Universidade Federal do Amazonas
Instituto de Ciências Exatas
Departamento de Ciência da Computação
Programa de Pós-Graduação em Informática

Síntese de Escalas Utilizando Semântica de Passo Máximo em Redes de Petri com Temporização

Romulo Devezas Freitas

Manaus – Amazonas
Julho de 2006

Romulo Devezas Freitas

Síntese de Escalas Utilizando Semântica de Passo Máximo em Redes de Petri com Temporização

Dissertação apresentada ao Programa de Pós-Graduação em Informática do Departamento de Ciência da Computação da Universidade Federal do Amazonas, como requisito parcial para obtenção do Título de Mestre em Informática. Área de concentração: Sistemas Embarcados.

Orientador: Prof. Dr. Raimundo da Silva Barreto

Romulo Devezas Freitas

Síntese de Escalas Utilizando Semântica de Passo Máximo em Redes de Petri com Temporização

Dissertação apresentada ao Programa de Pós-Graduação em Informática do Departamento de Ciência da Computação da Universidade Federal do Amazonas, como requisito parcial para obtenção do Título de Mestre em Informática. Área de concentração: Sistemas Embarcados.

Banca Examinadora

Prof. Dr. Raimundo da Silva Barreto – Orientador
Departamento de Ciência da Computação – DCC/UFAM

Prof. Dr. Paulo Romero Martins Maciel
Centro de Informática – CIn/UFPE

Prof. Dr. Vicente Ferreira de Lucena Junior
Departamento de Eletrônica e Telecomunicações – DET/UFAM

Manaus – Amazonas
Julho de 2006

Para

Minha amada esposa.

Metade de tudo que sou...

... a melhor metade.

Agradecimentos

Meus sinceros agradecimentos ao meu orientador e amigo Raimundo Barreto, pelo esforço, dedicação e paciência: "Se existe algum mérito na realização deste trabalho, ele é seu. Obrigado por acreditar que era possível. Sou honrado em tê-lo tido como orientador".

Obrigado ao PPGI, na pessoa de seu Coordenador, Edleno Silva, pela oportunidade e pelos recursos.

Sou "devedor" à Fundação Paulo Feitoza, pela liberação de tempo parcial, sem a qual este trabalho jamais teria se realizado.

Minha profunda gratidão à minha esposa, por nunca me deixar esquecer que existem coisas na vida mais importantes que um mestrado (por exemplo: terminá-lo). "Não tenho palavras para expressar a diferença que você faz em minha vida".

Também sou grato aos colegas e amigos que cooperaram com diversas idéias, longas conversas de corredor e boas piadas (essas foram as mais produtivas)

Por fim, meu obrigado ao meu Deus, de quem sou fã incondicional, e que tem um modo todo especial de transformar em realizações um monte de planos mal traçados.

Pois dele, por ele e para ele são todas as coisas.

Romanos 11:36

Resumo

No contexto restritivo dos sistemas embarcados de tempo real críticos, a geração de código escalonado mostra-se uma maneira eficiente de proporcionar previsibilidade e demais garantias exigidas por tais sistemas. Entretanto, a geração de código escalonado pressupõe conhecimento prévio da ordem em que as tarefas devem ser executadas. Assim, a síntese de escalonamento é peça chave em um metodologia para síntese de software neste ambiente. Este trabalho apresenta uma abordagem para síntese de escalonamento estático utilizando semântica de passo máximo em redes de Petri com temporização.

Este trabalho está inserido no contexto de uma metodologia para síntese de software de sistemas embarcados de tempo real críticos. Em linhas gerais, essa metodologia compreende (i) a especificação do sistema; (ii) a transformação da especificação em um modelo formal baseado em rede de Petri; (iii) a síntese de escalonamento baseada neste modelo e (iv) a geração de código escalonado.

No escopo da síntese de escalonamento, este trabalho propõe um conjunto de definições que caracterizam uma estratégia de disparo de passo máximo sobre redes de Petri com temporização, e aplica estas definições na síntese de escalonamento estático para sistemas embarcados de tempo real críticos.

Os resultados obtidos nos experimentos apresentados indicam a viabilidade da aplicação da semântica de passo máximo proposta, demonstrando melhorias de desempenho relacionadas ao tempo de execução e ao volume de informação processada.

Adicionalmente, além da apresentação de uma abordagem de passo máximo para síntese de escalonamento, este trabalho ainda propõe algumas melhorias e aperfeiçoamentos na metodologia e no modelo adotados como base.

Palavras-chave: Semântica de Passo, Redes de Petri, Escalonamento Estático, Síntese Software.

.

Sumário

1 Introdução **1**

1.1 Contexto 2

1.2 Descrição do Problema 3

1.3 Trabalhos Relacionados 4

1.3.1 Escalonamento Estático 4

1.3.2 Escalonamento Utilizando Redes de Petri 6

1.3.3 Semântica de Passo Máximo em Redes de Petri com Temporização 7

1.4 Organização da Dissertação 8

2 Conceitos Básicos **9**

2.1 Sistemas Embarcados 9

2.1.1 Introdução 10

2.1.2 Representações ou Visões de Projeto 11

2.1.3 Projeto de Sistemas Embarcados – Hardware-Software Codesign . . 12

2.1.4 Softwares Embarcados 14

2.2 Sistemas de Tempo Real 15

2.2.1 Introdução 16

2.2.2 Características dos Sistemas de Tempo Real 16

2.2.3 Classificação dos Sistemas de Tempo Real 18

2.2.4 Classificação das Tarefas em Sistemas de Tempo Real 18

2.2.5 Principal Desafio 19

2.3 Escalonamento em Sistemas de Tempo Real 19
2.3.1 Introdução 20
2.3.2 Escalonamento Dinâmico 21
2.3.3 Escalonamento Estático 22
2.3.4 Complexidade Algorítmica 23
2.3.5 Vantagens do Escalonamento Estático 24
2.4 Redes de Petri 25
2.4.1 Introdução 26
2.4.2 Redes de Petri Lugar-Transição 26
2.4.3 Redes de Petri Elementares 30
2.4.4 Extensões Temporais em Redes de Petri 31
2.4.5 Propriedades das Redes de Petri 33
2.4.6 Métodos de Análise em Redes de Petri 35

3 Um Modelo para Sistemas Embarcados de Tempo Real Críticos 37

3.1 Especificação 38
3.1.1 Restrições Temporais e de Energia 38
3.1.2 Relações entre Tarefas 39
3.1.3 Métodos de Escalonamento 40
3.1.4 Comunicação entre Tarefas 40
3.1.5 Especificação Comportamental 41
3.1.6 Exemplo de Especificação 41
3.2 Fundamentos Computacionais 42
3.2.1 Fundamentos para um Modelo com Restrições Temporais 43
3.2.2 Extensão para um Modelo com Restrições de Consumo de Energia . 51
3.3 Construção do Modelo 53
3.3.1 Regras de Composição 53
3.3.2 Período 58
3.3.3 Blocos Básicos 59

3.3.4 Modelando Tarefas 65
3.3.5 Modelando Relações entre Tarefas 68
3.3.6 Modelando Comunicação entre Tarefas 70
3.3.7 Modelando Sobrecarga do Despachante 71

4 Síntese de Escalonamento Estático 75

4.1 Questões Iniciais 76
4.1.1 Eliminação de Estados Indesejados 76
4.1.2 Escalonamento Estático e Consumo de Energia 77
4.2 Abordagem Baseada em *Interleaving* - Passo Unitário 77
4.2.1 Redução por Ordem Parcial 78
4.2.2 Heurísticas para Ordenação 80
4.2.3 Algoritmo para Síntese de Escalonamento Estático - *Interleaving* . . 81
4.2.4 Aplicação do Algoritmo 83
4.3 Abordagem Baseada em Passo Máximo 85
4.3.1 Resolução de Conflitos Dinâmicos 85
4.3.2 Definição do Domínio de Disparo 88
4.3.3 Algoritmo para Síntese de Escalonamento Estático - Passo Máximo 90
4.3.4 Aplicação do Algoritmo 93

5 Experimentos 95

5.1 Sistema Básico Composto de Duas Tarefas 95
5.2 Oxímetro 96
5.3 Outros Resultados Práticos 98
5.4 Avaliação 99

6 Conclusões 101

6.1 Contribuições 103
6.2 Limitações 104
6.3 Trabalhos Futuros 105

6.4 Comentário Final 106

Referências Bibliográficas **107**

Lista de Figuras

2.1 Principais Fases da Metodologia Hardware-Software Codesign 13

2.2 Comparação entre Escalonamento Dinâmico (a) e Estático (b) 25

2.3 Representação Gráfica para Rede de Petri: (a) Lugar, (b) Arco, (c) Transição e (d) *Token* 27

2.4 Disparo de uma Transição em Rede de Petri. (a) Formalismo; (b) Antes do Disparo de t_1; (c) Após o Disparo de t_1 29

2.5 Redes de Petri Elementares 30

3.1 Exemplo de Rede de Petri com Temporização: (a) Marcação Inicial; (b) Marcação Alcançada após Disparo de t_0 44

3.2 Grafo de Alcançabilidade - Disparo de t_1 e t_2: (a) Fora do Intervalo de Disparo comum; (b) Dentro do Intervalo de Disparo Comum e no Mesmo Instante; (c) Dentro do Intervalo de Disparo Comum e em Instantes Distintos 47

3.3 Rede de Petri com Conflitos Dinâmicos 49

3.4 Diagrama Temporal para o Passo $\{t_1, t_2, t_3, t_4\}$ 50

3.5 Refinamento Serial de Lugar 56

3.6 Bloco de Chegada de Tarefas 61

3.7 Bloco de Execução de Tarefa Não-Preemptiva 61

3.8 Bloco de Execução de Tarefa Preemptiva 62

3.9 Bloco de Verificação de *Deadline* 63

3.10 Bloco de Recursos: (a) Processador; (b) Barramento 63

3.11 Bloco de Disjunção 64

3.12 Bloco de Junção 65
3.13 Bloco de Envio de Mensagem 65
3.14 Modelo para Execução da Tarefa T_0 67
3.15 Modelo de um Sistema Monoprocessado com Duas Tarefas Não-preemptivas . 68
3.16 Modelo para Relação de Precedência para as Tarefas T_1 e T_2 69
3.17 Modelo para Relação de Exclusão entre T_0 e T_2 70
3.18 Modelo para Envio de Mensagem a partir de τ_i 72
3.19 Modelo para Recepção de Mensagem em τ_i 72
3.20 Bloco Básico para Tarefa com Sobrecarga do Despachante 73

4.1 Algoritmo para Síntese de Escalonamento Estático - *Interleaving* 82
4.2 Rede para as Tarefas da Tabela 3.2 83
4.3 Rede de Petri com Conflitos Dinâmicos 86
4.4 Algoritmo Para Criação do Conjunto de Passos Máximo 87
4.5 Rede de Petri com Temporização e Conflitos Dinâmicos 88
4.6 Diagrama Temporal de Disparo para Seis Transições 88
4.7 Diagrama Temporal de Disparo: (a) Passo st_1; (b) Passo st_2 89
4.8 Algoritmo para Síntese de Escalonamento Estático - Passo Máximo 91
4.9 Algoritmo para Síntese de Escalonamento Estático - Passo Máximo Local . . . 92
4.10 Diagrama Temporal para Duas Tarefas 94

5.1 Resumo dos Experimentos 99

Lista de Tabelas

3.1 Exemplo de Especificação 42
3.2 Restrições Temporais para Duas Tarefas 59
3.3 Exemplo de Especificação com 3 Tarefas - Sem Consumo de Energia 66

4.1 Ordem de Escolha para Cada Classe de Transição 79
4.2 Execução do Algoritmo - *Interleaving* 84
4.3 Execução do Algoritmo - Passo Máximo 94

5.1 Restrições Temporais para Duas Tarefas 96
5.2 Especificação de Tarefas para o Oximetro de Pulso 97

Capítulo 1

Introdução

Sistemas embarcados são, na grande maioria dos casos, sistemas que reagem a estímulos produzidos pelo ambiente (sistemas reativos). Quando esta reação esta sujeita a restrições temporais, estes sistemas são chamados de sistemas embarcados de tempo real. Tais sistemas são ditos *críticos* quando o não cumprimento das restrições temporais não é uma opção aceitável, podendo representar conseqüências desastrosas.

O desenvolvimento de software para este tipo de sistema pressupõe a aplicação de metodologias de desenvolvimento que possam fornecer maiores garantias quando a exatidão do produto de software gerado em relação às especificações realizada na fase de projeto. Neste contexto, a síntese de software é o processo pelo qual uma especificação é automaticamente convertida em um software (código fonte) [12].

Considerando as diversas limitações e restrições presentes em um sistema embarcado de tempo real crítico, a síntese de software para este tipo de sistema tem como principal objetivo a geração de um software com o mínimo de sobrecarga e atendendo todas as restrições temporais e de energia. Neste sentido, um modo eficaz de atingir esse objetivo é através da geração de código escalonado. Entretanto, a geração de código escalonado, pressupõe o conhecimento prévio da ordem em que as tarefas serão executadas (escala[1]).

Este trabalho apresenta um método para síntese de escalonamento estático utilizando

[1]O termo escala será utilizado neste trabalho para identificar o resultado do processo de escalonamento de um conjunto de tarefas

semântica de passo máximo em redes de Petri com temporização. Este tipo de escalonamento de tarefas é dito *estático* por se tratar de uma atividade realizada durante a elaboração do sistema (em tempo de projeto), e não durante a execução deste.

O uso da abordagem de passo máximo é uma tentativa de otimizar o processo de geração de escala, através da redução do espaço de estados e do tempo necessário para avaliação desses estados. A utilização de redes de Petri com temporização como fundamentação teórica para o modelo é motivada pela capacidade que este formalismo possui em expressar situações como: concorrência, paralelismo, compartilhamento de recursos, limitações temporais de execução e alguns outros elementos presentes no contexto dos sistemas embarcados de tempo real críticos.

1.1 Contexto

Este trabalho está inserido no contexto de síntese de software para sistemas embarcados de tempo real críticos. Os sistemas embarcados são, geralmente, sistemas com funcionalidades específicas, um conjunto restrito de tarefas e inúmeras restrições físicas e computacionais (como por exemplo, consumo de energia, uso limitado de memória, baixo poder de processamento). Já os sistemas de tempo real críticos, são sistemas sujeitos a rígidas restrições temporais.

O escopo é o de síntese de escalonamento estático, tendo como base a metodologia para desenvolvimento de sistemas embarcados proposta por Barreto [6].

