativações antecipadas de tarefas esporádicas é apresentada para um dos algoritmos descritos no capítulo 3. Uma comparação entre um algoritmo *pre-runtime* e um algoritmo híbrido é feita no capítulo 5. Também nesse capítulo, os resultados para a solução proposta no capítulo 4 são mostrados. Finalmente, o capítulo 6 traz as conclusões gerais e trabalhos futuros.

2. Conceitos Básicos

Este capítulo apresenta alguns conceitos e termos importantes para o entendimento do trabalho.

2.1. Sistemas de Tempo Real

Sistemas de tempo real (STR) são sistemas computacionais que possuem restrições temporais [26]. Normalmente são sistemas reativos (geram saídas baseados nos estímulos de entrada do ambiente) que executam suas tarefas em prazos bem definidos [27]. Isso requer dos STR uma atenção maior quando forem projetados, pois devem apresentar corretude lógica, mas também corretude temporal [24].

Muitas vezes, erroneamente, estes sistemas são confundidos com sistemas de alta performance [26]. Na verdade, STR exigem não computações mais rápidas, mas sim computações que atendam os requisitos temporais. As tarefas devem ser executadas no tempo correto e não quanto antes melhor.

Um dos pontos chaves dos sistemas de tempo real é a previsibilidade. A capacidade de conhecer o comportamento de um sistema antes de ele entrar em execução é uma característica importantíssima. Com um STR previsível pode-se saber quando e como agir para processar determinado estímulo.

O não cumprimento das restrições temporais pode acarretar diferentes efeitos dependendo do sistema. Sistemas como uma máquina de lavar ou um condicionador de ar não teriam um resultado tão danoso caso os prazos de suas atividades não fossem cumpridos. Já se as restrições em STR como controlador de tráfego aéreo ou um sistema de monitoramento e controle de uma planta nuclear não fossem atendidas os efeitos seriam catastróficos. Dessa forma classificamos os sistemas como sistemas de tempo real críticos e sistemas de tempo real não-críticos [15].

2.2. Tarefas

Tarefas, também chamadas de processos, são parte da modelagem de um sistema de tempo real do ponto de vista do problema de escalonamento [26]. São as unidades básicas de processamento seqüencial que concorrem por um ou mais recursos computacionais.

Atributos temporais

As tarefas são determinadas por alguns atributos temporais que definem o seu comportamento temporal. Para uma tarefa A temos:

- *Tempo de computação* C_A é um limitante superior para o tempo que uma tarefa leva para ser computada no processador. Pode ser encontrado na literatura como *computation time* ou WCET (*Worst Case Execution Time*).

- *Tempo de início* st_A é o instante em que uma tarefa começa a ser executada no processador.

- *Tempo de término* et_A é o instante em que uma tarefa termina sua execução.

- *Tempo de chegada* at_A é o instante em que o escalonador fica ciente da ativação de uma tarefa.

- *Tempo de liberação* rt_A é o instante em que o escalonador libera a tarefa que chegou para a fila de pronto (fila em que uma tarefa espera ser alocada para o processador).

- *Jitter máximo* J_A é o tempo máximo entre o tempo de chegada e o tempo de liberação.

- *Deadline* dl_A é o tempo limite para uma tarefa terminar sua execução. É o parâmetro mais importante em um sistema crítico.

Os atributos temporais podem ser mais bem observados na figura 2.1.

Classificações

Os diferentes efeitos de o *deadline* não ser respeitado definem dois tipos de tarefa. As tarefas críticas (*hard*) se forem finalizadas após o seu *deadline* podem causar resultados catastróficos como a destruição de um projeto muito caro ou até mesmo colocar em risco vidas humanas. Tarefas não-críticas (*soft*) são tarefas que não representam um efeito tão danoso como as anteriores se o deadline for perdido. No máximo podem acarretar na diminuição da performance do sistema. Há ainda as tarefas *deadline firm*. Essas são uma variação de tarefas não-críticas. Enquanto as *deadline soft*, mesmo estando atrasadas, devem terminar sua computação, as *deadline firm* são abortadas no caso de perderem o *deadline*.

Outra forma de classificar as tarefas de um STR é quanto ao tempo de ativação. Elas podem ser periódicas ou aperiódicas. Uma tarefa periódica P é aquela cuja ativação acontece sempre em um intervalo de tempo regular chamado período T_P. É determinada por seu período, tempo de computação e *deadline*. Tarefas aperiódicas são ativadas, normalmente por estímulos externos, em intervalos aleatórios e são definidas por seu tempo de computação e *deadline*. Dentro das tarefas aperiódicas existe o subgrupo de tarefas esporádicas, no qual uma tarefa esporádica S possui ativação aleatória, mas é conhecido um tempo mínimo entre ativações T_S. Determina-se uma tarefa esporádica por tempo mínimo entre chegadas, tempo de computação e *deadline*.

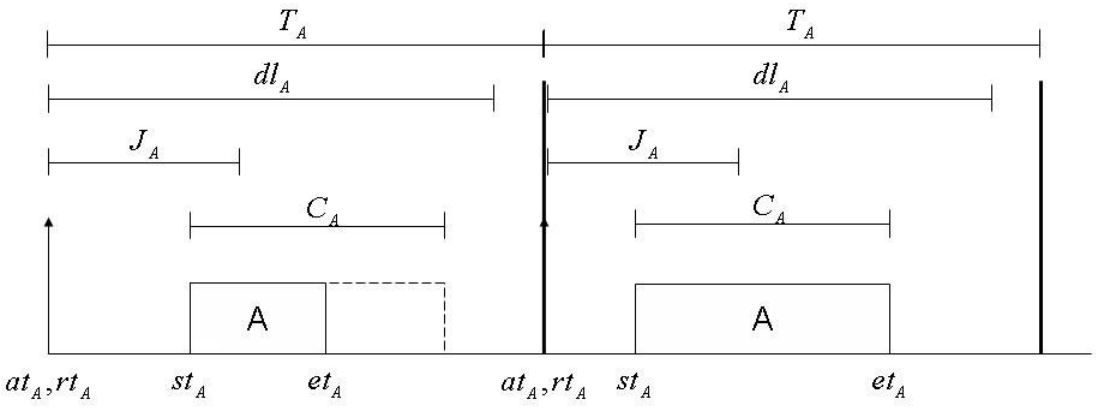

Figura 2.1 – Atributos temporais de uma tarefa A

Relações de restrição

Além dos atributos temporais listados acima, as tarefas podem ter restrições de sincronização com outras tarefas. Essas restrições também influenciam na ordem de execução das tarefas, assim como *deadline* e tempo de computação.

Precedência – Às vezes uma tarefa depende de dados ou resultados de uma outra tarefa. Também, uma tarefa pode ser dividida em duas com a exigência de a primeira parte ser executada antes da segunda. Esta situação trás o conceito de precedência, que define que uma tarefa só pode iniciar sua execução depois que a outra acabar.

Exclusão mútua – Outra relação importante entre tarefas é a relação de exclusão. Quando duas tarefas compartilham o mesmo recurso, além do processador, uma não pode iniciar sua execução enquanto a outra estiver em andamento. Isto quer dizer que elas se excluem mutuamente.

2.3. Escalonamento

Escalonamento é o problema de alocar as tarefas aos recursos computacionais do sistema de forma a satisfazer todos os requisitos temporais [24]. O resultado do processo de escalonamento é uma escala que indica a ordem de ocupação do processador e acesso a outros recursos [26]. O escalonador é, então, o componente responsável pela gestão de utilização dos recursos computacionais.

2.3.1. Classificações

Existem três tipos principais de classificação que podem ser utilizados para distinguir um escalonamento.

- As propriedades das tarefas definem dois tipos de escalonamento: estático e dinâmico. Quando os parâmetros das tarefas são conhecidos em tempo de projeto o escalonamento é estático. Normalmente os parâmetros definem a prioridade da tarefa, como, por exemplo, o período da tarefa. No escalonamento dinâmico os parâmetros mudam em tempo de execução.
- Há duas abordagens para escalonamento que tomam como referência o momento em que a escala é montada [24]. Escalonamento on-line tem a característica de montar a escala em tempo de execução. O escalonador decide, durante a execução, qual tarefa da fila de pronto entra em execução. A abordagem on-line tem a vantagem de ser flexível e adaptar-se às mudanças do ambiente, no que diz respeito à ativação de tarefas. Já o escalonamento off-line, também conhecido como escalonamento *pre-runtime*, toma as decisões de ordem de processamento das tarefas antes delas entrarem em execução. É criada uma tabela de despacho, em tempo de projeto, para o despachante, que funciona em tempo real e aloca as tarefas na ordem e instante descritos na tabela. Para ser possível, esse tipo de escalonamento precisa saber *a priori* os atributos das tarefas sobre restrições e relações. Por isso os escalonamentos off-line são também estáticos. Tem a vantagem de os sistemas que utilizam esta técnica para gerar suas escalas serem totalmente previsíveis. Como vimos esta é uma característica importantíssima em um sistema de tempo real crítico.
- Existe ainda outra classificação de escalonadores: preemptivos e não-preemptivos. Escalonamento preemptivo permite que uma tarefa de maior

prioridade interrompa uma outra de menor prioridade. Nos escalonadores não-preemptivos uma tarefa depois de iniciada só é desalocada do processador após seu término. Tanto escalonamentos on-line como off-line podem ser preemptivos.