Em linhas gerais, esta metodologia contempla (i) a especificação de um conjunto de características de interesse em um sistema embarcado de tempo real críticos; (ii) a transformação desta especificação em um modelo baseado em redes de Petri com temporização; (iii) a síntese de escalonamento para o conjunto de tarefas que compõem o modelo; (iv) e por fim, a geração automática do código escalonado e específico para a plataforma a qual o sistema de destina. Todo este processo é amparado por um conjunto de ferramentas integradas que ocultam do usuário (projetista de sistemas) os detalhes relacionados ao modelo formal empregado.

Como se trata de uma metodologia específica para sistema embarcados de tempo real crítico, a proposta de Barreto emprega uma abordagem de escalonamento estática, que proporciona a previsibilidade e o controle sobre a execução necessárias neste tipo de sistema.

A síntese de escalonamento empregada nessa metodologia é baseada na exploração do espaço de estados da rede de Petri utilizada para representação do modelo, utilizando uma abordagem de disparo de transições individuais. Esta abordagem resulta, naturalmente, em um crescimento exponencial do espaço de estados, um problema conhecido como *explosão de estados* [16, 52]. Para minimizar o impacto deste problema, são utilizadas algumas técnicas para redução do número de estados que necessitam ser obrigatoriamente analisados.

1.2 Descrição do Problema

O problema considerado nesta dissertação é a aplicação de uma abordagem diferente da utilizada na metodologia proposta por Barreto [6] para o disparo de transições durante a geração e análise do espaço de estados do modelo (rede de Petri). Esta abordagem diferente consiste no disparo de um conjunto máximo de transições aptas em cada estado.

Em redes de Petri convencionais (não temporizadas) o disparo simultâneo de um conjunto de transições parece ser uma excelente estratégia para se atingir mais rapidamente uma determinada marcação, uma vez que o espaço de estados pode ser bastante reduzido com esta abordagem. Entretanto, é preciso considerar que este tipo de disparo exige um esforço maior na determinação de qual o conjunto a ser disparado. Além disso, em redes de Petri com temporização, os estados são caracterizados não somente pela marcação da rede, mas também estão condicionados a restrições temporais, e essas restrições não podem ser desconsideradas na definição de uma semântica de passo máximo compatível com o modelo adotado.

Também é preciso considerar que a redução no espaço de estados proporcionada pela estratégia de passo máximo adotada, não pode inviabilizar a oportunidade de que uma

escala viável possa ser encontrada, caso esta exista.

Desse modo, o problema considerado neste trabalho pode ser sintetizado na seguinte questão: *pode a abordagem de passo máximo, em redes de Petri com temporização, ser utilizada para síntese de escalonamento estático em sistemas embarcados de tempo real crítico?*

Para tratar essa questão, esta dissertação propõe algumas extensões e redefinições na especificação, no modelo e na síntese de escalonamento apresentadas originalmente em [6]. Entretanto, aspectos ligados à geração automática de código, resultante da escala gerada, não são diretamente abordados.

1.3 Trabalhos Relacionados

A seguir são citados alguns trabalhos significativos e que estão de algum modo relacionado com o tema proposto nesse trabalho. A seção está dividida em: (i) trabalhos relacionados a escalonamento estático; (ii) trabalhos que empregam redes de Petri na avaliação e análise de escalonamento em sistemas de tempo real; e (iii) trabalhos que utilizam semântica de passo máximo na avaliação de redes com temporização.

1.3.1 Escalonamento Estático

Em Xu e Parnas [59] é apresentada a primeira tentativa de formalizar um método de escalonamento estático para sistemas de tempo real críticos com relações de exclusão e precedência arbitrarias. Neste trabalho, os autores definem o escopo do problema de escalonamento estático e fornecem diversos termos para caracterizá-lo. As principais definições são as seguintes: cada processo é dividido em um conjunto de *segmentos* que possuem tempos específicos de liberação, de computação e limite para finalização (*deadline*[2]); todos os intervalos temporais são divididos em *unidades de tempo*, onde uma *unidade de tempo* é definida como sendo a menor quantidade de tempo que um segmento pode estar

[2]Por se tratar de um termo consagrado pelo uso, a palavra inglesa *deadline* será utilizada em diversas situações no lugar da expressão "tempo limite para finalização"

em execução sem ser interrompido; o atraso (*lateness*) de um segmento é calculado como sendo o tempo em que o segmento completa sua execução subtraído pelo tempo limite de finalização para o segmento; o atraso de uma escala, para um conjunto de processos, é o máximo atraso de qualquer segmento pertencente a qualquer processo do conjunto; uma escala viável é uma escala onde todos os tempos limites de finalização são respeitados e uma escala ótima é uma escala com o mínimo atraso.

Este trabalho propõe um algoritmo, baseado na técnica de *branch-and-bound*, onde um grande número de possíveis escalas é analisado com o objetivo de que uma escala ótima possa ser encontrada. Se comparado com métodos empíricos, esta abordagem reduz a possibilidade de erros e o tempo necessário à definição de uma escala em tempo de projeto. Ainda assim, os autores não apresentam experimentos baseados em sistemas reais.

Em [56], Xu apresenta um método para encontrar uma escala viável considerando uma arquitetura multiprocessada. Entretanto, a solução apresentada é bastante limitada, uma vez que considera que as tarefas não podem ser preemptadas, que os processadores são sempre idênticos, que tarefas podem ser executada em qualquer processador disponível, e que o custo de comunicação entre tarefas em processadores distintos é desprezível.

Com o objetivo de lidar com sistemas de tempo real distribuídos, em [1] é proposta uma extensão para o algoritmo de escalonamento estático de Xu e Parnas. Neste algoritmo as tarefas são escalonadas em múltiplos processadores e a troca de mensagens também é considerada.

O problema de escalonamento de tarefas com consumo mínimo de energia nas atividades de entrada e saída (I/O) em sistemas de tempo real crítico é tratado em [48]. Este trabalho adota uma abordagem de escalonamento estático, empregando técnicas de poda baseadas em tempo e energia, como também, métodos heurísticos para redução da complexidade do problema. Entretanto, a abordagem proposta não contempla múltiplos processadores, nem tão pouco, relações entre tarefas.

AlEnawy e Aydin [3] propõe mecanismos de escalonamento estático e dinâmico que

levam em consideração questões temporais e de energia. Mesmo considerando a possibilidade de preempção, esta proposta não considera relações entre tarefas e não garante que todas as tarefas serão executas, e sim apenas um grupo selecionado de tarefas com alta prioridade.

1.3.2 Escalonamento Utilizando Redes de Petri

A maior parte dos trabalhos envolvendo redes de Petri e escalonamento está relacionada com a análise de escalabilidade, ou seja, são trabalhos que tem como objetivo principal determinar se uma seqüência de disparo de transições é ou não escalonável. A seguir são apresentados alguns desses trabalhos.

Bucci [9] apresenta uma extensão para redes de Petri com temporização chamada Rede de Petri com Temporização e Preemptiva (*Preemptive Time Petri Net* - pTPN), que é uma rede de Petri com temporização e com um mecanismo adicional de associação de recursos. Este mecanismo condiciona o progresso do tempo de cada transição habilitada à disponibilidade de um conjunto de recursos preemptivos. Neste trabalho, as pTPN são utilizadas para verificação de sistemas de tempo real com tarefas preemptivas. Também são consideradas a comunicação e as relações de precedência e exclusão entre tarefas.

Em [51] é proposta uma extensão temporal para redes de Petri chamada Redes de Petri com Restrições Temporais (*Timing Constraint Petri Net* - TCPN) que associa a cada lugar e transição uma restrição temporal máxima e mínima e utiliza uma política que considera os disparos como não obrigatórios. Mesmo oferecendo maior expressividade, as TCPN são certamente muito mais difíceis de serem analisadas, e segundo [55] suas complexas restrições temporais oferecem pouca ajuda na modelagem e análise de sistemas de tempo real.

Em [55] é apresentada uma abordagem composicional para análise de escalabilidade em sistemas de tempo real utilizando redes de Petri com temporização [28]. A técnica de análise proposta neste trabalho separa a análise das propriedades temporais das demais propriedades comportamentais não-temporais. Desse modo, a análise pode ser realizada,

primeiramente, através da análise de alcançabilidade, sem considerar as restrições temporais, e posteriormente, através da análise temporal das seqüências encontradas.

Mesmo considerando somente arquiteturas monoprocessadas, [55] apresenta contribuições significativa, sendo: (i) uma abordagem que é capaz de determinar se uma seqüência especifica de transições é ou não escalonável, calcular o tempo de execução de uma tarefa escalonável, ou descobrir qual transição não é escalonável, de modo que suas restrições temporais possam ser ajustadas; e (ii) uma abordagem composicional para lidar com seqüência de transições complexas, onde uma seqüência é decomposta em subseqüências.

Em [4] é proposta uma extensão para redes de Petri denominada Redes de Petri com Tempo Limite (*Petri Nets with Deadline* - PND), bem como uma linguagem de modelagem chamada Automatos Temporizados com Tempo Limite (TAD *Timed Automata with Deadlines*) e um modelo semântico. A abordagem apresentada sintetiza todas as escalas satisfazendo uma dada propriedade. Entretanto, os autores não tratam diretamente o problema de explosão de estados.

1.3.3 Semântica de Passo Máximo em Redes de Petri com Temporização

Poucos trabalhos tratam as questões relacionadas ao disparo simultâneo de um conjunto de transições em redes de Petri com temporização. Mesmo ferramentas consagradas de modelagem, análise e verificação em redes de Petri (como por exemplo, INA [46]) não contemplam a possibilidade do disparo de um passo máximo neste tipo de rede.

Em [10] é apresentada uma extensão para redes de Petri denominada de Rede de Petri com Temporização, Preemptiva e Estocástica (*stochastic preemptive Time Petri Nets* - spTPN), que é uma rede de Petri na qual o progresso dos relógios das transições habilitadas esta condicionada a disponibilidade de um conjunto de recursos, e tanto o atraso de uma transição quanto a resolução de conflitos são caracterizados por funções de probabilidade. Este trabalho utiliza uma abordagem de disparo de transições baseada

em passo máximo. Porém, a complexidade na definição e análise das mudanças de estado através de passos máximos é reduzida pela introdução de uma política de disparo que obriga o disparo das transição, tão logo essas estejam aptas a disparar.

A mesma abordagem é também assumida em [47], onde o autor apresenta uma ferramenta para prototipação e análise em redes de Petri com temporização.

1.4 Organização da Dissertação

A organização deste trabalho conserva uma relação de interdependência entre os capítulos, de modo que conceitos e definições apresentas nos capítulos iniciais são amplamente utilizados nos capítulos seguintes. O texto está organizado em seis capítulos, dos quais este é o primeiro.

O Capitulo 2 apresenta os conceitos necessários à contextualização e entendimento deste trabalho. Neste capítulo são apresentados conceitos relacionados a sistemas embarcados, sistemas de tempo real, escalonamento e redes de Petri.

O Capítulo 3 apresenta o modelo para sistemas utilizado neste trabalho. O capítulo inicia pela apresentação de um padrão de especificação para sistemas embarcados de tempo real crítico, em seguida, apresenta o formalismo computacional utilizado no processo de modelagem e por fim caracteriza o processo de transformação da especificação em um modelo formal representado por uma rede de Petri com temporização

O Capítulo 4 apresenta a síntese de escalonamento estático com base no modelo proposto. Neste capítulo são discutidos elementos envolvidos na síntese de escalonamento estático utilizando disparo de um passo máximo e semática de transição.

O Capitulo 5 apresenta e discute os resultados de alguns experimentos selecionados.

Finalmente, no Capítulo 6 são apresentadas algumas conclusões, discutidas algumas fragilidades do modelo e dos métodos propostos, apresentadas as contribuições e as direções para trabalhos futuros.

Capítulo 2

Conceitos Básicos

Este capítulo apresenta alguns conceitos básicos necessários à contextualização e compreensão deste trabalho. Desse modo, os conceitos são apresentados de forma disjunta, e somente posteriormente, nos capítulos seguintes, serão relacionados de maneira a compor um referencial lógico e estruturado.

O texto está organizado da seguinte maneira: a primeira seção expõe conceitos relacionados a sistemas embarcados. A seção seguinte discorre sobre sistemas de tempo real. Em seguida são aparentados alguns conceitos relacionados a escalonamento, e por fim, é apresentado o referencial teórico relacionado as redes de Petri.

2.1 Sistemas Embarcados

Esta seção discorre sobre alguns tópicos e conceitos selecionados em sistemas embarcados, com ênfase nas questões relacionadas a projeto de sistemas e de softwares. Após uma breve introdução, são apresentando três possíveis representações (visões) de projeto de um sistema embarcado. Posteriormente é apresentada a metodologia de projeto chamada Hardware-Software Codesign, e finalmente, questões relacionadas a projetos de software embarcados são discutidas.

2.1.1 Introdução

Sistemas embarcados são, geralmente, sistemas digitais que executam um conjunto específico de funções dentro de um sistema maior. São sistemas controlados por microprocessadores ou microcontroladores onde, na maioria dos casos, o usuário não está ciente da existência de um computador como parte do sistema, uma vez que sua principal função não é a computacional [54].

Este tipo de sistema já está completamente inserido no mundo moderno, estando presente tanto no cotidiano doméstico, através de uma infinidade de dispositivos eletro-eletrônicos, quanto em aplicações de alta tecnologia como naves espaciais e equipamentos bélicos.

Mesmo possuindo valor significativo, a funcionalidade de um sistema embarcado é, normalmente, secundária em relação ao sistema maior no qual este está inserido. Esta posição secundária, aliada a diversidade de contextos nos quais este tipo de sistema é aplicado, explicam, em parte, as enormes restrições às quais tais sistemas estão sujeitos. É bastante usual a exigência de características como: confiabilidade garantida, baixos custos, requisitos de tempo-real, dimensões reduzidas, baixo consumo de energia, garantia de funcionalidade em ambientes atípicos, entre tantos outros requisitos que os diferenciam profundamente de outros sistemas computacionais de propósito geral.

Inicialmente, a maior parte dos sistemas embarcados era essencialmente baseada em hardware, utilizando-se por exemplo Circuito Integrado de Aplicação Especifica (ASIC - *Application Specific Integrated Circuit*). Entretanto, à medida que o avanço tecnológico dos processadores tem aumentado o poder computacional e reduzido as dimensões e preços desses componentes, um número cada vez maior de funcionalidades tem sido implementadas por meio de software. Atualmente, estima-se que mais de 80% das funcionalidades dos sistemas embarcados modernos esteja baseada em software [38]. Mesmo trazendo algumas desvantagens relacionadas a desempenho e consumo, esta migração proporciona vantagens que facilmente a justificam, como: flexibilidade, menores custos e maior acessibilidade. Assim, exerce influência direta na popularização e proliferação desse tipo de

sistema. Essa proliferação tem demandado o desenvolvimento de sistemas cada vez mais complexos em prazos cada vez mais curtos.

Nesse contexto de inúmeras restrições, crescente complexidade e prazos muito curtos, metodologias de especificação, projeto e desenvolvimento, específicas para sistemas embarcados, desempenham um importante papel, sendo que em muitos casos, já não é possível fornecer as garantias necessáras com o uso de uma abordagem puramente empírica. Este trabalho se concentra no paradigma denominado Hardware-Software Codesign que é apresentado mais a frente.

2.1.2 Representações ou Visões de Projeto

Como o objetivo de controlar e mesmo reduzir a complexidade no desenvolvimento de sistemas embarcados, vários níveis de abstração são aplicados, produzindo diferentes representações do sistema. A seguir são apresentados três dessas representações ou visões [14]:

a) **Representação Comportamental**. É uma visão simplificada baseada em "caixas-pretas", que define as respostas do sistema para cada valor de entrada, mas omite os detalhes de projeto para cada "caixa-preta". Assim, apenas a representação funcional do sistema é capturada, porém, nenhuma informação de como as funcionalidades serão implementadas é fornecida;

b) **Representação Estrutural**. Como o nome sugere, esta representação se concentra na especificação da estrutura e implementação do produto, definindo os conjuntos de componentes e suas interconexões. No entanto, não descreve as funcionalidades;

c) **Representação Física**. Especifica as características físicas dos componentes descritos na representação estrutural, podendo, por exemplo, fornecer as dimensões e localização de cada componente e as características das conexões entre eles.

Normalmente, o projeto segue um ciclo baseado na seqüência : representação comportamental, representação estrutural e finalmente representação física. Desse modo, detalhes de projeto são acrescentados em cada etapa.

2.1.3 Projeto de Sistemas Embarcados – Hardware-Software Codesign

Existem várias paradigmas para o desenvolvimento de sistemas embarcados. Entretanto, essa seção se concentra apenas em Hardware-Software Codesign, que pode ser definido como um paradigma de desenvolvimento cooperativo de hardware e software. Sendo, portanto, voltada a sistemas heterogêneos e compreendendo não somente especificação, como também projeto e síntese.

Assim, pode-se dizer que a grande questão em Codesign é determinar quais funcionalidades serão implementadas por meio de software e quais serão implementadas diretamente em hardware. Em outras palavras, equacionar a flexibilidade (alcançada através da migração de funcionalidades de hardware para software) com o desempenho (obtido por meio da migração de funcionalidades implementadas por software para implementações diretamente em hardware). Este balanceamento é primordialmente guiado pela redução de esforços, prazos e custos.

Fases da Metodologia Codesign

A metodologia de Hardware-Software Codesign pode ser descrita em quatro fases principais (Figura 2.1), sendo estas: *especificação*, *particionamento*, *co-síntese* e *analise e validação*. Essas fases são resumidamente descritas a seguir.

a) **Especificação**. Esta fase descreve os requisitos do sistema em um alto nível de abstração e leva em consideração requisitos funcionais e não-funcionais. Normalmente esta especificação é feita por meio de algum tipo de formalismo como por exemplo Redes de Petri [34], Máquina de Estados Finitos [14], Diagrama de Estados [17], Álgebra de Processos [20], etc.

b) **Particionamento**. Esta fase determina quais funcionalidades serão implementadas diretamente no hardware e quais serão implementadas por meio de software. Este processo de balanceamento pode ser executado de maneira automática, ma-

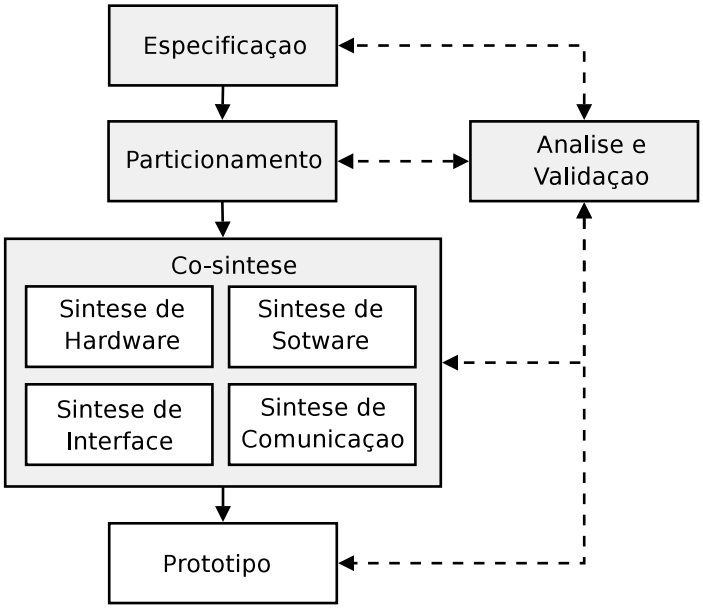

Figura 2.1: Principais Fases da Metodologia Hardware-Software Codesign

nual ou pela combinação de ambas as formas. O particionamento é guiado pelos requisitos estipulados na fase de especificação. Assim, um particionamento pode ser considerado satisfatório à medida que suas estimativas de execução atingem valores previamente estipuladas para métricas como: custo, tempo de execução, consumo de energia, taxa de comunicação, tamanho do software, número de pinos, etc.

c) **Co-síntese**. O resultado obtido da fase de particionamento é um conjunto de módulos, chamado protótipo virtual. A fase de co-síntese tem o papel de mapear este modelo virtual em um modelo real. Nesta fase, decisões de síntese como: interfaces de comunicação; protocolos de comunicação e etc. são tomadas. Esta fase compreende a síntese de hardware, síntese de software, síntese de interfaces (entre módulos de hardware e software) e síntese de comunicação (entre módulos de software em diferentes processadores).

d) **Análise e Validação**. A análise do sistema consiste na confirmação de várias métricas de qualidade. A validação pode acorrer no final de cada fase e pode ser realizada através de simulação ou por meio de um protótipo real. Como o sistema é composto de componentes de software e hardware, essa metodologia pode exigir

a interação entre diversos ambientes de simulação (*co-simulation*).

2.1.4 Softwares Embarcados

Definir quais módulos ou funcionalidades de um sistema embarcado serão implementados através de software é apenas parte do problema. As severas restrições impostas a esses sistemas, bem como as diversas peculiaridades que os tornam diferentes das aplicações de software destinadas aos computadores pessoais, fazem com que as metodologias de desenvolvimento de software convencionais não possam ser diretamente aplicadas ao seu desenvolvimento.

Por outro lado, como já foi citado, o aumento no poder de processamento e redução de custo dos processadores tem impulsionado a migração de funcionalidades de hardware para o software. Esta migração, aliada ao aumento natural de complexidade das aplicações embarcadas (aplicações mais "inteligentes" e "elaboradas"), tem exigido dos projetistas soluções cada vez mais flexíveis e adaptativas, o que normalmente exige um ambiente de desenvolvimento de software bastante poderoso e ao mesmo tempo especializado.

Em [38, 39] Sangiovanni-Vincentelli e Martin apresentam alguns problemas relacionados ao desenvolvimento de softwares embarcados e alguns desafios que qualquer metodologia proposta deve considerar.

Resumidamente os problemas relacionados em [38] são:

- o uso de linguagens de programação que proporcionam alto desempenho, mas que são conhecidamente fracas em legibilidade e manutenibilidade (ex. C e assembly);
- a necessidade de suporte de hardware específico para depuração e avaliação de desempenho;
- paradigmas de programação que não fornecem garantias suficientes de que parâmetros de qualidade e prazos serão de fato atingidos;
- dificuldade de verificação da corretude do projeto;

- pouca atenção dada a restrições relacionadas a *deadline*, uso de memória e consumo de energia.

De acordo com [39], uma metodologia para desenvolvimento precisa obrigatoriamente considerar os seguintes desafios:

- reuso;
- hardware/software codesign;
- modelagem de propriedades não-funcionais;
- uso extensivo de componentes de software;
- arquitetura do sistema e software;
- validação e verificação em nível de sistema;
- adoção de arquiteturas de hardware e software reconfiguráveis e componentes *plug and play*;
- composição do software utilizando componentes reutilizáveis;
- suporte a desenvolvimento paralelo via tecnologia de integração;
- desenvolvimento de processos padronizados.

É importante destacar que para o projeto de softwares embarcados considerados simples, não existe a necessidade de uma metodologia rebuscada. No entanto, uma abordagem empírica não pode ser considerada para aplicações mais complexas.

2.2 Sistemas de Tempo Real

Essa seção inicia com uma introdução que define e apresenta os sistemas de tempo real e suas principais restrições temporais, Em seguida, são mostradas algumas características e classificações para sistemas e tarefas. A seção finaliza com uma breve discussão sobre o desafio presente na elaboração de sistemas de tempo real.

2.2.1 Introdução

Sistemas de tempo real são aqueles onde não somente o resultado lógico da computação é importante, mas também o tempo no qual este resultado é obtido [44]. Mesmo sendo uma definição simples é necessário esclarecer que computação em tempo real não tem o mesmo significado de computação rápida. Computação rápida tem o objetivo simples de conseguir resultados o mais rápido possível, enquanto a computação em tempo real tem o objetivo de obter resultados obedecendo a restrições temporais previamente estipuladas.

Segundo [22] duas abordagens diferentes para este tipo de sistemas podem ser identificadas: orientado por eventos (*event-triggered*) e orientada por tempo (*time-triggered*). Na abordagem orientada por eventos as atividades do sistema são iniciadas em reposta e estímulos internos ou externos (ocorrência de um evento significativo). Na abordagem orientada por tempo as atividades são iniciadas em momentos predeterminados.

As principais restrições temporais as quais um sistema de tempo real esta sujeito são: (i) *período*, indicando a execução de tarefas em intervalos regulares; (ii) *tempo limite (deadline)*, correspondendo ao tempo limite para a finalização da execução de uma tarefa; (iii) *tempo de chegada*, que é o instante no tempo em que o sistema toma conhecimento da chegada de uma tarefa ; (iv) *tempo de início*, indicando o instante em que uma tarefa começa a ser processada; (v) *tempo de execução ou computação*, que é o tempo necessário para que uma tarefa seja executada; (vi) *tempo de finalização*, que é o instante em que uma tarefa é finalizada; (vii) *tempo de liberação (release)*, que indica o instante em que uma começa de fato a ser executada.

As restrições temporais só podem ser garantidas se os valores referentes a execução do pior caso (WCET - *worst-case execution time*) de cada uma das tarefas que compõem o sistema for conhecido previamente. Este valor é definido pelo limite superior de tempo entre a ativação (tempo de início) de uma tarefa e sua finalização.

2.2.2 Características dos Sistemas de Tempo Real

Segundo [44], sistemas de tempo real podem ser caracterizados pelos seguintes fatores:

a) **Respostas condicionadas ao tempo**. Fornecer a resposta correta não é o único objetivo. A característica mais importante é que as repostas aos estímulos externos precisam ocorrer em um intervalo de tempo pré-determinado;

b) **Comportamento previsível**. Outra característica é que o sistema precisa possuir comportamento determinístico. Isto é, todas as execuções do sistema devem ocorrer de maneira similar, sendo possível determinar quando cada tarefa será executada;

c) **Robustez**. O sistema deve ser imune a pequenas mudanças em seu estado, de modo que possa continuar executando sua atividade sem degradação em relação ao que foi inicialmente projetado. Sobrecargas, falhas de hardware, mudanças de ambiente, etc. não devem afetar o desempenho do sistema como um todo;

d) **Precisão nas repostas**. O sistema não deve ser somente previsível e robusto, mas também preciso. Em muitos casos não é simples conseguir precisão nas respostas cumprindo todos os requisitos temporais. Nessa situação, um balanceamento entre precisão e tempo de computação precisa ser realizado de acordo com contexto e aplicação;

e) **Concorrência entre as tarefas**. Sistemas de tempo real podem ser projetados de forma distribuída, considerando mais de um processador, diversos sensores independentes, etc. Desse modo, concorrência é de fato uma característica bastante comum nesses sistemas. No entanto esse paralelismo acarreta algumas outras complexidades, sendo: (i) processos paralelos precisam ser escalonados corretamente de modo que as restrições temporais possam ser cumpridas; (ii) a sincronização das tarefas paralelas pode não ser uma atividade trivial; (iii) a comunicação entre os processos paralelos pode significar uma carga adicional ao sistema; e (iv) o sistema é mais suscetível a falhas, uma vez que possui um número maior de unidades de processamento.

2.2.3 Classificação dos Sistemas de Tempo Real

Os sistemas de tempo real podem ser classificados em duas grandes categorias: os sistemas de tempo real críticos (*hard real-time systems*) e os sistemas de tempo real brandos (*soft real-time systems*).

a) **Os sistemas críticos**. Exigem garantia completa quanto à previsibilidade de suas respostas e comportamento. São geralmente utilizados para tarefas críticas, onde qualquer falha ou atraso pode significar perda de vidas humanas e grandes prejuízos materiais. Assim, são normalmente sistemas mais robustos que os convencionais.

b) **Os sistemas brandos**. Não possuem requisitos tão restritos quanto os sistemas críticos e buscam um balanceamento entre tempo de computação e precisão de suas respostas. Nesse tipo de sistema, o não cumprimento de uma restrição temporal não representa, obrigatoriamente, um evento crítico, podendo significar apenas uma perda de desempenho em relação ao desejado.

2.2.4 Classificação das Tarefas em Sistemas de Tempo Real

Em relação à periodicidade, as tarefas que compõem um sistema podem ser classificadas em três categorias:

a) **Tarefas Periódicas**. São tarefas que ocorrem em intervalos regulares chamados de período. As restrições temporais deste tipo de tarefa são geralmente representadas pela periodicidade, tempo de computação (WCET), tempo limite para finalização (*deadline*) e tempo de liberação (*release*).

b) **Tarefas Aperiódicas**. São tarefas que ocorrem em intervalos de tempo aleatório.

c) **Tarefas Esporádicas**. São tarefas que também ocorrem em intervalos de tempo aleatório, mas que possuem a garantia de um intervalo mínimo, previamente conhecido, entre duas ocorrências consecutivas. As restrições temporais deste tipo de

tarefa são geralmente representadas pelo tempo de computação, tempo limite para finalização e intervalo mínimo entre ocorrências.

Como previsibilidade é característica marcante em sistemas de tempo real, os sistemas são normalmente modelados em função de tarefas periódicas e esporádicas. De outro modo, é impossível garantir que todas as restrições temporais possam ser cumpridas [60].

2.2.5 Principal Desafio

Na prática, a principal característica dos sistemas de tempo real, as restrições temporais, representa também, a maior dificuldade para especificação, análise e validação desse tipo de sistemas.