3. Escalonamento Híbrido

Neste capítulo um breve conceito de escalonamento híbrido é introduzido e o estado da arte é apresentado com a descrição de algumas abordagens.

3.1. Time-triggered *vs.* Event-triggered

Em um sistema puramente *event-triggered* (ET) todas as atividades são iniciadas pela ocorrência de eventos. Neste caso, evento é significa toda mudança de estado do ambiente que precisa ser tratada pelo sistema computacional [21]. Normalmente, em sistemas embarcados de tempo real, os eventos são sinalizados por meio de mecanismos de interrupção do processador.

Os sistemas ET são dirigidos a demanda e geralmente são apoiados por uma estratégia de escalonamento dinâmica. Possuem uma grande flexibilidade para processar mudanças no ambiente, já que as tarefas são ativadas por eventos. Esta é uma grande vantagem para sistemas com tarefas de tempo de resposta curto, ou seja, tarefas que possuem *deadlines* apertados, visto que estas tarefas podem iniciar sua execução quase que instantaneamente após o instante de chegada. Outro ponto forte é a maior utilização do processador. Já que as decisões de escala são tomadas *on-line*, não é necessário esperar o tempo máximo de computação de uma tarefa para começar a executar a outra.

Entretanto, este paradigma possui limitações quanto às relações de restrição entre as tarefas. Relações de precedência e exclusão mútua muito complexas fazem com que a decisão de escalonamento seja muito custoso para ser feito em tempo de execução [21]. O *overhead*, custo adicional em processamento ou armazenamento que, como conseqüência, piora o desempenho de um programa, também é um fator negativo. Essa característica existe em sistemas ET pelas preempções e cálculos de escalonamento exigidos pela abordagem.

O conceito de *time-triggered* (TT) é centrado na exatidão temporal [22]. Sistemas TT têm suas tarefas ativadas em instantes de tempo bem definidos. Essa precisão, obtida em tempo de projeto, faz com que esse tipo de sistema tenha uma previsibilidade muito grande. Por isso é muitas vezes preferido à abordagem *event-triggered* para modelar sistemas críticos.

Outra característica inerente a este modelo é o baixo *overhead* causado por sistemas *time-triggered*. Como as relações de restrição e requisitos temporais são resolvidos pelo

algoritmo *pre-runtime*, os custos referentes à decisão de escalonamento durante a execução do sistema é nulo. Apenas pequenos custos referentes ao despacheante *on-line*, que ativa as tarefas baseado em uma tabela gerada *off-line*, são considerados no atraso entre tarefas. Relações de restrição complexas são melhores tratadas por escalonamentos *off-line*, como explicado em [25].

Em compensação, sistemas TT possuem a utilização do processador mais baixa que em sistemas ET. Isto se dá pelo fato de o escalonador baseado em *time-triggered* usar o pior caso de computação das tarefas para gerar as escalas. Se uma tarefa concluir sua execução antes do tempo máximo de término a próxima tarefa terá de esperar para ser ativada. Além disso, este paradigma lida de forma muito pobre com tarefas esporádicas com *deadlines* curtos. Como será mostrado na técnica de Mok [16] na próxima seção, para tratar este tipo de tarefa é preciso alocar um grande número de *slots* de tempo, diminuindo ainda mais a utilização do processador e aumentando o *overhead*.

Embora estas abordagens sejam bem diferentes, elas não são excludentes [23]. Os algoritmos híbridos tentam unir o melhor de cada mundo para tratar de conjuntos de tarefas mistas, periódicas e aperiódicas, que também podem ser enxergadas como *time-triggered* e *event-triggered*. A previsibilidade de sistemas TT agregada à flexibilidade dos modelos ET fornece, quase sempre, uma maneira segura e eficiente de projetar sistemas críticos. A junção das abordagens pode resolver o problema de tarefas aperiódicas para algoritmos TT, tendo um sistema mais flexível, e melhorar a escalonabilidade de técnicas ET, resolvendo relações de restrição complexas.

Na seção seguinte serão apresentados alguns algoritmos com diferentes técnicas e características para o escalonamento de tarefas periódicas (*time-triggered*) e tarefas esporádicas (*event-triggered*).

3.2. Algoritmos Híbridos Existentes

3.2.1. Mok, 1984

Esta é a abordagem mais simples para escalonar tarefas periódicas e esporádicas no mesmo sistema. Consiste em modelar as tarefas esporádicas como periódicas e usar um escalonador *pre-runtime* para gerar a escala de execução [15,16].

Modelagem do sistema

As tarefas periódicas são definidas por suas propriedades temporais:

- período,

- tempo de computação,

- tempo de liberação, e

- *deadline*.

Pelo fato de o escalonamento ser totalmente *pre-runtime*, estas tarefas podem ter todos os tipos de relação de restrição descritos no capítulo anterior.

As tarefas esporádicas não possuem um tempo de chegada bem definido, por isso não se pode dizer quando elas serão liberadas, porém é conhecida *a priori* a freqüência máxima de chegada destas tarefas, que são caracterizadas por:

- tempo mínimo entre chegadas,

- tempo de computação, e

- *deadline*.

Algoritmo

Primeiramente as tarefas esporádicas são modeladas como tarefas periódicas. Cada tarefa esporádica S é substituída por uma tarefa periódica P que tem um período curto o suficiente para conseguir garantir o cumprimento do *deadline* de S, e cujos atributos são os seguintes [15, 16]:

$$rt_P = 0$$

$$C_P = C_S$$

$$dl_P = C_S$$

$$T_P = min(T_S, dl_S - C_S + 1).$$

Depois o novo conjunto de tarefas periódicas é escalonado com um algoritmo *off-line* e a escala de execução é obtida.

Avaliação

Esta abordagem é muito simples, permite relações de restrição complexas e é totalmente previsível, ótimo para sistemas críticos. Contudo, uma tarefa P originada de uma tarefa esporádica S com *deadline* apertado causa um *overhead* muito grande devido ao curto período da tarefa periódica, tornando o sistema inviável. Além disso, a utilização do processador diminui consideravelmente, já que a tarefa P é ativada várias vezes entre os eventos referentes à tarefa S.

3.2.2. Servidor Esporádico, 1989

Esta técnica utiliza um servidor periódico para tratar das requisições aperiódicas [17]. O servidor é escalonado junto com as outras tarefas periódicas. Quando a tarefa servidora é executada ela trata a fila de tarefas aperiódicas requisitadas, ordenadas independentemente da fila de tarefas periódicas. Esta abordagem é mais utilizada em sistemas de escalonamento de prioridade fixa (*on-line*) [5,14].

Modelagem do sistema

As tarefas periódicas são determinadas por:

- período,
- tempo de computação, e
- *deadline*.

O servidor é uma tarefa SS caracterizada por:

- período, e
- capacidade máxima C_{SS}, análoga ao tempo de computação.

A capacidade máxima indica o máximo de unidades de tempo por período que o servidor pode utilizar para tratar as tarefas aperiódicas pendentes. Normalmente é atribuída uma alta prioridade à tarefa servidora.

Algoritmo

Quando uma tarefa aperiódica chega é colocada na fila de pendentes. No momento em que a tarefa servidora for ativada, iniciar a execução, a carga aperiódica na fila é

executada e o tempo consumido (no máximo C_{SS}) é decrementado da capacidade do servidor. Se não existem mais tarefas aperiódicas pendentes a capacidade atual é mantida e aguarda-se a próxima requisição. O tempo restante é utilizado para execução das tarefas periódicas. A restauração da capacidade do servidor é descrita como segue:

p_{exe} é a prioridade da tarefa que está sendo executada.

p_{SS} é a prioridade da tarefa servidora.

intervalo ativo é o espaço de tempo na escala no qual $p_{SS} \leq p_{exe}$.

intervalo ocioso é o tempo em que $p_{SS} > p_{exe}$.

RT_{SS} é o instante em que a capacidade da servidora é restaurada.

Se a tarefa servidora entrar no *intervalo ativo* no instante t_A e alguma capacidade for consumida durante o *intervalo ativo* a restauração ocorrerá no tempo $RT_S = t_A + T_S$. A quantidade de capacidade a ser restaurada é a quantidade de capacidade consumida neste *intervalo ativo*.

Avaliação

Este algoritmo apresenta um bom tempo de resposta às tarefas aperiódicas, comparado com outras abordagens *on-line*, mas não é rápido o suficiente para tratar tarefas com curtíssimo tempo de resposta. Esta técnica aumenta a utilização do processador por fazer uso de escalonamento *event-triggered*, porém causa um *overhead* grande e não permite tarefas periódicas com relações de restrição muito complexas.