Muitos dos métodos propostos apresentam grandes dificuldade de utilização dos formalismos que normalmente fazem uso ou, simplesmente, não se apresentam como soluções práticas. Entretanto, prejuízos e diversos acidentes poderiam ser evitados caso estes métodos fossem de fato utilizados.

2.3 Escalonamento em Sistemas de Tempo Real

Escalonamento tem sido um assunto estudado por muitos anos e a quantidade de resultados das pesquisas nesta área é muito vasta [1, 2, 5, 8, 19, 23, 25, 41, 42, 53, 59, 56, 58, 61, 62]. No entanto, esta seção se propõe a apresentar somente uma breve revisão das contribuições e conceitos relacionados a escalonamento envolvendo restrições temporais. Os conceitos fornecidos restringem-se àqueles que possam auxiliar na compreensão do restante do texto.

A seção inicia com uma breve introdução sobre escalonamento, seguida da apresentação das abordagens de escalonamento estática e dinâmica. Após isso, a complexidade algorítmica do problema de escalonamento é discutida. Por fim, algumas vantagens do uso de escalonamento estático em relação ao dinâmico são apresentadas.

2.3.1 Introdução

Como citado anteriormente (Seção 2.2), sistemas de tempo real são geralmente compostos por um conjunto de tarefas concorrentes e com restrições temporais próprias. Nesse contexto, o escalonamento é o processo que define a ordem de execução das tarefas pertencentes a um sistema, sendo, deste modo, um elemento chave na busca por um comportamento previsível, controlado e que possa atender todas as restrições impostas ao sistema.

As abordagens para escalonamento de tarefas podem ser caracterizadas em relação ao comportamento das tarefas ou em relação ao instante no tempo no qual as decisões são tomadas (instante em que a escala é gerada).

Com relação ao comportamento das tarefas, a abordagem para escalonamento pode ser: (i) *preemptiva*, quando uma tarefa em execução pode ser interrompida por outra; ou (ii) *não-preemptiva*, quando esta interrupção não é permitida. Porém, em termos práticos, a idéia de preempção é mais bem representada quando associada à tarefa e não ao tipo de escalonamento executado. Desse modo, um único processo de escalonamento pode contemplar tanto tarefas preemptivas como não-preemptivas.

Quanto ao instante em que as decisões são tomadas, a abordagem de escalonamento pode ser: (i) *dinâmica*, também denominada de escalonamento em tempo de execução (*runtime*)[1]; (ii) *estática*, também chamada de escalonamento em tempo de projeto (*pre-runtime*)[2]; ou (iii) *híbrida*, quando uma parte das tarefas é escalonada estaticamente e outra parte dinamicamente.

A seguir são apresentados detalhes sobre *escalonamento dinâmico* e *escalonamento estático*.

[1]Os termos "escalonamento dinâmico" e "escalonamento em tempo de execução" serão utilizados como sinônimos durante este trabalho, de acordo com o que for mais conveniente à clareza do texto

[2]Os termos "escalonamento estático" e "escalonamento em tempo de projeto" serão utilizados como sinônimos durante este trabalho, de acordo com o que for mais conveniente à clareza do texto

2.3.2 Escalonamento Dinâmico

A abordagem de escalonamento em tempo de execução toma decisões durante a execução do sistema, selecionando uma dentre as tarefas que estejam prontas para serem executadas. Normalmente, essa seleção é baseada em algum tipo de prioridade. Estas prioridades podem ser atribuídas dinamicamente (calculadas durante a execução do sistema) ou estaticamente (determinadas em tempo de projeto).

Alguns dos algoritmos para escalonamento em tempo de execução são apresentados a seguir. Neste contexto, é assumido que n representa o número de tarefas e c_i, d_i e p_i representam, respectivamente, o tempo de computação, *deadline* e o período da tarefa τ_i.

Rate Monotonic Scheduling (RMS). É um algoritmo baseado em prioridade estática e proposto em [25]. É um método válido somente para um conjunto restrito de aplicações, uma vez que suas premissas definem um modelo de tarefas muito simples, sendo que (i) todas as tarefas são periódicas e independentes; (ii) o *deadline* de cada tarefa é igual ao seu período; (iii) o tempo de computação de cada tarefa é previamente conhecido e constante; e (iv) a sobrecarga resultante do chaveamento de contexto é ignorada. No RMS, a tarefa que possuir o período mais curto tem maior prioridade. Além disso, a análise para determinar se é possível encontrar uma escala que satisfaça todas as restrições temporais (análise de escalabilidade) pode ser feita em tempo de projeto, por meio da seguinte inequação:

$$U = \sum_{i=1}^{n} (c_i/p_i) \leq n(2^{1/n} - 1)$$

Em outras palavras, se o fator de utilização U é menor que $n(2^{1/n} - 1)$, então o conjunto de tarefas é escalonável. Em [23] é apresentada uma extensão chamada *deadline monotonic scheduling (DMS)* que permite *deadlines* menores que o período das tarefas.

Earliest Deadline First (EDF) [25]. É um algoritmo baseado em prioridade dinâmica e com as mesmas premissas que o RMS. Após qualquer evento significativo a tarefa que possuir o deadline mais eminente recebe a maior prioridade e é selecionada

para a execução. A análise de escalabilidade é obtida por meio da seguinte inequação:

$$U = \sum_{i=1}^{n} (c_i/p_i) \leq 1$$

Priority Ceiling Protocol (PCP) [40]. Possui as mesmas premissas que o RMS e é utilizado para tarefas que possuam regiões críticas protegidas por semáforos.

Um conjunto de tarefas periódicas pode ser escalonado se a seguinte condição é satisfeita:

$$\frac{c_1}{p_1} + \frac{c_2}{p_2} + \cdots + \frac{c_n}{p_n} + \max\left(\frac{B_1}{p_1}, \cdots, \frac{B_{n-1}}{p_{n-1}}\right) \leq n(2^{1/n} - 1)$$

onde B_i é o pior caso de tempo de bloqueio da tarefa τ_i por qualquer processo com prioridade menor.

2.3.3 Escalonamento Estático

No escalonamento em tempo de projetos, as principais características temporais das tarefas precisam ser conhecidas antecipadamente. Desse modo, uma escala ótima pode ser determinada antes da execução do sistema, e o projetista pode ter absoluta certeza de que todos as restrições temporais serão cumpridas.

Este abordagem somente suporta tarefas periódicas. Para que tarefas esporádicas possam ser incluídas no sistema, estas necessitam ser, primeiramente, convertidas em tarefas periódicas [30, 60], o que somente é possível nos casos em que um tempo mínimo entre duas execuções consecutivas é garantido e previamente conhecido, e o deadline não é muito curto. Entretanto, essa estratégia pode resultar em sobrecarga, ou redução significativa, no fator de utilização dos processadores.

Dado um conjunto de tarefas pertencentes a um sistema, o período completo para a escala a ser encontrada é igual ao mínimo múltiplo comum (MMC) dos períodos de cada tarefa[60].

Várias técnicas e algoritmos foram propostos para escalonamento em tempo de projeto. Xu e Parnas [59] propõe um algoritmo que adota uma técnica *branch-and-bound*, onde

um grande número de escalas é analisado com o objetivo de encontrar uma escala ótima. Tal trabalho foi a primeira tentativa de formalizar um método de escalonamento estático para sistema em tempo real que considerem relações de exclusão e precedência entre as tarefas. Shepard e Gagné [42] estende esta abordagem para lidar com multiprocessadores, e Abdelzaher e Shin [1] a estende para sistemas de tempo real distribuídos.

Esse trabalho adota uma abordagem de escalonamento em tempo de projeto baseada no algoritmo proposto em [6]. Este algoritmo utiliza a técnica *depth-first* aliada a métodos de poda para encontrar uma escala viável.

2.3.4 Complexidade Algorítmica

O problema de encontrar uma escala viável para um conjunto de tarefas não preemptivas, considerando tempo de liberação e *deadlines*, em uma arquitetura mono ou multiprocessada é NP-completo [15]. Entretanto, quando as tarefas consistem de seguimentos únicos que podem ser preemptadas por qualquer outra tarefa (tarefas independentes), o problema pode ter uma solução em tempo polinomial, mesmo se vários processadores forem considerados [25].

Muitos trabalhos utilizam heurísticas, *branch-and-bound*, *depth-first* e outras técnicas para encontrar escalas viáveis considerando relações de precedência e exclusão. Essas técnicas são geralmente aplicadas estaticamente (em tempo de projeto). Porém, mesmo nesses casos, o problema é NP-Completo [59]. Isso se dá devido ao fato que qualquer algoritmos que utilize tais técnicas e considere um sistema mono processado com relação de exclusão entre as tarefas, precisa lidar com o caso especial onde todas as tarefas se excluem mutuamente, o que torna o problema idêntico ao estudado em [15].

O problema se torna mais complexo quando uma arquitetura multiprocessada é considerada. Nesta situação, a alocação e escalonamento são problemas NP-completos mesmo se o objetivo for simplesmente a minimização do tempo de execução

2.3.5 Vantagens do Escalonamento Estático

Tanto o escalonamento em tempo de execução quanto o escalonamento em tempo de projeto possuem vantagens e desvantagem. Entretanto a abordagens em tempo de projeto tem se mostrado mais adequada a sistemas de tempo real, uma vez que esta resulta em um comportamento mais previsível. Uma comparação mais detalhada entre essas duas abordagens pode ser encontrada em [61].

A abordagem de escalonamento em tempo de execução exige mais recursos do sistema que a abordagem em tempo de projeto, uma vez que a própria tarefa de escalonamento consume recursos, o que pode sobrecarregar o sistema e afetar diretamente a previsibilidade esperada, podendo, em algumas situações, implicar no não cumprimento de restrições temporais (perda de *deadline* de tarefas). Por outro lado, a sobrecarga do escalonamento em tempo de projeto é bastante reduzida e controlada, já que, juntamente com as tarefas do sistema, apenas um pequeno despachante precisa ser executado em cada troca de contexto.

Mesmo sendo comum a existência de aplicações reais compostas por diversas tarefas com algum tipo de relação entre si (precedência e exclusão, por exemplo), se a abordagem for utilizada, não é uma questão trivial estender a análise de escalabilidade para considerar tais restrições. Isso acontece porque, em muitas situações, essas restrições adicionais são conflitantes com as prioridades existentes, sendo impossível mapeá-las em uma hierarquia de prioridades. Esta mesma dificuldade não se apresenta no escalonamento estático.

Além disso, a abordagem dinâmica exige um mecanismo complexo para controlar a execução de forma a possibilitar sincronismo e evitar acessos simultâneos a recursos compartilhados. Este mecanismo representa uma sobrecarga adicional ao sistema, e situações que conduzam a impasses (*deadlocks*) podem ser inevitáveis. Mais uma vez, esta não é uma preocupação para a abordagem estática, já que uma vez encontrada uma escala, esta será garantidamente livre de estados bloqueantes.

Uma outra desvantagem da abordagem dinâmica é que esta tem chances menores de encontrar uma escala viável que a abordagem estática. Isso pode ser visto no seguinte

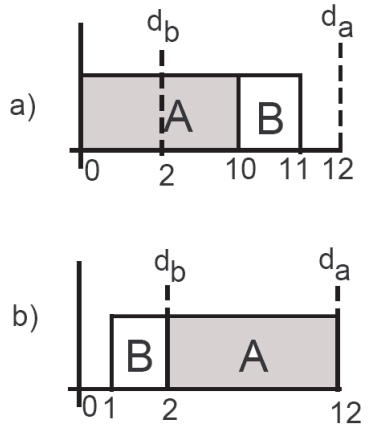

Figura 2.2: Comparação entre Escalonamento Dinâmico (a) e Estático (b)

exemplo: dadas duas tarefas $A = (0, 10, 12)$ e $B = (1, 1, 2)$, com seus respectivos tempos de liberação, computação e deadlines; e com a restrição adicional de B não poder ser preemptada por A, a abordagem dinâmica não é capaz de encontra uma escala (Figura 2.2 (a)) Entretanto, a abordagem estática consegue encontrar uma escala (Figura 2.2(b)) deixando o processador ocioso no intervalo entre 0 e 1, mesmo estando a tarefa A liberada para execução (o tempo de liberação de A é 0)

2.4 Redes de Petri

Esta seção apresenta conceitos básicos sobre o formalismo adotado neste trabalho, chamado redes de Petri. Após uma breve introdução, é apresentada a classe de rede de Petri chamada lugar-transição, que será utilizada como componente básico na maior parte das definições fornecidas no restante do texto. Em seguida, são mostradas as regras de habilitação e disparos de uma transição em uma rede de Petri e algumas estruturas elementares, a partir das quais é possível construir redes que representem modelos mais elaborados. Após isso, discorre-se sobre a extensão temporal para redes Petri utilizada neste trabalho. A seção finaliza com a apresentação de algumas propriedade e um resumo dos métodos de análise em redes de Petri.

2.4.1 Introdução

Em 1962 Carl Adam Petri apresentou à Faculdade de Matemática e Física da Universidade Técnica de Darmstadt na Alemanha, sua tese de doutorado contendo uma nova abordagem para modelagem e análise de comunicação entre autômatos [35]. Esta teoria se baseava em um tipo de grafo bipartido[1] com estados associados. Posteriormente, essa teoria chamou atenção da comunidade científica e passou a ser conhecida desde então como redes de Petri.

Diversas extensões tem sido propostas em relação ao modelo original. Alguns exemplos significativos dessas extensões são: arco inibidor, redes temporizadas estocásticas e determinísticas, redes de alto nível como redes predicado/transição e redes coloridas.

As potencialidades das redes de Petri têm sido aplicadas nos mais diversos contextos. Conceitos e relações comuns ao domínio computacional, como por exemplo: paralelismo e concorrência, compartilhamento de recursos, sincronização, memorização, limitação de recursos, podem facilmente ser representados.

Este trabalho assume como modelo básico uma das variantes mais difundidas de redes de Petri, conhecida como redes de Petri lugar-transição. Esta classe de rede é tão amplamente utilizada que algumas vezes é chamada simplesmente de rede de Petri[3] [13].

2.4.2 Redes de Petri Lugar-Transição

Uma rede de Petri lugar-transição é um grafo direcionado bipartido, representado pela tupla $PN = (P, T, F, W, m_0)$[4] tal que,

- $P = \{p_1,p_2,...,p_n\}$;
- $T = \{t_1,t_2,...,t_m\}$;

[1] Um grafo é dito bipartido quando seus vértices podem ser divididos em dois conjuntos, tais que nenhum vértice pertencente a qualquer um dos conjuntos esteja ligado a um outro vértice no mesmo conjunto

[3] Exceto quando explicitamente indicado, esta será a nomenclatura adotada neste texto

[4] A nomenclatura utilizada neste trabalho para nomes de conjuntos e seus elemento, segue a mesma convenção utilizada na maior parte da literatura disponível, ou seja, inicias ou acrônimos dos termos em inglês

- $F \subseteq (P \times T) \cup (T \times P)$ é o conjunto de arcos;
- $W : F \to \mathbb{N}^+$ é uma função peso para cada arco;
- m_0 é a marcação inicial.