3.2.3. Sandström *et al.*, 1998

Esse algoritmo, [2], apresenta uma maneira de integrar escalonamento *pre-runtime* de tarefas periódicas e tratamento de interrupções com um tempo mínimo entre chegadas. São aplicadas técnicas de escalonamento *off-line* junto com análise de tempo de resposta [1].

A ideia chave do algoritmo é preemptar a execução de uma tarefa periódica quando ocorrer uma interrupção, executar a rotina de tratamento de interrupção e retornar depois para a tarefa anterior. Porém os atrasos causados pelas interrupções às tarefas periódicas, na situação de pior caso, são adicionados no cálculo do tempo de resposta destas tarefas, o que permite uma garantia em tempo de projeto do cumprimento dos requisitos temporais do sistema.

O algoritmo pode ser aplicado em escalas já prontas ou acoplado no escalonador para construção de escalas.

Modelagem do sistema

Cada tarefa periódica é especificada pelo:

- período,
- tempo de liberação,
- *deadline*, e
- tempo de computação.

Também, essas tarefas podem ter relações de precedência, exclusão mútua e relações de comunicação entre elas.

As interrupções são definidas por:

- tempo de computação,
- *deadline*, e
- tempo mínimo entre chamadas.

A escala é representada por cadeias (*chains*), no qual as tarefas de cada cadeia são executadas em seqüência. Cada cadeia tem um tempo de início st independente do término das outras cadeias, o que permite preempção entre cadeias. A figura 3.1 mostra como a escala é organizada em cadeias. Existem três cadeias: cadeia1 com três tarefas e tempo de início 0, cadeia2 com duas tarefas e tempo de início 1000 e cadeia 3 com duas tarefas e tempo de início 3000. É papel do despachante *on-line* iniciar a execução de cada cadeia.

Algoritmo

Primeiro um escalonador *pre-runtime* cria uma escala, baseada na representação por cadeias, para as tarefas periódicas. A *n-ésima* tarefa de uma cadeia pode ser modelada como uma tarefa, cujo tempo de computação é a soma dos tempos de computação das *n* tarefas. Então, calcula-se o tempo de término da tarefa com a análise do tempo de resposta exato [1]. As interrupções são consideradas tarefas de maior prioridade e o tempo crítico para a análise é o instante de início da cadeia, considerando que todas as tarefas da cadeia são ativadas ao mesmo tempo.

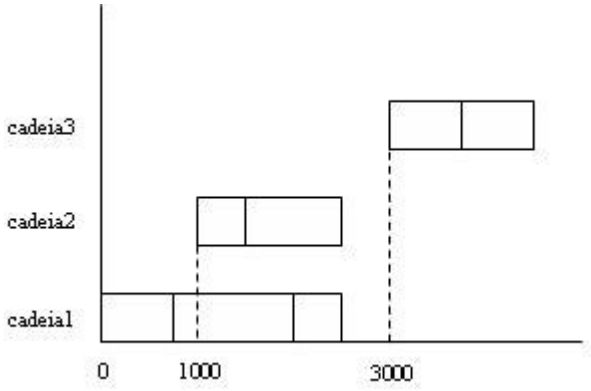

Figura 3.1 - Representação da escala em cadeias

Porém, levando em conta a preempção entre as cadeias, a análise fica mais complicada. O atraso imposto na cadeia preemptada pela cadeia preemptante tem de ser considerado. Os tempos de computação de todas as tarefas da cadeia preemptante são adicionados no cálculo, já que a tarefa preemptada só retorna à execução depois que toda a cadeia preemptante atingir seu fim. Conseqüentemente a interferência das interrupções sobre esta cadeia influencia a análise do tempo de resposta da tarefa sob análise.

O cálculo do tempo de resposta de pior caso da tarefa i na cadeia ch é dado então por:

$$R_i = \sum_{n=1}^{i} C_{ch,n} + \sum_{\forall p \in preemp(R_i)} \sum_{m=1}^{numTarefas(p)} C_{p,m} + \sum_{\forall \text{interrupcao}} \left\lceil \frac{R_i}{T_{\text{interrupcao}}} \right\rceil C_{\text{interrupcao}}$$

, onde:

$C_{a,b}$ é o tempo de computação da tarefa b na cadeia a.

$preemp(R_i)$ é o conjunto de todas as cadeias p no qual $p \neq ch$ e $st(ch) < st(p) \leq (R_i + st(ch))$, que significa todas as cadeias que preemptam i ou alguma de suas tarefas predecessoras em ch.

$numTarefas(p)$ é o número de tarefas na cadeia p.

Se o tempo de resposta de pior caso de todas as tarefas for menor ou igual ao seu deadline então a escala é válida.

O pseudo-código do algoritmo, cujo objetivo é calcular o tempo de resposta da tarefa i da cadeia ch com $st(ch) = t$, é o seguinte:

interferencia – é a soma dos tempos de computação de todas as tarefas que interferem em i.

hCadeias – são todas as cadeias no qual o tempo de início st é maior que t.

1. *interferencia* = tempo de computação de i + a soma dos tempos de computação de todas as tarefas que precedem i na cadeia ch.
2. $R_i(0) = interferencia$
3. para cada cadeia h_{cadeia} em *hCadeias*:

   se $R_i(n) + t > st(h_{cadeia})$ então

   $interferencia = interferencia$ + soma dos tempos de computação de todas as tarefas em h_{cadeia}.

   $R_i(n) = R_i(n)$ + soma dos tempos de computação de todas as tarefas em $h_{cadeias}$.

   remove h_{cadeia} de *hCadeias*.

4. $R_i(n+1) = interferencia + \sum_{\forall \text{interrupt}} \left\lceil \frac{R_i}{T_{\text{interrupt}}} \right\rceil C_{\text{interrupt}}$
5. se $R_i(n+1) > dl_i$ então

   aborte

   senão se $R_i(n+1) = R_i(n)$ e $R_i(n+1) + t$ for maior que o tempo de início de todas as cadeias em *hCadeias* então

   $R_i(n+1)$ é o tempo de resposta de pior caso da tarefa i.

   senão

   vá para o passo 3.

Obs.: n é o número da iteração do passo 3.

Se o algoritmo é utilizado para validar uma escala já pronta basta seguir os passos descritos. Caso deseje-se aplicá-lo na fase de escalonamento para construir uma escala existe uma alteração no processo do cálculo. Algumas cadeias no passo 3 podem ainda não existir no início do algoritmo. À medida que o escalonamento for prosseguindo as cadeias vão surgindo. Se uma nova cadeia que preempta a tarefa i for criada então o

algoritmo deve ser aplicado a esta cadeia de forma hierárquica. Primeiro aplica-se o algoritmo à nova cadeia. Depois, aplica-se à tarefa i e suas sucessoras. Em seguida à tarefa j, que é preemptada pela cadeia de i, e suas sucessoras. E assim por diante.

Avaliação

Já que esta abordagem utiliza um escalonamento *off-line* para alocar as tarefas, estas podem assumir relações de restrições complexas. Pelo fato de permitir interrupções, algumas tarefas esporádicas com deadline curto podem ser modeladas como interrupções. Entretanto, este algoritmo não aceita tarefas esporádicas com prioridade mais baixa que das tarefas periódicas.

Outro ponto forte é o de *overhead* diminuído. O fato de se fazer uso do paradigma de escalonamento *pre-runtime* já causa esse efeito. A tabela de execução já está pronta antes do sistema começar a executar e o despachante on-line só precisar atuar para iniciar uma nova cadeia.

Contudo, assumir o cenário de pior caso com as interrupções sendo ativadas em suas freqüências máximas acarreta num desperdício de recurso, e assim uma menor utilização do processador.

3.2.4. Isovic and Fohler, 1999

Este algoritmo trata de tarefas esporádicas em sistemas *time-triggered* distribuídos [18]. Entretanto, não focaremos na característica de multiprocessamento desta técnica. Uma escala gerada por um escalonador *pre-rumtime* é analisada junto com um conjunto de tarefas esporádicas, permitindo uma garantia em tempo de projeto dos cumprimentos dos requisitos temporais das tarefas. Esta abordagem consiste de uma parte *off-line* que testa a escalonabilidade e uma parte *on-line* que decide qual tarefa será executada e cuida da manutenção das capacidades de reserva, ou *spare capacities*. O algoritmo permite também tratamento de tarefas aperiódicas não-críticas, aumentando a utilização prática do processador.

Esta abordagem é baseada no método de *slot shifting* [6], que utiliza os espaços de tempo não utilizados na escala *off-line* para alocar as tarefas esporádicas. As tarefas escalonadas *off-line* são deslocadas no tempo sem violar suas restrições temporais.

Modelagem do sistema

Uma tarefa periódica P é caracterizada por:

- tempo de computação,
- período, e
- *deadline*.

A *k-ésima* instância da tarefa P é dada por P_k, que possui ainda o atributo tempo de início mais cedo possível est_{Pk}. O conceito de tempo de início mais cedo possível existe pelo fato de o algoritmo deslocar no tempo o início real da execução. As tarefas periódicas possuem também relações de restrição e tempos de troca de mensagens, já que se trata de um sistema distribuído.