Esta classe de rede consiste de dois conjuntos distintos de nós, chamados lugares (*places*) e transições (*transitions*), ou segundo a nomenclatura adotada, P e T, tal que $P \cap T = \emptyset$ e $P \cup T \neq \emptyset$. O conjunto de arcos F representa as conexões entre os lugares e as transições ou entre as transições e os lugares. A função W atribui um peso a cada arco do conjunto F. A função peso é também chamada de *multiplicidade* do arco. Assim, um arco possui multiplicidade k quando seu peso for k. Uma rede de Petri marcada possui marcas, também denominadas *tokens*[5], que residem nos lugares. A distribuição inicial dessas marcas nos lugares da rede é chamada marcação inicial (m_0).

O fluxo das marcas através da rede é determinado disparo das transições, sendo que o conjunto de todas as marcações alcançáveis é indicado por $m = \{m_0, m_1, \cdots, m_i, \cdots\}$, onde m_0 representa a marcação inicial. O vetor $m_i = (m_{i_1}, m_{i_2}, \cdots, m_{i_n})$ representa a quantidade de marcas em cada lugar na marcação m_i, e pode também ser definido com uma função $m_i : P \to \mathbb{N}$. Isso permite utilizar a notação $m_i(p_j)$ para especificar o número de *tokens* no lugar p_j na marcação m_i. Desse modo, neste trabalho, as notações m_{i_j} e $m_i(p_j)$ são consideradas equivalentes, podendo ser utilizado qualquer uma das formas.

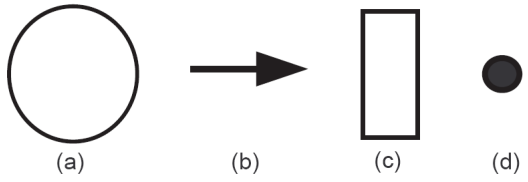

Figura 2.3: Representação Gráfica para Rede de Petri: (a) Lugar, (b) Arco, (c) Transição e (d) *Token*

Mesmo tratando-se de um formalismo matemático, redes de Petri possuem uma representação gráfica usual que facilita a visualização e compreensão dos sistemas represen-

[5]Por se tratar de um termo consagrado pelo uso, a palavra inglesa *token* será freqüentemente utilizada no lugar de seu equivalente em português

tados (Figura 2.3). Nesta simbologia, os lugares (a) são, normalmente, representados por círculos, arcos (b) são representados por setas, transições (c) são representadas por barras ou retângulos, e (d) *tokens* são geralmente representados por pequenos círculos preenchidos. A multiplicidade de um arco é indicada por um inteiro adjacente ao arco e pode ser omitida quando possuir valor igual a 1. Alternativamente, um arco com multiplicidade k pode também ser representado como um conjunto de k arcos paralelos.

As redes de Petri são modelos abstratos, e portanto, diversas interpretações distintas para lugares e transições são possíveis, de acordo com o que está sendo modelado. De uma forma genérica os lugares podem ser vistos como depósitos de recursos (representados pelos *tokens*) e as transições como ações que manipulam esses recursos. Usando o conceito de evento e condição, os lugares representam as condições e as transições os eventos, de forma que um único evento pode ter várias pré-condições e pós-condições. Outras interpretações podem ser encontradas em [31].

O conjunto de transições de entrada de um lugar $p_i \in P$ é:

$$^\bullet p_i = \{t_j \in T \mid (t_j, p_i) \in F\}$$

e o conjunto de transições de saída é:

$$p_i {}^\bullet = \{t_j \in T \mid (p_i, t_j) \in F\}$$

O conjunto de lugares de entrada de uma transição $t_j \in T$ é:

$$^\bullet t_j = \{p_i \in P \mid (p_i, t_j) \in F\}$$

e o conjunto de lugares de saída é:

$$t_j {}^\bullet = \{p_i \in P \mid (t_j, p_i) \in F\}$$

Este trabalho, em algumas situações, faz uso da nomenclatura utilizada no contexto de evento e condição, para designar os conjuntos de lugares de entrada e lugares de saída de uma transição $t_j \in T$. Assim, os lugares de entrada da transição t_j serão também chamados de pré-condições de t_j, e os lugares de saída serão chamada pós-condições de t_j.

Habilitação e Disparo de Transições

Com o objetivo de representar o comportamento de um sistema, os estados, caracterizados pelas marcações, sofrem modificações segundo as seguintes regras de disparo das transições:

a) Uma transição t está habilitada se cada um de seus lugares de entrada possui pelo menos o número de *tokens* indicado na multiplicidade do arco que o conecta com t.

b) Uma transição habilidade pode ou não disparar.

c) O disparo de uma transição habilitada t remove de cada lugar de entrada a quantidade de *tokens* indicada pela multiplicidade do arco que liga o lugar de entrada a transição, e adiciona a cada lugar de saída a quantidade de *tokens* indicada pela multiplicidade do arco que liga a transição ao lugar de saída.

Como forma de ilustrar estas regras, a Figura 2.4 ilustra (a) o formalismo para uma rede com três lugares e uma transição; (b) a representação gráfica desta rede; e (c) a mesma representação após o disparo da transição t_1.

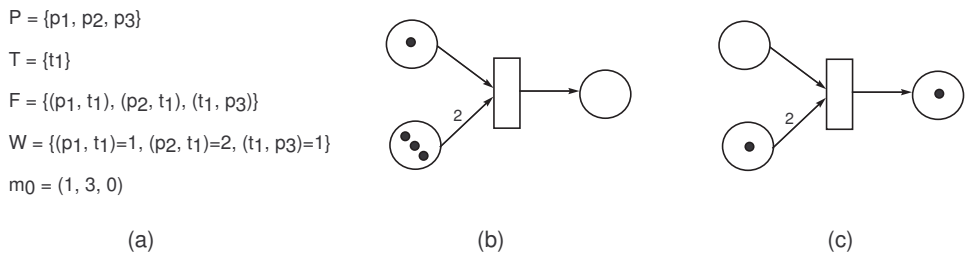

Figura 2.4: Disparo de uma Transição em Rede de Petri. (a) Formalismo; (b) Antes do Disparo de t_1; (c) Após o Disparo de t_1

Ainda em relação a disparo, merecem destaques as seguintes situações: (i) *transição fontes* (*source transition*), que é uma transição sem pré-condições e que está sempre habilitada; (ii) *transição solvedora* (*sink transition*), que é uma transição sem pós-condição e que ao disparar consome *tokens* mas não produz nenhum; e (iii) *auto-laço* (**self-loops**), que é um par formado por um lugar p e uma transição t, no qual o lugar p é tanto pós-condição quanto pré-condição de t. Uma rede é dita *pura* se não possuir auto-laços.

2.4.3 Redes de Petri Elementares

Redes elementares são blocos básicos a partir dos quais redes maiores e mais complexas podem ser montadas. Algumas dessas estruturas são:

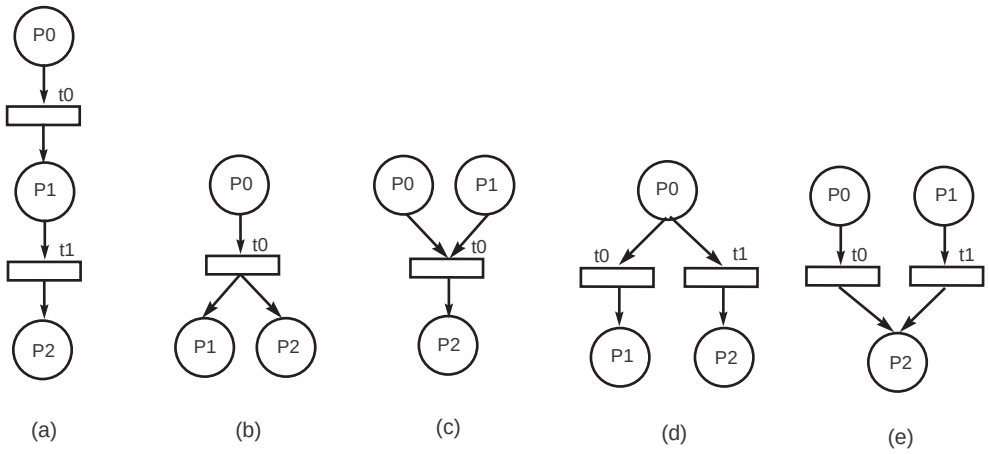

Figura 2.5: Redes de Petri Elementares

a) **Seqüência**. É uma estrutura que representa a execução seqüencial de ações (Figura 2.5(a)).

b) **Disjunção** (*Fork*). É uma estrutura que permite a criação de processos paralelos (Figura 2.5(b)).

c) **Junção**. É uma estrutura que sincroniza dois processos paralelos (Figura 2.5(c)).

d) **Conflito**. É uma estrutura que representa a escolha entre duas possibilidades. Se uma for escolhida a outra é imediatamente desabilitada (Figura 2.5(d)).

e) **Mesclagem** (*Merging*). É uma estrutura que permite que processos distintos habilitem uma mesma condição (Figura 2.5(e)).

2.4.4 Extensões Temporais em Redes de Petri

A definição original para redes de Petri tem como objetivo modelar somente o comportamento lógico dos sistemas, descrevendo apenas uma relação de causa e efeito entre os eventos. No entanto, tempo é uma questão essencial para algumas aplicações, como por exemplo, aplicações de tempo real.

Existem diversas maneiras de incorporar tempo nas redes de Petri. O tempo pode ser associado aos lugares, às transições, aos arcos e até mesmo aos *tokens*. No entanto, como as transições representam as mudanças no estado do sistema, parece mais natural que o tempo seja associado a estas.

É importante destacar que a inserção de tempo não modifica a estrutura básica de uma rede de Petri. Porém, estende algumas definições comportamentais. Desse modo, quando tempo é considerado, pelo menos duas políticas de disparo de transições e três políticas para manutenção e contagem de tempo podem ser observadas

As políticas para disparo das transições são:

a) **Disparo em três fases**. Primeiramente uma transição habilitada remove os *tokens* dos lugares de entrada. Em seguida, aguarda um período de tempo chamado duração. Finalmente, deposita os *tokens* nos lugares de saída. O disparo da transição consome tempo.

b) **Disparo atômico**. Os *tokens* permanecem nos lugares de entrada enquanto a transição não é disparada. Quando o disparo acontece, eles são instantaneamente consumidos dos lugares de entrada e gerados nos lugares de saída. O disparo da transição não consumo tempo.

As políticas para manutenção e contagem de tempo são:

a) **Sem memória temporal**. No disparo de qualquer transição, os relórios (*clocks*) de todas as transições são descartados e nova contagem é reiniciada para as transições habilitadas na nova marcação (mecanismo de reinício).

b) **Com memória de habilitação**. No disparo de qualquer transição, os relógios das transições desabilitadas são reiniciados, enquanto os relógios das transições que permanecem habilitadas conservam seus valores.

c) **Com memória temporal**. No disparo de qualquer transição, todos os relógios conservam seus valores (mecanismo contínuo).

Diversas extensões temporais para redes de Petri têm sido propostas. A extensão utilizada neste trabalho, denominada redes de Petri com temporização[6], é detalhada a seguir. Entretanto, com o objetivo de fazer distinção entre os conceitos e consolidar a nomenclatura adotada, uma breve definição de redes de Petri temporizadas[7] também é apresentada.

Rede de Petri com Temporização

Uma rede de Petri com temporização [28] é definida pelo par (PN, I), onde PN é uma rede Petri lugar-transição, e I é uma função que associa um intervalo delimitado por números naturais, a cada transição do conjunto de transições de PN ($I : T \rightarrow \mathbb{N} \times \mathbb{N}$). Desse modo, $I(t)$[8] associa a t um intervalo fechado, representando as restrições temporais de t, definido por (TDI_t, TDF_t), sendo $TDI_t \leq TDF_t$, onde TDI indica o *tempo de disparo inicial*[9] e TDF indica o *tempo de disparo final*[10]. Este intervalo não negativo expressa o tempo mínimo e máximo de disparo para a respectiva transição. A política de disparo adotada é o disparo atômico.

Uma transição t_i que se torna habilitada no instante de tempo θ, somente poderá disparar no intervalo $(\theta + TDI_t) \leq \delta \leq (\theta + TDF_t)$. Entretanto, atingido o limite de

[6] Adaptação da nomenclatura em inglês: *time Petri nets*
[7] Adaptação da nomenclatura em inglês: *timed Petri nets*
[8] As notações $X(t)$ e X_t serão usada como equivalentes
[9] Adaptação da expressão presente na bibliografia em inglês: *EFT (Earliest Firing Time)*
[10] Adaptação da expressão presente na bibliografia em inglês: *LFT (Lateste Firing Time)*

tempo $\theta + TDF_t$, dois comportamentos distintos podem ser aceitos, dependendo do modo de disparo adotado. No *modo de disparo obrigatório* a transição t_i é forçada a disparar. No *modo de disparo não-obrigatório*, a transição não é obrigada a disparar (o disparo pode acontecer ou não).

É importante observar que segundo esta definição, as redes de Petri não temporizadas são um caso particular das redes de Petri com temporização, no qual todas as transições estariam associadas a um intervalo de disparo com $TDI = 0$ e $TDF = \infty$. Também vale ressaltar que o conjunto de marcações alcançáveis de uma rede de Petri com temporização, é igual ou é um subconjunto do conjunto de marcações da rede não temporizada equivalente, uma vez que a informação temporal pode somente restringir este conjunto, nunca ampliá-lo.

Rede de Petri Temporizadas

Uma rede de Petri temporizada [36] é definida pelo par (PN, D), onde PN é uma rede de Petri lugar-transição e D é uma função que associa um número real não-negativo a cada transição do conjunto de transições de PN ($I : T \rightarrow \mathbb{R}^+$). Este valor é conhecido como tempo de duração da transição.

As regras para considerar uma transição habilitada são as mesmas consideras nas redes de Petri lugar-transição. O disparo das transições obedece a uma política de *disparo em três fases* e as transições precisam disparar tão logo se tornem habilitadas.

2.4.5 Propriedades das Redes de Petri

Em adição ao poder de representação das redes de Petri, a análise e verificação de suas propriedades fornecem uma excelente ferramenta na validação de diversas características do sistema modelado.