Uma tarefa esporádica S é determinada por:

- tempo mínimo entre chegadas,
- tempo de computação, e
- *deadline*.

As tarefas aperiódicas A possuem apenas a informação do tempo de computação.

O tempo é discretizado em *slots* definidos pelo *tick* de tempo do sistema. Os slots têm o mesmo tamanho, que é determinado pelo intervalo de *tick.* Os parâmetros das tarefas são todos definidos como múltiplos do tamanho do *slot*.

Algoritmo

Esta técnica de escalonamento realiza primeiramente uma preparação *off-line*. Como descrito em [6], um escalonador *off-line* cria uma tabela de escalonamento das tarefas periódicas, resolvendo as relações de restrição e ordenando a execução das tarefas. Esta tabela de escalonamento fixa o tempo de início e o tempo de término apenas de tarefas que não são flexíveis, como tarefas que enviam ou recebem alguma mensagem e tarefas que iniciam ou terminam a seqüência de execução. Todas as outras têm seus atributos temporais definidos recursivamente, já que a suas execuções podem variar dentro da ordem de precedência. Variação que chamamos de *shifting*.

Os *deadlines* absolutos das tarefas ($est + dl$) são então ordenados e cada um deles define o fim do que é dito como intervalo de execução disjunta I. Todo intervalo inicia-se no final

do anterior e o primeiro começa no instante zero. Um intervalo possui uma capacidade de reserva $sc(I)$ que representa o tempo de sobra que existe no determinado intervalo para ser liberado à execução de uma tarefa esporádica ou aperiódica. A equação da capacidade de reserva é dada por:

$$sc(I_i) = |I_i| - \sum_{T \in I_i} C_T + \min(sc(I_{i+1}),0)$$

, onde:

i é o número de ordem do intervalo,

$|Ii|$ é a quantidade de *slots* no intervalo,

$\sum_{P \in I} C_P$ é a soma dos tempos de computação das tarefas pertencentes ao intervalo,

e

$min(sc(I_{i+1}), 0)$ significa que tarefas no intervalo subseqüente podem iniciar a execução antes, tomando emprestado algum tempo da capacidade de reserva do intervalo I_i.

Até então só foi falado de tarefas periódicas. As tarefas esporádicas críticas tem de ter garantia em tempo de projeto. As outras tarefas aperiódicas são não-críticas e podem ser executadas ou não, dependendo da capacidade de reserva disponível. Um algoritmo de garantia *off-line* é aplicado ao conjunto de tarefas esporádicas como teste de aceitação. As tarefas esporádicas são ditas ser realizáveis em conjunto com as tarefas periódicas já escalonadas se passar pelo teste.

O algoritmo de garantia verifica os piores casos de chegada para cada tarefa. Cada intervalo possui apenas um instante crítico e cada tarefa é testada em todos os instantes críticos do sistema. Define-se o instante crítico $t_c(I_i)$ de um intervalo como o *slot* no qual uma tarefa esporádica tem seu início atrasado ao máximo pela execução de uma tarefa periódica (escalonada *off-line*). A equação é dada por:

$$t_c(I_i) = st(I_i) + sc(I_i)$$

, onde $st(I_i)$ é o *slot* de início do intervalo I_i.

O algoritmo de garantia *off-line* assume que as tarefas esporádicas chegam com a sua freqüência máxima e considera os piores casos de chegada, como mostrado a seguinte.

Para um conjunto Ψ de tarefas esporádicas, o teste é descrito pelo seguinte pseudo-código:

i é o índice do intervalo no qual a tarefa chega.

k é o índice do intervalo que possui o *slot* de deadline da tarefa.

sc_a é a capacidade de reserva disponível para uma tarefa esporádica S entre sua chegada e seu *deadline* absoluto.

R é um registro dos *slots* reservados para tarefas esporádicas que foram testadas anteriormente.

iniciaR() inicia R vazio.

qntR(x,y) indica o número de *slots* reservados entre o *slot* x e y.

reservaR(n, d) reserva n *slots* o mais próximo possível do *slot* d.

1. para cada instante crítico (cada intervalo):
2. *iniciaR()*
3. para cada tarefa esporádica S do conjunto Ψ:
4. para cada instância S_n da tarefa S dentro do mínimo múltiplo comum dos tempos mínimos entre chegadas das tarefas do conjunto Ψ:
5. $$sc_a(S_n) = \sum_{j=i+1}^{k-1} sc(I_j) + min(sc(I_k), at_{Sn} + dl_S - st(I_k)) - qntR(at_{Sn}, at_{Sn} + dl_S)$$
6. se $sc_a(S_n) \geq C_S$ então

   $$reservaR(C_S, at_{Sn} + dl_S)$$
7. caso contrário o conjunto é rejeitado.

Durante o tempo de execução o escalonador *on-line* é invocado após cada *slot* de tempo. Ele tem a função de decidir que tarefa será executada no próximo *slot*. Esse mecanismo *on-line* é descrito da seguinte forma:

t é o tempo atual.

$sc(I)_t$ é a capacidade de reserva do intervalo atual no tempo t.

$P(t)$ é o conjunto de tarefas prontas, todas as tarefas que possuem $est \leq t$.

1. Se $P(t)=\{\}$ não há nenhuma tarefa pronta para executar. O sistema fica em *idle* e um *slot* da capacidade de reserva do intervalo é consumido.

2. Se $P(t) \neq \{\}$, $\exists A \mid A$ é uma tarefa aperiódica não-crítica. Pode haver os seguintes subcasos:

   a. $sc(I)_t > 0$, $\exists S \in P(t)$ então S é executada e a capacidade de reserva do intervalo é decrementada.

   b. $sc(I)_t > 0$, $\neg\exists S \in P(t)$ executa-se A. Um *slot* da capacidade de reserva do intervalo é utilizado.

   c. $sc(I)_t = 0$ indica que uma tarefa periódica deve ser executada.

3. $P(t) \neq \{\} \neg\exists A \mid A$ é uma tarefa aperiódica não-crítica. Utiliza-se EDF [5] para escolher a tarefa a ser executada. Caso seja uma tarefa esporádica, um *slot* da capacidade de reserva é diminuído.

Avaliação

Este paradigma de escalonamento híbrido apresenta uma forma de aproveitar uma melhor utilização do processador, visto que os espaços livres na escala *off-line* são usados para escalonar tarefas esporádicas críticas, podendo também tratar de tarefas aperiódicas não-críticas quando é possível.

Entretanto, este algoritmo é apenas reativo, ou seja, é aplicado somente a escalas estáticas já prontas, e não na construção de escalas. Além disso, o mecanismo apresentado aqui possui um *overhead* muito alto, já que o escalonador *on-line* atua em cada *slot* de tempo. Se o tamanho do *slot* for aumentado para diminuir o *overhead* do sistema tarefas com *deadline* mais apertado tornam o sistema não-realizável.

3.2.5. Mäki-Turja, 2005

Em [19] é apresentado um método híbrido de escalonamento, *off-line* e *on-line*, utilizando o modelo com *offsets* de [9] para análise do tempo de resposta das tarefas escalonadas dinamicamente. A ideia é escolher o modelo de escalonamento não para o sistema no todo, mas para cada tarefa. Tarefas com relações de restrição mais complexas ou com maior necessidade de previsibilidade são escalonadas estaticamente. Tarefas mais independentes ou que respondam a mudanças no ambiente (*event-triggered*) são escalonadas em tempo de execução. Como se trata de sistemas críticos, mesmo as

tarefas dinâmicas têm garantias de tempo de resposta em tempo de projeto. Interrupções também são permitidas nesta abordagem.

Modelagem do sistema

Um conjunto de tarefas Γ é composto de k transações, $\Gamma_1,...,\Gamma_k$. Cada transação Γ_i é ativada em uma taxa regular igual ao período T_i. Uma transação Γ_i contém $|\Gamma_i|$ tarefas, e cada tarefa é ativada depois de um tempo de *offset* relativo ao início da transação a qual ela pertence.

Na tarefa τ_{ij} o índice i indica qual a sua transação, e o índice j é o número da tarefa dentro da transação. Esta tarefa possui algumas propriedades definidas por:

- tempo de computação C_{ij},
- *offset* O_{ij},
- *deadline* dl_{ij},
- *jitter* J_{ij},
- tempo máximo de bloqueio B_{ij} e
- prioridade P_{ij}.

O atributo tempo máximo de bloqueio é o tempo de pior caso no qual uma tarefa de maior prioridade é bloqueada por uma tarefa de menor prioridade usando um recurso compartilhado pelas duas tarefas. Este modelo de tarefa apresentado em [19] traz este conceito mas não focaremos esta característica neste trabalho.

As interrupções possuem vários níveis de prioridade entre si, mas todas têm maior prioridade que tarefas estáticas e tarefas dinâmicas. Cada interrupção é modelada como uma transação Γ_i de período T_i igual ao tempo mínimo entre chegadas da interrupção, e com uma única tarefa τ_{i1} correspondente à interrupção.

As tarefas estáticas são todas tarefas periódicas com *jitter* nulo, organizadas em uma transação Γ_E com período T_E igual ao mínimo múltiplo comum (LCM) dos períodos de cada tarefa. Os tempos de bloqueio das tarefas de Γ_E também são nulos, já que as tarefas são escalonadas *off-line*. Uma tarefa estática é derivada em várias instâncias, suficientemente para preencher toda a escala de Γ_E e cada instância é uma tarefa τ_{Ej} com *offset* O_{Ej} relativo ao início da transação.

O conjunto de tarefas dinâmicas é composto de tarefas esporádicas e/ou tarefas periódicas. Cada tarefa é modelada em uma transação Γ_D com período T_D igual ao período ou tempo mínimo de chegada, para tarefa periódica ou esporádica respectivamente. O tempo de bloqueio B_{Dj} pode ser calculado com protocolos de bloqueio de recursos como protocolo de prioridade de teto apresentado em [10]. Os outros atributos temporais, como *jitter* e *offset*, são definidos de acordo com as especificações de projeto das tarefas. Estas tarefas possuem menor prioridade que interrupções e tarefas estáticas. Assume-se que as tarefas dinâmicas não se comunicam com tarefas estáticas via recursos compartilhados para não afetarem o comportamento temporal destas últimas.

Algoritmo

Primeiro uma escala *off-line* é construída para as tarefas estáticas utilizando um escalonador *pre-runtime*. Esse escalonamento leva em consideração as interferências das interrupções. Para isso utiliza-se a técnica de análise de tempo de resposta descrita por Sandström *et. al* em [2].

Com a escala estática pronta aplica-se o algoritmo do cálculo do tempo de resposta apresentado em [9] às tarefas dinâmicas. Se o tempo de resposta de uma tarefa é menor ou igual ao seu *deadline* então a tarefa é garantida para este sistema crítico.

As análises de tempo de resposta clássicas, como em [1] assumem que o instante crítico acontece quando todas as tarefas são liberadas ao mesmo tempo. Porém esse modelo é muito pessimista. Análise de tempo de resposta com *offset* utiliza o fato de que as tarefas não são liberadas ao mesmo tempo, aproveitando os intervalos ociosos do processador e obtendo tempos de resposta menores.

A análise de tempo de resposta com *offset* parte do conceito de que pelo menos uma tarefa em cada transação Γ_i (não é a transação de interrupção descrita acima) é liberada no instante crítico [12]. Como a tarefa que coincide com o instante crítico não é conhecida, cada tarefa da transação deve ser examinada como a tarefa τ_{ic} liberada no instante crítico.

Como descrito em [9], a ideia central é contabilizar a interferência causada por uma tarefa τ_{ij} de maior ou igual prioridade que da tarefa sob análise τ_{ua} durante um intervalo de tempo t. Dependendo do período da transação que a contém, uma tarefa pode interferir várias

vezes dentro do intervalo t com as suas instâncias. As instâncias que interferem τ_{ua} são classificadas em dois conjuntos:

Set1, cujas instâncias são ativadas antes ou no instante crítico, mas, pelo atraso causado pelo *jitter*, podem coincidir o tempo de liberação com o instante crítico.

Set2, cujas instâncias são ativadas depois do instante crítico.

A análise de tempo de respostas de tarefas com *offset* é baseada em dois teoremas fundamentais [13]:

- o pior caso de interferência que uma tarefa τ_{ij} causa a τ_{ua} é quando as instâncias de *Set1* têm um *jitter* tal que todas elas são liberadas no instante crítico, e as instâncias de *Set2* têm *jitter* nulo.

- a tarefa τ_{ic} que coincide com o instante crítico faz isso depois de ser atrasada pelo *jitter* máximo, ou seja, o instante crítico é o instante de liberação da tarefa assumindo o *jitter* máximo.

Daí obtém-se o conceito de fase entre uma tarefa τ_{ij} e o instante crítico, explicado em [9], que é dado pela equação:

$$\phi_{ijc} = (O_{ij} - (O_{ic} + J_{ic})) \bmod T_i$$

Depois de definir os conjuntos *Set1* e *Set2* de instâncias e calcular as suas fases relativas ao instante crítico, pode-se calcular a interferência causada pela tarefa τ_{ij}. O cálculo da interferência é dividido em duas partes:

I_{ijc}^{Set1}, é a parte causada pelas instâncias do conjunto *Set1*.

I_{ijc}^{Set2}, é a parte causada pelas instâncias do conjunto *Set2* e é função do intervalo de tempo t.

São definidas pelas equações:

$$I_{ijc}^{Set1} = \left\lfloor \frac{J_{ij} + \phi_{ijc}}{T_i} \right\rfloor C_{ij}$$

$$I_{ijc}^{Set2}(t) = \left\lceil \frac{t - \phi_{ijc}}{T_i} \right\rceil C_{ij}$$

O conceito de “interferência imposta” apresentado por [9] é uma forma mais eficiente da análise mostrada em [13]. A equação de I_{ijc}^{Set2} é mudada para:

$$I_{ijc}^{Set2}(t) = \left\lceil \frac{t - \phi_{ijc}}{T_i} \right\rceil C_{ij} - x$$

$$x = \begin{cases} C_{ij} - ((t - \phi_{ijc}) \bmod T_i), \text{se } t > \phi_{ijc} \text{ e } (0 < ((t - \phi_{ijc}) \bmod T_i) < C_{ij}) \\ 0, \text{caso contrário} \end{cases}$$

A interferência que uma transação Γ_i causa a τ_{ua}, durante um intervalo t, quando a tarefa τ_{ic} é liberada no instante crítico, é:

$$W_{ic}(\tau_{ua}, t) = \sum_{\forall j \in hp_i(\tau_{ua})} (I_{ijc}^{Set1} + I_{ijc}^{Set2}(t))$$

, onde $hp_i(\tau_{ua})$ é o conjunto de tarefas da transação Γ_i com prioridade maior ou igual a τ_{ua}.

Então, pela função de aproximação W_i^*, definida em [12], a interferência causada por uma transação Γ_i é dada por:

$$W_i^*(\tau_{ua}, t) = \max_{\forall c \in hp_i(\tau_{ua})} (W_{ic}(\tau_{ua}, t)) .$$

O cálculo completo do tempo de resposta para uma tarefa τ_{ua} é apresentado em [9,13]. Para o sistema modelado nesta abordagem de escalonamento híbrido o tempo de resposta pode se resumir a:

$$R_{ua} = C_{ua} + \sum_{\forall i \neq u} W_i^*(\tau_{ua}, R_{ua}) .$$

O cálculo é iterativo e termina quando R_{ua} convergir para um ponto fixo.

Para cada tarefa dinâmica o tempo de resposta é calculado e comparado ao seu *deadline*. Se o tempo de resposta for maior ou igual ao *deadline* da tarefa esta é dita como sendo garantida. Assim é possível garantir em tempo de projeto que um conjunto de tarefas dinâmicas consegue cumprir seus requisitos temporais para uma dada escala estática. O escalonamento *on-line* é realizado por meio de um escalonador de prioridade fixa, por exemplo Rate Monotonic [5] e Deadline Monotonic [14].

Avaliação

A abordagem apresentada aqui permite um melhor aproveitamento do processador, misturando dois tipos de escalonamento, *off-line* e *on-line*. O simples fato de escalonar uma tarefa esporádica dinamicamente, em vez de estaticamente, já diminui drasticamente o *overhead*. Em compensação, o *overhead* do sistema é aumentado pelo fato de ser utilizado escalonamento *on-line*. Porém, a flexibilidade é favorecida por esse motivo.

As tarefas com maior complexidade nas relações de restrição são escalonadas *off-line* e as tarefas que exigem maior flexibilidade são escalonadas *on-line*. Tarefas que exigem tempo de resposta curtíssimo podem ser modeladas como interrupções. Isso permite escalonar tarefas esporádicas com prioridades maiores e menores que as tarefas periódicas, diferente do método de Sandström *et al*., que só permite tarefas esporádicas com prioridade maior que das tarefas periódicas.

Este paradigma possibilita também que o tamanho da escala estática possa ser comprimido movendo tarefas com períodos grandes do grupo de tarefas estáticas para o grupo de tarefas dinâmicas.

3.2.6. Taxonomia

A tabela 3.1 mostra um resumo das abordagens descritas neste capítulo, baseando-se em características que foram apresentadas na seção 3.1 e são definidas abaixo:

Tempo de resposta curto indica se o algoritmo de escalonamento trata de tarefas com *deadlines* muito apertados.

Overhead determina se a técnica de escalonamento acarreta em um *overhead* alto ou baixo.

Relações de restrição mostra se a abordagem permite que as tarefas tenham relações de restrição complexas, com precedência e exclusão mútua, ou as tarefas possuam apenas relações simples, como algumas precedências e poucas relações de exclusão.