Duas classes de propriedades podem ser consideradas: as propriedades influenciadas pela marcação inicial, conhecidas como propriedades comportamentais, e as propriedades

não influenciadas pela marcação inicial, conhecidas como propriedades estruturais. A seguir é apresentado um resumo, baseado em [26], de algumas dessas propriedades.

A principais propriedades comportamentais são:

a) **Alcançabilidade**. Indica a possibilidade de, a partir de uma marcação inicial, alcançar um determinado conjunto de marcações através dos disparos de transições.

b) **Limitação**. Uma rede de Petri marcada é dita k-limitada quando nenhum dos lugares pode acumular um número de *tokens* maior que k.

c) **Segurança**. A propriedade de segurança é um caso particular da propriedade limitação. Uma rede de Petri é segura se todos os lugares possuírem um ou nenhum *token* (se for 1-limitada), em qualquer marcação alcançável.

d) **Vivacidade**. Uma rede de Petri marcada é dita viva se, não importando qual seqüência de transições tenha sido disparada para atingir a marcação atual, é sempre possível disparar alguma transição de modo a avançar para outra marcação. A ausência de *impasses* (*deadlocks*) é um conceito relacionado ao conceito de vivacidade. De fato, se uma rede é livre de impasses, isto não significa obrigatoriamente que ela seja viva, entretanto se uma rede é vida, ela certamente será livre de impasses. Exemplos de redes livres de impasse mas que ainda assim não são vidas, são aquelas que não possuem nenhum estado de impasse mas possuem pelo menos uma transição que nunca é disparada.

e) **Cobertura**. É um conceito que tem relação estreita com alcançabilidade. Se uma marcação m' cobre uma marcação m, isso significa que m pode ser alcançada a partir de m'.

f) **Persistência**. Se para qualquer par de transições habilitadas, o disparo de uma delas não desabilita a outra.

g) **Reversibilidade**. Uma rede de Petri é dita reversível quando a partir de uma marcação qualquer é possível sempre alcançar a marcação inicial.

h) **Justiça** Diz respeito número de disparos de uma transição em relação à outra. Se este número é finito, a rede é dita justa.

As principais propriedades estruturais são:

a) **Limitação Estrutural**. Uma rede de Petri é estruturalmente limitada se for limitada para qualquer marcação inicial.

b) **Conservação**. Uma rede de Petri é dita conservativa se o disparo de qualquer transição não altera somatório ponderado (pesos são associados aos lugares) de *tokens* na rede.

c) **Repetitividade**. Uma rede de Petri é repetitiva se existir uma marcação inicial e uma seqüência de disparos, a partir dessa marcação, na qual todas as transições da rede sejam infinitamente disparadas

d) **Consistência**. Uma rede de Petri é considerada consistente se existir uma marcação inicial e uma seqüência de disparo que consiga partir e retornar a esta marcação inicial e na qual todas as transições da rede sejam disparadas pelo menos uma vez.

2.4.6 Métodos de Análise em Redes de Petri

A seguir é apresentada uma possível classificação para os métodos de análise em redes de Petri [31], sendo estes: (a) enumeração; (b) redução ou decomposição; e (c) matriz de incidência e equação de estado.

a) **Análise por Enumeração**. Este método se baseia na construção de um grafo representando todas as marcações alcançáveis a partir da marcação inicial. Neste grafo, cada vértice corresponde a uma marcação alcançável e cada aresta corresponde ao disparo de uma transição ou de um conjunto não vazio de transições. Sendo possível a construção deste gráfico, a verificação das propriedades pode ser executada por simples navegação ao longo do grafo. Entretanto, devido sua natureza combinatória (que resulta no problema conhecido como *explosão de estados*),

essa abordagem pode se mostrar inviável em alguns casos. Em [31] é detalhado um algoritmo para construção desse grafo.

b) **Análise por redução (ou transformação)**. Consiste na redução de um modelo (rede de Petri) complexo em um modelo mais simples e tratável, mas que mantenha as mesmas propriedades que se pretende analisar. Apresentações mais detalhadas podem ser obtidas em [7, 31].

c) **Matriz de Incidência e Equação de Estado**. Este método permite estudar o comportamento de um sistema por meio de equações algébricas. Nesta abordagem utiliza-se a uma matriz de incidência para descrever as ligações entre lugares e transições. Nos casos em que este método pode ser aplicado, evita-se a utilização de técnicas de enumeração.

Capítulo 3

Um Modelo para Sistemas Embarcados de Tempo Real Críticos

Um modelo pode ser definido como a descrição de um sistema por meio de equações, relações matemáticas ou representação gráfica, baseadas nas leis ou princípios que governam o sistema modelado. Normalmente, um modelo captura somente um subconjunto das propriedades do sistema modelado. Em outras palavras, o modelo é uma abstração do sistema original.

Desse modo, fazendo uso dos conceitos básicos apresentados anteriormente (Capítulo 2), este capítulo apresenta um modelo computacional para sistemas embarcados de tempo real críticos, baseado na teoria de redes de Petri. O modelo apresentado é uma extensão do modelo proposto em [6].

Este capítulo está organizado do seguinte modo: a primeira seção apresenta um padrão de especificação para sistemas embarcados de tempo real críticos; a segunda seção apresenta os fundamentos computacionais para construção de um modelo para este tipo de sistema; e, finalmente um modelo para e especificação, baseado nos fundamentos computacionais é apresentado.

3.1 Especificação

O processo de modelagem de qualquer tipo de sistema, passa primeiro pela especificação das características e propriedades que descrevem os requisitos de tal sistema. Em um sentido mais rigoroso, não se pode afirma que a especificação modele o sistema, pois esta é apenas a descrição padronizada de um determinado conjunto de propriedades e comportamentos. Entretanto, é baseado nela que o modelo para o sistema pode ser construído.

O padrão para especificação de sistemas embarcados de tempo real crítico utilizado neste trabalho é baseado em um conjunto de tarefas e contempla os seguintes elementos:

a) Restrições temporais e de energia para cada tarefa;

b) Relações entre tarefas (precedência e exclusão);

c) Método de escalonamento para cada tarefas (preemptivo e não-preemptivo);

d) Alocação de tarefas para processadores distintos;

e) Comunicação entre tarefas;

f) Associação de código para cada tarefa.

3.1.1 Restrições Temporais e de Energia

A abordagem de escalonamento utilizada neste trabalho (escalonamento estático) pressupõe que as restrições temporais de cada tarefa sejam previamente conhecidas, e que todas as tarefas são periódicas. Assim, caso o sistema especificado possua tarefas esporádicas, estas devem ser primeiramente convertidas em tarefas periódicas. Uma técnica para transformar tarefas esporádicas em tarefas periódicas pode ser encontrada em [29].

Uma *tarefa periódica* pode ser definida em função de suas restrições temporais e de energia, como segue.

Definição 3.1 (Tarefa Periódica) *Sejam $\mathcal{T}$ o conjunto de tarefas de um sistema, $\mathcal{P}$ o conjunto de processadores; ph_i, r_i, c_i, d_i, $p_i \in \mathbb{N}$; $e_i \in \mathbb{R}^+$; e $proc_i \in \mathcal{P}$; tal que $c_i \leq d_i \leq p_i$. Uma tarefa periódica $\tau_i \in \mathcal{T}$ é definida por (ph_i, r_i, c_i, d_i, p_i, e_i, $proc_i$), onde ph_i é o tempo de deslocamento; r_i é o tempo de liberação; c_i é o tempo de computação; d_i é o deadline; p_i é o período; e_i é a energia consumida; e $proc_i$ é o processador alocado para a tarefa.*

O deslocamento (ph_i) é o atraso associado à primeira solicitação de τ_i após a iniciação do sistema. Sempre que este valor não for especificado, será considerado igual a 0. O período (p_i) indica a periodicidade com que τ_i ocorre, de modo que os valores r_i, c_i e d_i são instantes de tempo em relação ao início do período. O tempo de computação (c_i) é o tempo requerido para executar a tarefa τ_i, considerando o pior caso (WCET). O *deadline* (d_i) indica o tempo limite em que τ_i precisa ser completada. Todas essas restrições temporais são expressas em *unidade de tempo de tarefa* (TTU)[1], que representa a menor unidade na qual uma tarefa pode ser dividida e durante a qual não pode ser preemptada.

A energia consumida (e_i) indica a carga necessária para a execução da tarefa.

Este trabalho considera que em uma arquitetura multi processada, a alocação das tarefas em processadores é uma atividade realizada em tempo de projeto. Esta alocação é determinada através de $proc_i$.

3.1.2 Relações entre Tarefas

As relações entre as tarefas são representadas por precedências e exclusões.

Se a tarefa τ_i precede a tarefa τ_j (τ_i `PRECEDE` τ_j), isto significa que a tarefa τ_j só pode iniciar sua execução após a finalização da tarefa τ_i. Esta relação é normalmente adequada quando existe dependência entre as tarefas, de modo que a tarefa sucessora (τ_j, neste caso) necessite de alguma informação gerada pela tarefa predecessora (τ_i, neste caso).

[1]Do inglês: *Task Time Unit*

Esta relação implicitamente exige uma execução de τ_j para cada execução de τ_i, o que na prática, obriga a especificação de períodos iguais para as duas tarefas.

Se a tarefa τ_i exclui a tarefa τ_j (τ_i `EXCLUI` τ_j), isto significa que a tarefa τ_j não pode iniciar sua execução enquanto a tarefa τ_i está sendo executada. Em um contexto mono processado, isto significa que a tarefa τ_i não pode ser preemptada pela tarefa τ_j. Neste trabalho esta relação é considera simétrica, ou seja, se τ_i `EXCLUI` τ_j isto implica que τ_j `EXCLUI` τ_i

3.1.3 Métodos de Escalonamento

Os dois métodos considerados neste trabalho são: (i) preemptivo e (ii) não-preemptivo.

Uma tarefa τ_i é classificada como preemptiva se sua execução pode ser interrompida por outra tarefa. Se a tarefa é classificada como não-preemptiva, esta interrupção não é permitida e a tarefa será executa sem interrupção até sua conclusão.

As questões relacionadas à decisão de qual classificação utilizar estão além do escopo deste trabalho. Entretanto, é importante observar que, em algumas situações específicas, a classificação de todas as tarefas como não-preemptivas pode dificultar ou mesmo impossibilitar a determinação de uma escala viável.

3.1.4 Comunicação entre Tarefas

Este trabalho considera que a comunicação entre tarefas é realizada ponto-a-ponto, o que implica em uma especificação baseada em pares de tarefas. Assim, quando uma tarefa τ_i necessita se comunicar com outras duas tarefas τ_j e τ_h, esta comunicação é especificada como dois processos de comunicação distintos. Primeiramente, τ_i comunica-se com τ_j e, posteriormente, τ_i comunica-s com τ_h.

A comunicação entre tarefas de um mesmo processador é tratada como uma relação de precedência, o que é perfeitamente adequado ao ambiente altamente restritivo encontrado na maioria dos sistemas embarcados de tempo real, onde este tipo de comunicação é normalmente realizado através de compartilhamento de memória.

Por outro lado, a comunicação entre tarefas em processadores distintos não pode assumir as mesmas premissas, já que neste caso, o tempo e a energia consumida no processo de comunicação não podem ser desprezados. Deste modo, uma nova tarefa representando a comunicação é especificada. A definição para esta nova tarefa é a seguinte:

Definição 3.2 (Tarefa de Comunicação) *Sejam $\mathcal{T}$ o conjunto de tarefas de um sistema; $\mathcal{M}$ o conjunto de tarefas de comunicação; $\mathcal{B}$ o conjunto de barramentos de comunicação; τ_i, $\tau_j \in \mathcal{T}$, onde τ_i = (ph_i, r_i, c_i, d_i, p_i, e_i, $proc_i$) e τ_j = (ph_j, r_j, c_j, d_j, p_j, e_j, $proc_j$), tal que $proc_i \neq proc_j$; $bus_m \in \mathcal{B}$; $ct_m \in \mathbb{N}$ $ce_m \in \mathbb{R}^+$;. Uma tarefa de comunicação $\mu_m \in \mathcal{M}$ é definida por μ_m = (τ_i, τ_j, ct_m, ce_m bus_m), onde τ_i é a tarefa emissora, τ_j é a tarefa receptora, ct_m é o pior tempo de comunicação, ce_m é a energia consumida na comunicação; e bus_m é o barramento utilizado para comunicação.*

Esta definição reforça o conceito de comunicação ponto-a-ponto e restringe a existência de uma tarefa de comunicação a situações onde a comunicação ocorra entre tarefas alocadas em processadores distintos. O barramento (bus_m) é uma abstração do canal de comunicação, e a energia consumida na comunicação (ce_m) está relacionada com o uso do barramento.

3.1.5 Especificação Comportamental

A especificação comportamental do sistema consiste na definição detalhada das atividades que de fato serão executadas por cada tarefa. Esta definição é fornecida na forma de um código fonte que deve estar de acordo com as especificações e restrições da plataforma para o qual o sistema se destina.

3.1.6 Exemplo de Especificação

A Tabela 3.1 mostra um exemplo de especificação parcial de um sistema com seis tarefas, dois processadores e três tarefas de comunicação. Esta especificação é dita parcial por não apresentar o código associado a cada tarefa.

Tabela 3.1: Exemplo de Especificação

Tarefa	deslocamento	liberação	computação	deadline	período	proc/bus	de	para	energia
A	0	0	2	10	30	proc1	-	-	0.3
B	0	2	3	20	30	proc1	-	-	1.2
C	0	4	3	30	30	proc1	-	-	0.4
D	0	0	2	20	30	proc2	-	-	1.2
E	0	2	3	30	30	proc2	-	-	0.5
F	0	0	2	10	30	proc2	-	-	0.5
M1	-	-	1	-	-	bus1	F	A	0.5
M2	-	-	1	-	-	bus1	F	B	0.3
M3	-	-	2	-	-	bus1	B	E	0.2
Relações entre Tarefas									
A PRECEDE B, B PRECEDE E									
A EXCLUI D, F EXCLUI C									
F PRECEDE M1, M1 PRECEDE A									
F PRECEDE M2, M2 PRECEDE B									
B PRECEDE M3, M3 PRECEDE E									

Para cada tarefa periódica (A, B, C, D, E e F), a tabela apresenta as respectivas restrições temporais (deslocamento, liberação, computação, *deadline* e período), bem como o processador no qual a tarefa será executada (proc) e o consumo de energia associado a execução da tarefa (energia).

Para as tarefas de comunicação (M1, M2 e M3) a tabela apresenta os valores referentes ao tempo de computação e consumo de energia, assim como as informações de origem (de) e destino (para) da comunicação e o barramento (bus) sobre o qual a comunicação deve se processar.

Adicionalmente, na parte inferior da tabela, são apresentadas as relações de precedência e exclusão entre as tarefas do sistema. É importante notar que mesmo as tarefas de comunicação são consideradas na especificação das relações. Assim, uma comunicação entre a tarefa F e a tarefa A (como indicado na tarefa de comunicação M1) resulta em duas relações de precedência distintas (F `PRECEDE` M1 e M1 `PRECEDE` A).

3.2 Fundamentos Computacionais

Esta seção apresenta os fundamentos computacionais a partir dos quais a especificação de um sistema embarcado de tempo real crítico pode ser transformada em um modelo. A

estrutura central é uma rede de Petri com temporização e algumas extensões que permitem capturar as questões temporais, de consumo de energia e comportamentais de um sistema.

3.2.1 Fundamentos para um Modelo com Restrições Temporais

A seguir são apresentadas as definições que compõem a fundamentação computacional para o modelo que será posteriormente apresentado. Neste primeiro momento, as questões relacionadas ao consumo de energia não são consideradas.

Definição 3.3 (Rede de Petri com Temporização) *Uma rede de Petri com temporização é um grafo direcionado bipartido representado pela tupla $\mathcal{P} = (\mathcal{PN}, I)$, onde $\mathcal{PN}$ é uma rede de Petri marcada lugar-transição, e $I : T \rightarrow \mathbb{N} \times \mathbb{N}$ é uma função que associa um intervalo fechado a cada transição do conjunto de transições de $\mathcal{PN}$, tal que $I(t) = (TDI(t), TDF(t))\ \forall t \in T$ e $TDI(t) \leq TDF(t)$.*

Como detalhado na Seção 2.4.4, esta classe de rede limita o disparo de uma transição a um intervalo temporal estático delimitado pelos valores de *tempo de disparo inicial* (TDI) e *tempo de disparo final* (TDF). Adicionalmente, o modo de disparo adotado neste trabalho é o *disparo obrigatório*, o que força o disparo das transições quando o limite de tempo é atingido.

A rede de Petri com temporização representada na Figura 3.1 será utilizada como forma de ilustrar algumas das definições que se seguem.

Definição 3.4 (Conjunto de Transições Habilitadas) *Sejam $\mathcal{P}$ uma rede de Petri com temporização, e m_i uma marcação alcançável. O conjunto de transições habilitadas na marcação m_i é definido por:*

$$ET(m_i) = \{t \in T \mid m_i(p_j) \geq W(p_j, t)\ \forall p_j \in P\}.$$

Na rede apresentada na Figura 3.1(a), somente a transição t_0 esta habilitada na marcação m_0.

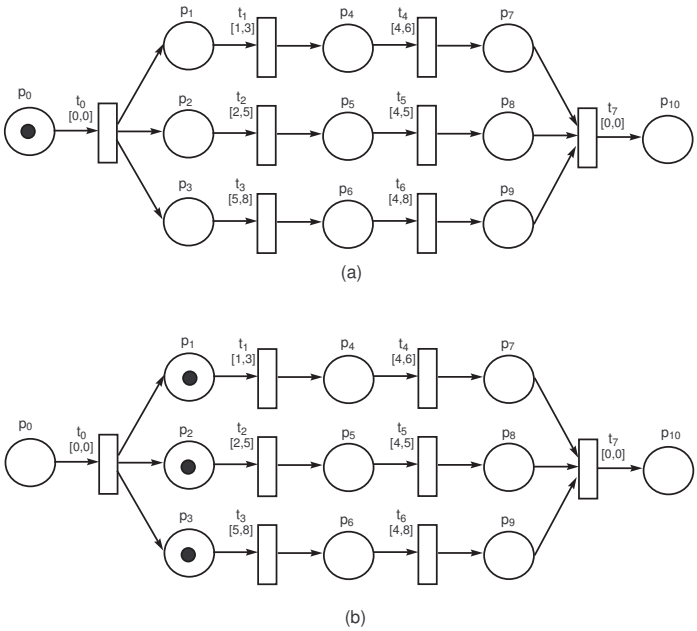

Figura 3.1: Exemplo de Rede de Petri com Temporização: (a) Marcação Inicial; (b) Marcação Alcançada após Disparo de t_0

Definição 3.5 (Relógios) *Sejam $\mathcal{P}$ uma rede de Petri com temporização, e m_i uma marcação alcançável. O relógio é definido por $c_i : ET(m_i) \rightarrow \mathbb{N}$, onde c_i é uma função de temporização, que para cada transição pertencente a $ET(m_i)$ associa o tempo decorrido desde sua habilitação.*

A função de temporização está associada a um conjunto de transições habilitadas. Este conjunto, por sua vez, depende de uma marcação específica. Desse modo, o valor do relógio para a transição t_j na marcação m_i pode ser especificado como $c_i(t_j)$, ou simplesmente, c_{i_j}. Entretanto, como tentativa de simplificar a notação utilizada, sempre que a marcação envolvida não for indispensável à compreensão do texto, a notação $c(j)$ será utilizada

Como especificado na Seção 2.4.4, as redes de Petri com temporização utilizam uma política de *disparo atômica* e uma política de manutenção e contagem de tempo *com memória de habilitação*, o que em termos práticos significa que o disparo não consome tempo, e que após o disparo de uma transição, somente o relógio desta é reiniciado.

Definição 3.6 (Intervalo Dinâmico de Disparo) *Seja* $I(t) = (TDI(t), TDF(t))$ *o intervalo estático de disparo da transição* t*. O intervalo dinâmico de disparo de* t *é definido por* $I_D(t) = (TDI_D(t), TDF_D(t))$*, onde* $TDI_D(t) = \max(0, TDI(t) - c(t))$ *e* $TDF_D(t) = TDF(t) - c(t)$*.*

Logo que t é habilitada, $I_D(t) = I(t)$. No entanto, à medida que o relógio de t é incrementado, $I_D(t)$ é modificado.

Novamente tomando como exemplo a rede apresentada na Figura 3.1, se a transição t_1 se torna habilitada no instante de tempo $\theta = 0$, então $c(t_1) = 0$, $I_D(t_1) = [1, 3]$ e t_i ainda não está apta a disparar por causa suas restrições temporais. No instante $\theta + 1$, $c(t_1) = 1$, $I_D(t_1) = [0, 2]$ e t_i está apta a disparar. Se por qualquer motivo t_1 não disparar, no instante $\theta + 3$, $c(t_1) = 3$, $I_D(t_1) = [0, 0]$, t_1 é forçada a disparar.

Definição 3.7 (Estados) *Sejam* $\mathcal{P}$ *uma rede de Petri com temporização,* M *o conjunto de todas as marcações alcançáveis em* $\mathcal{P}$*, e* C *o conjunto de todos os vetores de temporização de* $\mathcal{P}$*. O conjunto de estados* S *de* $\mathcal{P}$ *é definido por* $S \subseteq (M \times C)$*. Desse modo, um único estado* s *é definido pelo par* (m, c)*, onde* m *é uma marcação e* c *é seu respectivo vetor de temporização para* $ET(m)$*. O estado inicial é* $s_0 = (m_0, c_0)$*, onde* $c_0(t) = 0,\ \forall\ t \in ET(m_0)$*.*

Por esta definição, um estado é caracterizado não somente por sua respectiva marcação, mas também pelo vetor de temporização associado às transições habilitadas nessa marcação. Assim, duas situações determinam as mudanças de estado: (i) o disparo de transições, que resulta em mudanças na marcação e no vetor de temporização, e (ii) a passagem do tempo, que afeta o vetor de temporização.

Neste trabalho, a simples passagem do tempo sem mudança de marcação não captura nenhum aspecto significativo no modelo proposto. Logo, as mudanças de estado resultantes exclusivamente da passagem do tempo (onde não ocorrem mudanças de marcação) não serão consideradas.

Definição 3.8 (Conjunto de Transições Disparáveis) *Seja $s = (m, c)$ um estado de uma rede de Petri com temporização. O conjunto de transições disparáveis no estado s é definido por:*

$$FT(s) = \{t_i \in ET(m) \mid TDI_D(t_i) \leq min(TDF_D(t_k)) \; \forall t_k \in ET(m)\}.$$

Esta definição reforça que estar habilitada é condição necessária, mas não suficiente, para que uma transição possa ser disparada. Assim, $FT \subseteq ET \subseteq T$.

No exemplo da Figura 3.1(b), logo após o disparo de t_0, o conjunto de transições habilitadas (ET) é $\{t_1, t_2, t_3\}$ e o conjunto de transições disparáveis (FT) é $\{t_1, t_2\}$, uma vez que o *tempo dinâmico de disparo inicial* de t_3 ($TDI_D(t_3)$) é igual a 5, e este valor é superior ao menor *tempo dinâmico de disparo final* entre todas as transições habilitadas, que é igual a 3.

Definição 3.9 (Transições Independentes) *Sejam s e s' estados pertencentes ao conjunto de estados de uma rede de Petri com temporização; t_1 e t_2 transições disparáveis em s (t_1,$t_2 \in FT(s)$, e θ um instante no tempo. As transições t_1 e t_2 são independentes entre si, se e somente se:*

(i) Existe um intervalo de tempo de disparo comum entre t_1 e t_2 definido por $[max(TDI_D(t_1), TDI_D(t_2)), mim(TDF_D(t_1), TDF_D(t_2))]$ tal que θ pertence a este intervalo

(ii) O disparo de t_1 no estado s no instante de tempo θ leva ao estado s' ($s \xrightarrow{(t_1,\theta)} s'$), tal que t_2 é também disparável em s' ($t_2 \in FT(s')$; e

(iii) Existe um único estado s', tal que o disparo de t_1 e t_2 no instante de tempo θ leva ao estado s' ($s \xrightarrow{(t_1,\theta);(t_2,\theta)} s'$ e $s \xrightarrow{(t_2,\theta);(t_1,\theta)} s'$).

Segundo esta definição, duas transições são independentes se não estiverem em conflito dinâmico, ou seja, o disparo de uma delas não desabilita a outra (condição dependente

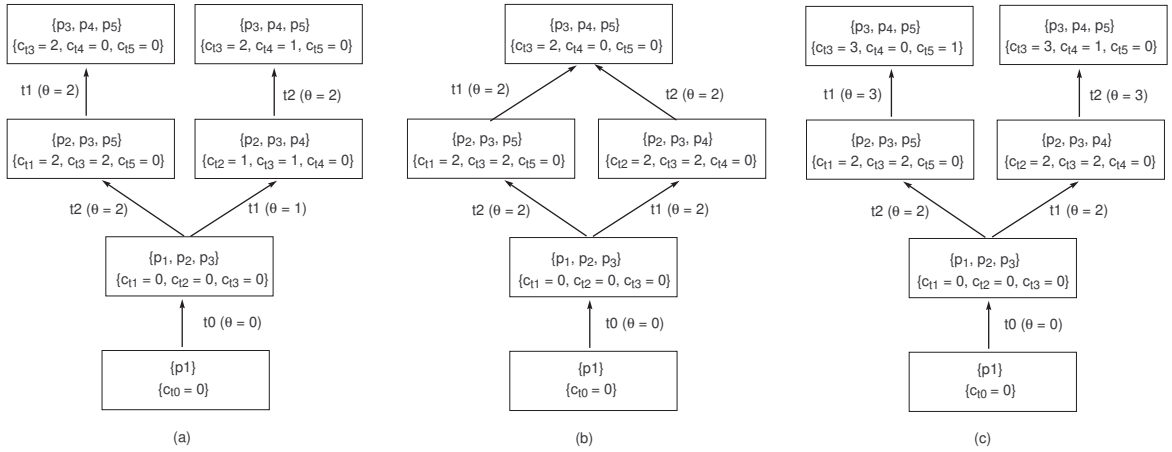

Figura 3.2: Grafo de Alcançabilidade - Disparo de t_1 e t_2: (a) Fora do Intervalo de Disparo comum; (b) Dentro do Intervalo de Disparo Comum e no Mesmo Instante; (c) Dentro do Intervalo de Disparo Comum e em Instantes Distintos

somente da marcação) e se, independentemente da ordem em que forem disparadas, conduzirem ao mesmo estado (condição dependente da marcação e dos relógios). Desse modo, a definição estabelece a necessidade da existência de um intervalo de disparo comum entre as duas transições, e delimita o disparo de ambas a um mesmo instante no tempo dentro desse intervalo.

A Figura 3.2 ilustra parte do grafo de alcançabilidade para a rede apresentada na Figura 3.1(a). Como t_1 e t_2 não estão em conflito dinâmico, o disparo destas duas transições, em qualquer ordem ou instante de tempo, leva sempre a mesma marcação (p_3,p_4,p_5). Entretanto, segundo a Definição 3.7, um estado é caracterizado não somente pela marcação, mas também pelo vetor de temporização das transições habilitadas nesse estado. Desse modo, como pode ser visto na Figura 3.2(a), o disparo das transições t_1 e t_2 fora de um intervalo de disparo comum (t_1 dispara no instante 1, quando t_2 ainda não está apta a disparar), conduz a dois estados distintos. O mesmo ocorre quando os disparos acontecem dentro do intervalo de disparo comum as duas transições ([2,3]) mas em momentos diferentes (Figura 3.2(c)). Quando, porém, o disparo ocorre dentro do intervalo comum de disparo e no mesmo instante no tempo (t_1 e t_2 disparam no instante 2) um único estado é alcançado (Figura 3.2(b))

Desse modo, na rede tomada como exemplo (Figura 3.1), t_1 e t_2 são exemplos de transições independentes no intervalo [2,3].

Definição 3.10 (Conjunto de Passos e Passo) *Sejam s um estado pertencente ao conjunto de estados de uma rede de Petri e $FT(s)$ o conjunto de transições disparáveis no estado s. O conjunto de todos os passos disparáveis em s ($ST(s)$) é um conjunto onde cada um de seus elementos (st_i) é um subconjunto de $FT(s)$ não vazio tal que: se $|st_i| > 1$, para qualquer par de transições distintas t_m e t_n pertencentes a st_i ($t_m, t_n \in st_i \mid m \neq n$), t_m e t_n são transições independentes.*

Por essa definição, um passo st_i é qualquer subconjunto de $FT(s)$ unitário ou composto somente de transições independentes entre si. Esta definição é equivalente à definição de passo simples (*simple step*) apresentada por Reising em [37], uma vez que não considera disparos simultâneos (multiplos disparos) de uma mesma transição. Desse modo, neste trabalho, as referências a passo estão sempre relacionadas a um conjunto e não a um multiconjunto de transições. Esta será a definição de passo assumida ao longo deste trabalho.

É fácil perceber que a cardinalidade de $ST(s)$ é, no mínimo, igual à cardinalidade de FT(s), caso em que não existem transições independentes em $FT(s)$, e somente uma seqüência de subconjuntos unitários é gerada.