Utilização do processador determina se o algoritmo possibilita uma grande ou pequena execução útil de tarefas no processador, ou seja, se a grande parte do tempo o processador está executando tarefas.

Flexibilidade caracteriza o escalonamento como flexível a eventos externos, podendo ser muito flexível ou pouco flexível.

Tabela 3.1 - Taxonomia dos algoritmos de escalonamento híbrido

Características		**Mok**	**Servidor Esporádico**	**Sandström**	**Isovic**	**Mäki-Turja**
Tempo de resposta curto				•		•
Overhead	Alto	•	•		•	•
	Baixo			•		
Relações de restrição	Complexas	•		•	•	•
	Simples		•			
Utilização do processador	Grande		•		•	•
	Pequena	•		•		
Flexibilidade	Grande		•		•	•
	Pequena	•		•		

3.3. Análise

Os algoritmos híbridos apresentados têm, cada um, suas peculiaridades, como pode ser visto na tabela 3.1. O Servidor Esporádico não é tão apropriado a sistemas muito críticos comparado às outras abordagens, que fornecem uma maior previsibilidade. A técnica descrita por Mok é totalmente previsível, possui flexibilidade para tarefas esporádicas, mas só é viável se estas tarefas tiverem *deadlines* grandes. O algoritmo de Isovic *et al.* apresenta previsibilidade e flexibilidade, além de aumentar a utilização do processador. Entretanto, o *overhead* do sistema se torna grande demais utilizando esta abordagem. O algoritmo apresentado por Sandström *et al.* fornece uma boa previsibilidade, como também flexibilidade para tratamento de tarefas esporádicas com *deadline* curto. Porém,

as tarefas aperiódicas têm sempre prioridade maior que as tarefas periódicas. A técnica descrita por Mäki-Turja *et al.* utiliza o algoritmo de Sandström *et al.* para escalonar tarefas periódicas junto com interrupções, mas também permite uma flexibilidade maior para tratar de tarefas esporádicas com prioridades menores que as tarefas periódicas.

Em sistemas embarcados de tempo real críticos, a previsibilidade do sistema é essencial. Esses sistemas são geralmente implementados em microcontroladores, que costumam ter suporte a interrupções. Estas têm normalmente níveis de prioridade mais altos que as tarefas periódicas e tempos muito curtos de *deadline*. Os algoritmos que mais se adéquam a esse cenário é Sandström *et al.* e Mäki-Turja *et al.* Para os fins deste trabalho utilizaremos a abordagem de Sandström *et al.* que é mais simples de implementar e é apropriada a vários casos de sistemas críticos.

Como vimos na seção anterior, escalonamentos híbridos trazem maior flexibilidade aos sistemas críticos, permitindo que tarefas esporádicas sejam tratadas junto com tarefas periódicas. Porém esse fato traz consigo um problema de sistemas *event-triggered*, no qual falhas geram ativações indevidamente antecipadas de tarefas esporádicas, contradizendo a propriedade de tempo mínimo entre chegadas destas tarefas.

Este tipo de falha faz com que as tarefas não se comportem de acordo com a especificação do projeto e provoquem a perda da corretude temporal do sistema, assim como da corretude lógica.

O capítulo seguinte apresenta uma proposta de solução, realizada no nível de escalonamento, para o problema citado, aplicada ao algoritmo híbrido de Sandström *et al.*

4. Melhorando Escalonamento Híbrido

Poledna [20] introduz o problema, em sistemas dirigidos a evento, de sensores com falha, que podem gerar uma rajada de interrupções, fazendo com que o sistema não cumpra as suas restrições temporais. Para sistemas *pre-runtime* este fato compromete a previsibilidade inerente à abordagem *off-line*.

Também é apresentada por Poledna, para sistemas com escalonamento de prioridade fixa, uma forma de limitar as ativações de tarefas esporádicas e detectar falha nos sensores. Após a execução da interrupção, a mesma é desabilitada. Para cada interrupção é criada uma nova tarefa. É função desta nova tarefa habilitar novamente a interrupção após o tempo necessário para completar o tempo mínimo entre chegadas da interrupção.

Baseados na técnica de Poledna [20] para escalonamento *on-line*, propomos uma solução a ser aplicada no escalonamento híbrido de tarefas. Portanto, o resto deste capítulo descreverá o modelo do algoritmo de Sandström *et al.* alterado para proporcionar o tratamento seguro de interrupções em escalas *off-line* de tarefas periódicas.

Cada interrupção i é substituída por uma tarefa ST^i e por uma tarefa TM^i. A tarefa ST^i é responsável pela execução da rotina de interrupção e é estendida por uma rotina f que desabilita a interrupção e programa a ativação da tarefa TM^i. Esta última tem a função de detectar ativações antecipadas da interrupção e reabilitá-la. A tarefa TM^i pode ser implementada para não mais habilitar a interrupção caso um limiar de ativações antecipadas seja ultrapassado.

A tarefa ST^i é modelada da seguinte forma:

$$C_{ST}^i = C_i + C_f$$

$$T_{ST}^i = T_i$$

$$dl_{ST}^i = dl^i + C_f$$

A tarefa TM^i é definida pelas seguintes características:

C_{TM}^i dependente da implementação

$$rt_{TM}^i = rt_i + T_i - C_{TM}^i$$

dl_{TM}^i dependente da implementação

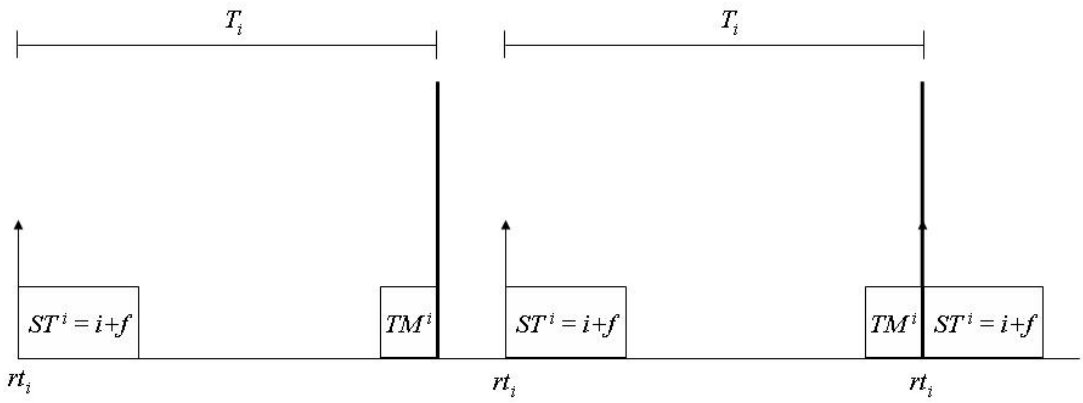

Figura 4.1 – Tratamento seguro de interrupções

Depois de efetuar esse processo para cada interrupção o sistema está pronto para ser escalonado.

Considerando as modificações descritas acima para cada interrupção, é preciso alterar alguns pontos na modelagem para adequá-la à técnica de escalonamento híbrido escolhida. As tarefas ST^i e TM^i são interrupções, que estão intimamente associadas. Já que para cada tarefa ST^i uma tarefa TM^i é ativada, podemos modelá-las como uma única interrupção i' com as seguintes propriedades:

$$C'_i = C^i_{ST} + C^i_{TM}$$

$$T'_i = T^i_{ST} = T_i$$

$$dl'_i = dl^i_{ST} + C^i_{TM}$$

É assumido que as interrupções podem ter diferentes níveis de prioridade entre si. Dessa forma, é necessário testar se cada interrupção i' tem seu *deadline* atendido, partindo do pior caso em que todas as interrupções com maior prioridade que i' são ativadas no mesmo instante que ela. O teste é feito na ordem da interrupção de maior prioridade para a interrupção de menor prioridade. O algoritmo funciona da seguinte forma:

$R(i') = C'_i + \sum_{j' \in hp(i')} C'_j$, onde $hp(i')$ é o conjunto de interrupções, derivadas da junção de tarefas ST e TM, com prioridade mais alta que i'.

Se $R(i') > dl'_i$ então a interrupção não pode ser escalonada.

Em seguida, um algoritmo de escalonamento *off-line* gera uma escala a partir do conjunto de tarefas periódicas. Para cada instância de uma tarefa é criada uma cadeia, descrita na seção 3.2.3, com tempo de início igual ao tempo de liberação da instância da tarefa, de acordo com a escala criada.

O algoritmo de escalonamento de Sandström *et al.* é então executado para o conjunto de cadeias e o conjunto de interrupções testadas.

Os resultados do método aqui proposto são apresentados no próximo capítulo por meio de um estudo de caso.

5. Estudo de Caso

Este capítulo mostra a aplicação do algoritmo proposto e se divide em duas partes: na primeira faz-se uma comparação entre a utilização de um escalonamento puramente *pre-runtime* e o algoritmo de Sandström *et al.,* escolhido na seção 3.3, levantando as vantagens de se usar esta última abordagem; na segunda parte a solução, descrita no capítulo anterior, para o tratamento seguro de interrupções é demonstrada.

Com a finalidade de simulação e estudo do trabalho, foi desenvolvida uma ferramenta para escalonamento de tarefas e exibição gráfica da escala. A ferramenta foi desenvolvida na linguagem Java, consumiu 3081 linhas de código e também terá fins educativos para a disciplina de Sistemas de Tempo Real no Centro de Informática da Universidade Federal de Pernambuco.

Figura 5.1 – Ferramenta de escalonamento

5.1. Pre-runtime vs. Híbrido

Esta simulação tem o objetivo de mostrar a vantagem de introduzir a abordagem híbrida no escalonamento de tarefas. Um conjunto de tarefas especificadas na tabela 5.1 é

escalonado junto com a interrupção da tabela 5.2. Primeiro foi feito o escalonamento puramente *pre-runtime,* explicado na seção 3.2.1, cuja escala gerada é mostrada na figura 5.2. Depois o algoritmo híbrido de Sandström *et al.* foi executado, resultando na escala da figura 5.3.

Tabela 5.1 - Tarefas Periódicas

Atributos	Tarefas Periódicas	
	T0	T1
C	5	4
dl	20	40
T	25	50

Tabela 5.2 - Interrupção

Atributos	Interrupções
	I0
C	1
dl	2
T	20

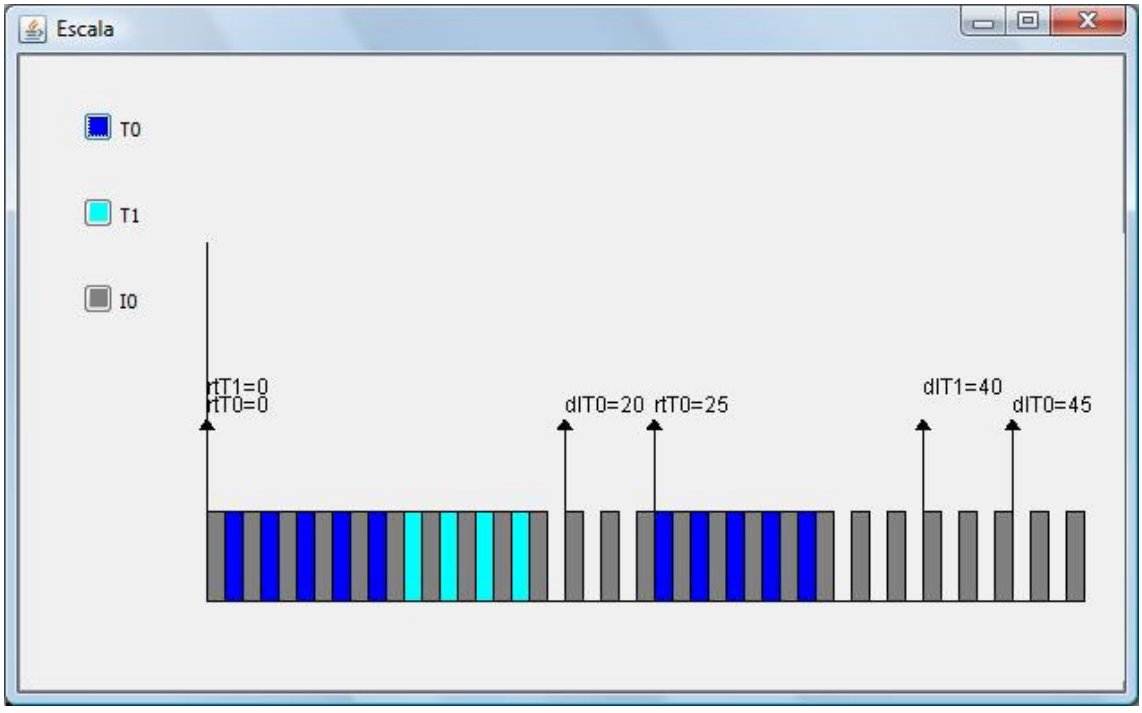

Figura 5.2 – Escalonamento pre-runtime de tarefas periódicas e interrupção

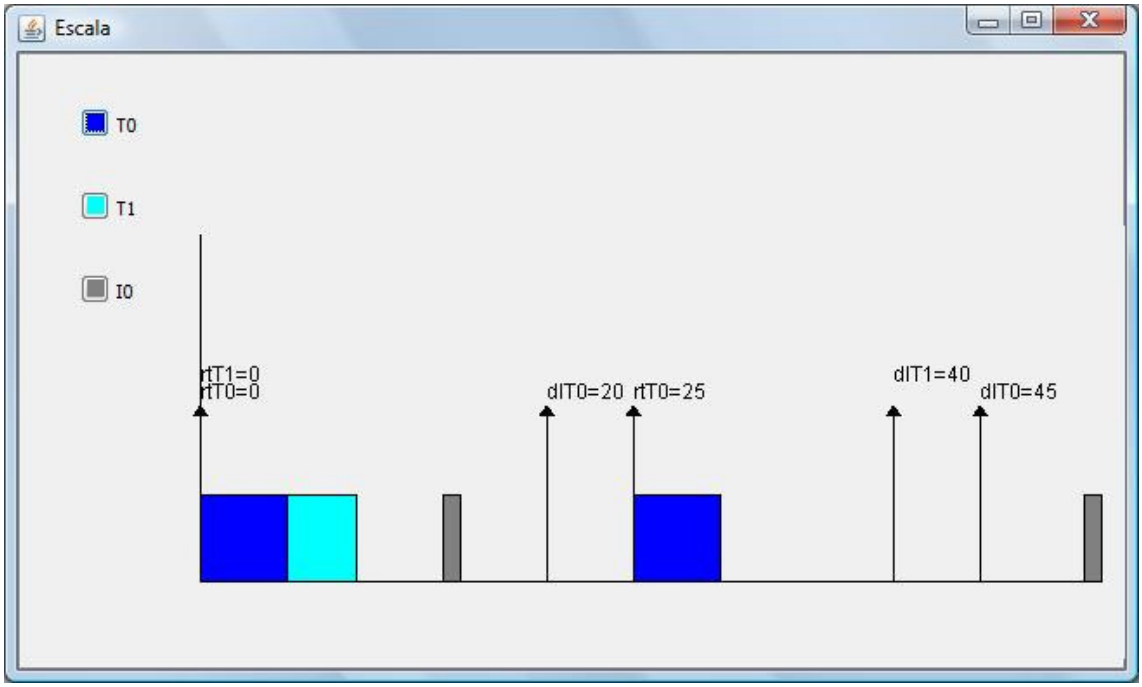

Figura 5.3 – Escalonamento híbrido de tarefas periódicas e interrupções

Podemos observar na figura 5.2 um *overhead* muito grande característico do tratamento de interrupções em sistemas totalmente *time-triggered*. A tabela 5.3 permite uma melhor comparação em termos de quantidade de ativações de tarefa. Conseguimos uma redução de 87% do *overhead* com o escalonamento híbrido.

Tabela 5.3 – Comparação de overhead

Escalonamento	Total de ativações	Ativações por unidade de tempo
Pre-runtime	39	0,800
Híbrido	5	0,096

Também é visível um melhor tempo de resposta das tarefas no segundo escalonamento comparado ao primeiro, como mostra a tabela 5.4.

Tabela 5.4 – Comparação de tempo de resposta

Escalonamento	Tempo de resposta	
	T0	T1
Pre-runtime	9	18
Híbrido	5	9

5.2. Tratamento seguro de interrupções

Este segundo experimento visa exemplificar o método de tratamento seguro de interrupções para escalonamento híbrido, proposto no capítulo anterior. As tarefas e interrupção escalonados são especificados nas tabelas 5.5 e 5.6, respectivamente. O resultado do escalonamento pode ser visto na figura 5.4.

Tabela 5.5 – Tarefas periódicas

Atributos	Tarefas Periódicas		
	T0	T1	T2
C	5	2	4
dl	20	40	20
T	25	50	25

Tabela 5.6 – Interrupção

Atributos	Interrupções
	I0
C	1
dl	2
T	10

As interrupções ativam as tarefa ST, que se auto-desabilitam. As tarefas TM reabilitam as interrupções no instante correto, controlando o tempo mínimo entre chegadas, como descrito no capítulo 4. Esta técnica aumenta o *overhead*, mas previne o sistema de interrupções erroneamente adiantadas.

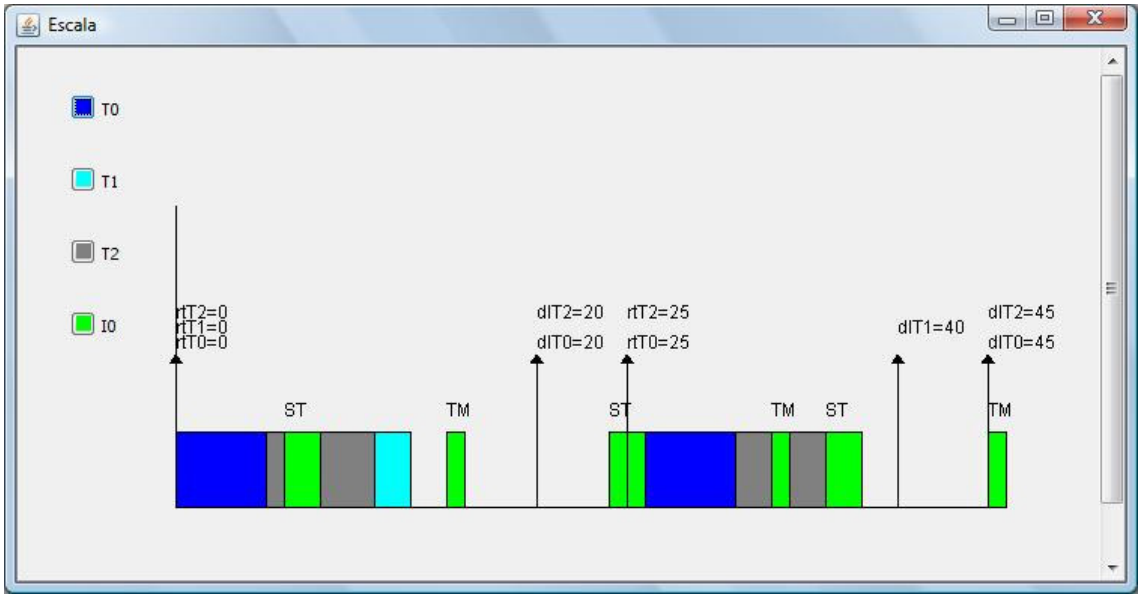

Figura 5.4 – Tratamento seguro de interrupções

6. Conclusão

O conceito de escalonamento híbrido de tarefas em sistemas de tempo real tem se tornado cada vez mais foco de estudos. A possibilidade de integrar o melhor dos sistemas *time-triggered* e *event-triggered* no que diz respeito a escalonamento é algo bastante interessante e tema de muitos trabalhos atuais.

Foi mostrado nesta monografia o estado da arte neste campo de estudo e uma pequena análise das abordagens apresentadas. As vantagens e desvantagens explicitam o melhor contexto de utilização das técnicas introduzidas no capítulo 3, e que podem ser resumidas na tabela taxonômica intuída no mesmo capítulo. Sistemas de tempo real muito críticos exigem previsibilidade e garantias em tempo de projeto. Tendo de tratar tarefas esporádicas de curtíssimo *deadline*, o algoritmo de Sandström *et al.* [2] se mostra muito bom e é bastante apropriado para sistemas que utilizam interrupções de prioridade mais alta que as tarefas periódicas, fato que acontece normalmente em sistemas críticos.

A inserção de características *event-triggered* no escalonamento das tarefas traz um problema típico deste modelo que é as ativações erroneamente adiantadas das tarefas esporádicas, comum em sensores com falhas. Poledna [20] colocou o problema e apresentou uma solução para sistemas de escalonamento *on-line*. Foi proposta neste trabalho uma solução do problema citado para o algoritmo de escalonamento híbrido de Sandström *et al.* Com o método proposto é possível criar um ambiente para projetar sistemas previsíveis, e com flexibilidade para tratar interrupções de uma forma segura garantindo os requisitos temporais do sistema de tempo real.

Como trabalho futuro, pode ser estudado o algoritmo híbrido de Mäki-Turja [19], permitindo o escalonamento de tarefas esporádicas com prioridade mais baixa que as tarefas periódicas. Este algoritmo ainda tem a vantagem de obter tempos de resposta mais curtos. Outro estudo que pode se derivar deste trabalho é o escalonamento híbrido para sistemas de tempo real multiprocessados. Esse tipo de sistema é muito comum e merece uma atenção especial. Poderia ser feita uma ferramenta, associada a um sistema operacional de tempo real, para projetar sistemas de tempo real críticos, permitindo tratamento seguro de interrupções com prioridades maiores ou menores que as tarefas periódicas, em ambientes multiprocessados.

7. Referências

[1] M. Joseph; P. K. Pandya. Finding response times in a real-time system. *The Computer Journal*, Vol. 29, n. 5, 1986.

[2] K. Sandström; C. Eriksson; G. Fohler. Handling Interrupts with Static Scheduling in an Automotive Vehicle Control System. Proceedings of the 1998 Real-Time technology and Applications Symposium, 1998.

[3] J. C. Palencia; M. González Habour. Schedulability Analysis for Tasks with Static and Dynamic Offsets. *IEEE Real-Time Systems Symposium*, pp. 26-37, 1998.

[4] Tindell, K. *Adding Time-Offsets to Schedulability Analysis*. Technical Report YCS 221, Dept. of Computer Science, University of York, 1994.

[5] C. L. Liu; C. L. Layland. Scheduling Algorithms for Multi-programming in a Hard Real-Time Environment. *Journal of the Association for Computing Machinery*, Vol 20, Vol. 46-61, 1973.

[6] G. Fohler. Joint scheduling of distributed complex periodic and hard aperiodic tasks in statically scheduled systems. *Proc. 16th Real-time Systems Symposium*, 1995.

[7] B. Sprunt; L. Sha; J Lenoczky. Aperiodic Task Scheduling for Hard-Real-Time Systems. *The Journal of Real-Time Systems*, Vol. 1, p. 27-60, 1989.

[8] M. Spuri; G. C. Buttazzo. Efficient Aperiodic Service under Earliest Deadline Scheduling. *Proc. IEEE Real-Time Systems Symposium*, pp. 2-11. 1994.

[9] J. Mäki-Turja; M. Nolin. Tighter Response-Times for Tasks with Offsets. *Proc. Of the 10th International conference on Real-Time Computing Systems and Applications (RTCSA'04)*, 2004.

[10] Buttazzo, G. *Hard Real-Time Computing Systems*. Kluwer Academic Publishers, 1997.

[12] Tindell, K. *Using Offset Information to Analyse Static Priority Pre-emptively Scheduled Task Sets*. Technical Report YCS-182, Dept of Computer Science, University of York, England, 1992.

[13] J. C. Palencia; M. Gonzalez Harbour. Schedulability Analysis of Tasks with Static and Dynamic Offsets. *Proc. of 19th IEEE Real-Time Systems Symposium (RTSS)*, 1998.

[14] Leung, J. Y. T.; Whitehead, J. *On the complexity of Fixed-Priority Scheduling of Periodic, Real-Time Tasks*. Performance Evaluation, 2 (4), pp. 237-250, December, 1982.

[15] Cavalcante, S. V. *A Hardware-Software Codesign System for Embedded Real-Time Applications*. PhD Thesis, Department of Electrical and Electronic Engineering, University of Newcastle, 1997.

[16] A. K. Mok. Design of real-time programming systems based on process models. *Proceedings of Real-Time Systems Symposium*, pp. 5-17. New York: IEEE, 1984.

[17] B. Sprunt; L. Sha; J Lehoczky. Aperiodic Task Scheduling for Hard Real-Time Systems. *The Journal of Real-Time Systems*, Vol. 1, pp. 27-60, 1989.

[18] D. Isovic; G. Fohler. Handling Sporadic Tasks in Off-line Scheduled Distributed Real-Time Systems. *Proc of 11th Euromicro Conference on Real-Time Systems*, 1999.

[19] J. Mäki-Turja; K Hänninen; M Nolin. Efficient Development of Real-Time Systems Using Hybrid Scheduling. *Proc. of the International conference on Embedded Systems and Applications*, 2005.

[20] S. Poledna. Tolerating Sensor Timing Faults in Highly Responsive Hard Real-Time Systems. *IEEE Transactions on Computers*, Vol. 44, Nº 2, 1995.

[21] H. Kopetz. Event-Triggered versus Time-Triggered Real-Time Systems. *Lecture Notes in Computer Science*, vol. 563, Springer Verlag, Berlin, 1991.

[22] H. Kopetz. The Time-Triggered Model of Computation. *Proc. 19th IEEE Real-Time System Symposium*, pp. 168–177, 1998.

[23] M. Young; L-C. Shu. Hybrid Offline/Online Scheduling for Hard Ral-Time Systems. *Proc. of 2nd International Symposium on Real-Time and Media Systems*, pp. 231-240, 1996.

[24] Kopetz, H. *Real-Time Systems – Design Principles for Distributed Embedded Applications.* Kluwer Academic Publishers, 1997.

[25] J. Xu; D. L. Parnas. Priority Scheduling Versus Pre-Run-Time Scheduling. *Journal of Real Time Systems*, Vol. 18, issue 1, pp. 7–24, 2000.

[26] Farines, J-M.; Fraga, J. S.; Oliveira, R. S. *Sistemas de Tempo Real*. Departamento de Autmoação e Sistemas, Universidade Federal de Santa Catarina, 2000.

[27] Burns, A. and Welling, A., Real-Time Systems and Their Programming Languages, Addison-Wesley, 1996.