Definição 3.11 (Domínio de Disparo de um Passo) *Sejam s um estado em uma rede de Petri com temporização, $FT(s)$ o conjunto das transições disparáveis em s, e st um passo qualquer pertencente a $ST(s)$. O domínio de disparo para st é definido pelo seguinte intervalo:*

$$FDS_s(st) = [\max(TDI_D(t_i)),\ \min(TDF_D(t_k))],\ \forall t_i \in st,\ t_k \in FT(s).$$

Definição 3.12 (Conjunto de Passos Unitários) *Sejam s um estado em uma rede de Petri com temporização, e $ST(s)$ o conjunto de todos os possíveis passos em s. O*

conjunto de passos unitários de s é definido por:

$$SST(s) = \{st_i \in ST(s) : |st_i| = 1,\ \forall st_i \in ST(s)\}.$$

O conjunto de passos unitários é um conjunto de passos onde todo elemento possui cardinalidade igual a 1 (são compostos por uma única transição).

Definição 3.13 (Conjunto de Passos Máximo) *Sejam s um estado em uma rede de Petri com temporização, e $ST(s)$ o conjunto de todos os possíveis passos em s. O conjunto de todos os passos máximo de s é definido por:*

$$MST(s) = \{st_i \in ST(s) \mid \nexists st_j \in ST(s)(st_i \subset st_j)\ \forall st_i, st_j \in ST(s)\}.$$

Ao contrário do que se possa inicialmente imaginar, o conjunto de passos máximo, não é exatamente o conjunto contendo todos os passos com maior cardinalidade (passos com maior número de transições disparáveis), e sim um conjunto contendo todos os passos aos quais não se pode acrescentar mais nenhuma transição.

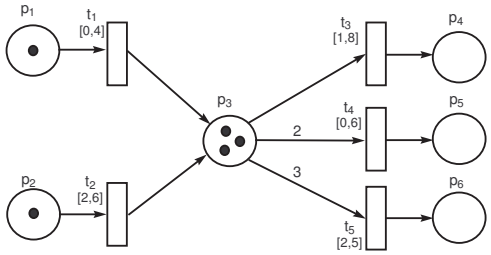

Figura 3.3: Rede de Petri com Conflitos Dinâmicos

No estado apresentado na rede da Figura 3.3 todas as transições estão habilitadas e são disparáveis. O domínio de disparo de qualquer passo desse estado tem como limite superior $TDF_D(t_1) = 4$ (Definição 3.11), que é o menor valor de TDF_D entre todas as transições disparáveis. Este limite é resultante da política de disparo obrigatório, que faz com que o disparo de t_1 tenha que ocorrer obrigatoriamente. Como este é o tempo limite

para que este disparo ocorra, e como, após esse disparo um novo estado é gerado, nenhum disparo em um instante de tempo maior que 4 faz sentido nesse estado.

Ainda nesse mesmo exemplo, o conjunto de todos os passos unitários é $SST = \{\{t_1\}, \{t_2\}, \{t_3\}, \{t_4\}, \{t_5\}\}$ (Definição 3.12). Como t_3 e t_4 estão em conflito dinâmico com t_5, o conjunto de todos passos máximo é $MST = \{\{t_1, t_2, t_3, t_4\}, \{t_1, t_2, t_5\}\}$, já que para estes dois passos, não existe nenhum outro passo no conjunto ST do qual estes sejam subconjuntos (Definição 3.13). Em outras palavras, estes são passos nos quais não é possível acrescentar mais nenhuma transição disparável. Inicialmente, tanto para o passo $\{t_1, t_2, t_3, t_4\}$, como para o passo $\{t_1, t_2, t_5\}$, o domínio de disparo é [2,4], que é um intervalo comum entre as transições nos respectivos passos.

A Figura 3.4 mostra o diagrama temporal das transições pertencentes ao passo máximo $\{t_1, t_2, t_3, t_4\}$, com destaque para o domínio de disparo.

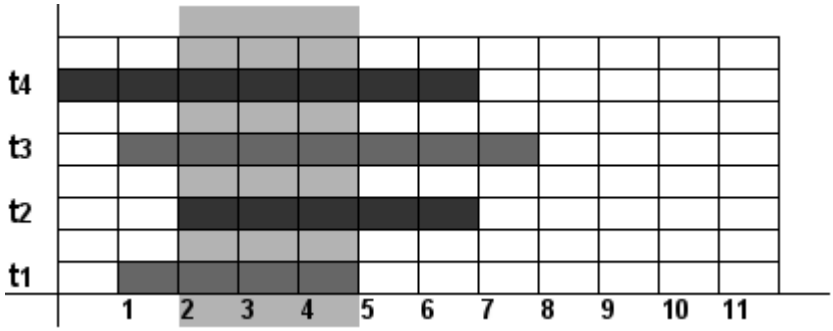

Figura 3.4: Diagrama Temporal para o Passo $\{t_1, t_2, t_3, t_4\}$

Definição 3.14 (Estados Alcançáveis) *Sejam P o conjunto de lugares de uma rede de Petri com temporização, $s_i = (m_i, c_i)$ um estado alcançável, st um passo ($st \in ST(s_i)$), e θ um instante no domínio de disparo do passo st ($\theta \in FDS_{s_i}(st)$). Um novo estado alcançável $s_j = \texttt{disparar}(s_i, (st, \theta))$ representa o disparo do passo st no instante θ a partir do estado s_i, de modo que um novo estado s_j é alcançado, tal que:*

- $\forall t_n \in st \ e \ \forall p \in P, \ m_j(p) = m_i(p) - W(p, t_n) + W(t_n, p)$;
- $\forall t_k \in ET(m_j), C_j(t_k) = \begin{cases} C_i(t_k) + \theta, & se \ (t_k \notin st) \wedge (t_k \in ET(m_j)) \wedge (t_k \in ET(m_i)) \\ 0, & em \ qualquer \ outro \ caso \end{cases}$

Definição 3.15 (Sistema de Transições Rotulado com Tempo) *Um sistema de transições rotulado com tempo é uma tupla* $\mathcal{L} = (S, \Sigma, \rightarrow, s_0)$*, onde* S *é um conjunto finito de estados discretos,* Σ *é um alfabeto de rótulo representando ações,* $\rightarrow \subseteq S \times \Sigma \times S$ *é a relação de transição, e* $s_0 \in S$ *é o estado inicial.*

Neste trabalho, a semântica de uma rede de Petri com temporização é definida através da associação desta com um sistema de transições rotulado com tempo $(TLTS)$[2]. O $TLTS$ é utilizado para representar uma escala viável, que é, na verdade, uma seqüência de disparos que garante que todas as restrições foram cumpridas.

Definição 3.16 (Escala Viável) *Sejam* $\mathcal{L}$ *um sistema de transições rotulado com tempo de uma rede de Petri com temporização,* s_0 *o estado inicial da rede,* $s_n = (m_n, c_n)$ *o estado final, onde* m_n *representa a marcação desejada.* $s_0 \xrightarrow{(st_{k1},\theta_{k1})} s_1 \xrightarrow{(st_{k2},\theta_{k2})} s_2 \rightarrow \ldots \rightarrow s_{n-1} \xrightarrow{(st_{kn},\theta_{kn})} s_n$ *é uma escala viável, onde* $s_{i+1} = \texttt{disparar}(s_i, (st_{ki}, \theta_{ki}))$*,* $i \geq 0$*,* $st_{ki} \in ST(s_i)$*, e* $\theta_{ki} \in FDS_{s_i}(st_{ki})$*.*

Definição 3.17 (Rede de Petri Rotulada com Código) *Uma rede de Petri rotulada com código (CTPN)*[3] *é representada por* $\mathcal{P}_c = (\mathcal{P}, \mathcal{C})$*, onde* $\mathcal{P}$ *é uma rede de Petri com temporização, e* $\mathcal{C}$*:*$T \nrightarrow \mathcal{SC}$ *é uma função parcial que associa código fonte (especificação comportamental) a transições, onde* $\mathcal{SC}$ *é um conjunto de códigos fonte.*

O fato de $\mathcal{C}$ ser uma função parcial implica que algumas transições podem não possuir código associado.

3.2.2 Extensão para um Modelo com Restrições de Consumo de Energia

A seguir, é apresentado um conjunto de definições que estende algumas das definições anteriomente enunciadas, de modo que as questões relacionadas ao consumo de energia possam ser consideradas.

[2]Do inglês: *Timed Labeled Transition System*
[3]Do inglês: *Code-labeled Time Petri Net*

Definição 3.18 (CTPN com Prioridade e Consumo de Energia) *Uma rede de Petri rotulada com código, prioridade e consumo de energia (CTPNPE) é representada por* $\mathcal{P}_E = (\mathcal{P}_c, \pi, \mathcal{E})$*, onde* $\mathcal{P}_c$ *é uma CTPN,* $\pi : T \rightarrow \mathbb{N}$ *é uma função prioridade que associa uma prioridade a cada transição de CTPNPE, e* $\mathcal{E} : T \rightarrow \mathbb{R}^+$ *é uma função energia que associa a cada transição um valor de consumo de energia.*

Definição 3.19 (Estados Considerando Consumo de Energia) *Sejam* $\mathcal{P}_E$ *uma CTPNPE,* C *o conjunto de todos os vetores de temporização em* $\mathcal{P}_E$*,* M *o conjunto de marcações alcançáveis de* $\mathcal{P}_E$*. O conjunto de estados* $S_\mathcal{E}$ *de* $\mathcal{P}_E$ *é definido por* $S_\mathcal{E} \subseteq (M \times \mathbb{N}^{|ET(M)|} \times \mathbb{R}^+)$*. Desse modo, um único estado é definido pela tupla* (m_i, c_i, e_i)*, onde* m_i *é uma marcação,* c_i *é o vetor de temporização relativo a* $ET(m_i)$*, e* e_i *é o consumo de energia acumulado até este estado. O estado inicial de uma CTPNPE é identificado por* $s_0 = (m_0, c_0, e_0)$*, onde* $c_0(t) = 0,\ \forall t \in ET(m_0),$ *e* $e_0 = 0$*.*

Definição 3.20 (Conjunto de Transições Disparáveis Considerando Consumo de Energia *Sejam* $\mathcal{P}_E$ *uma CTPNPE,* $s = (m, c, e)$ *um estado de* $\mathcal{P}_E$ *e* e_{max} *a restrição de máximo consumo de energia.* $FT(s, e_{max})$ *é o conjunto de transições disparáveis considerando consumo de energia no estado* s *e é definido por:*

$$FT(s, e_{max}) = \{t_i \in ET(m) \mid (\mathcal{E}(t_i) + e \leq e_{max}) \wedge (TDI_D(t_i) \leq \min(TDF_D(t_k))) \forall t_k \in ET(m)\}.$$

Esta definição acrescenta restrições de consumo de energia ($\mathcal{E}(t_i) + e \leq e_{max}$) às restrições temporais ($TDI_D(t_i) \leq \min(TDF_D(t_k))$) apresentadas anteriormente (Definição3.8).

Definição 3.21 (Estados Alcançáveis com Consumo de Energia) *Sejam* $\mathcal{P}_E$ *uma CTPNPE,* $s_i = (m_i, c_i, e_i)$ *um estado alcançável, st um passo (*$st \in ST(s_i)$*),* $\mathcal{E}_s(st)$ *o consumo de energia relacionado com o disparo do passo st (*$\mathcal{E}_s(st) = \sum_{t \in st} \mathcal{E}(t)$*), e* θ *um instante no intervalo de disparo de st (*$\theta \in FDS_{s_i}(st)$*).* $s_j =$ ***disparar***$(s_i, (st, \theta))$ *representa o disparo do passo st no tempo* θ *a partir do estado* s_i*, de modo que um novo estado* s_j *é alcançado, tal que:*

- $\forall t_n \in st \ e \ \forall p \in P, \ m_j(p) = m_i(p) - W(p, t_n) + W(t_n, p)$;
- $e_j = e_i + \mathcal{E}_s(st)$;
- $\forall t_k \in ET(m_j), C_j(t_k) = \begin{cases} C_i(t_k) + \theta, & se \ ((t_k \notin st) \wedge (t_k \in ET(m_j) \wedge (t_k \in ET(m_i) \\ 0, & em \ qualquer \ outro \ caso \end{cases}$

Esta definição estende a Definição 3.14, acrescentando a cada novo estado gerado, a informação relacionada ao consumo de energia acumulado ($e_j = e_i + \mathcal{E}_s(st)$).

3.3 Construção do Modelo

Baseado nos fundamentos computacionais apresentados anteriormente (Seção 3.2), esta seção expõe o processo de construção de um modelo capaz de capturar e representar as características de um sistema embarcado de tempo real crítico, de acordo com os elementos contemplados no padrão de especificação apresentados no início deste capítulo (Seção 3.1).

A estratégia utilizada neste trabalho para a construção de um modelo está baseada na definição de estruturas básicas (blocos básicos representados por redes de Petri) e na utilização dessas estruturas na composição de estruturas mais elaboradas. Assim, cada bloco captura parte dos elementos propostos na especificação, e o arranjo de uma coleção de blocos é capaz de modelar o sistema completo.

De acordo com essa estratégia, esta seção apresenta, inicialmente, um conjunto de regras de composição utilizado para integração dos blocos (manipulação de redes de Petri). O tópico seguinte discute a questão relacionada à definição do período a ser modelado. Em seguida, são apresentados os blocos básicos que compõem o modelo, bem como os passos de integração desses blocos na representação do sistema. Por fim, um bloco para modelagem de tarefas com sobrecarga do despachante é apresentado.

3.3.1 Regras de Composição

Essa seção apresenta a definição dos operadores que representam as regras de composição utilizadas neste trabalho. Os operadores apresentados são: mesclagem de lugar, refina-

mento de lugar, adição de arco, remoção de arco, adição de lugar e união de redes. Uma discussão mas detalhada da interpretação e uso destes operadores pode ser obtida em [6].

Definição 3.22 (Mesclagem de lugar) *Sejam N_1, N_2 e N_c redes de Petri com temporização, definidas por:*

$N_1 = (P_1, T_1, F_1, W_1, M_{01}, I_1)$;

$N_2 = (P_2, T_2, F_2, W_2, M_{02}, I_2)$;

$N_c = (P_c, T_c, F_c, W_c, M_{0c}, I_c)$,

onde:

$P_1 = \{p_{1_1}, p_{1_2}, \ldots, p_{1_{n_1}}\}$; $\quad T_1 = \{t_{1_1}, t_{1_2}, \ldots, t_{1_{m_1}}\}$

$P_2 = \{p_{2_1}, p_{2_2}, \ldots, p_{2_{n_2}}\}$; $\quad T_2 = \{t_{2_1}, t_{2_2}, \ldots, t_{2_{m_2}}\}$

$P_c = \{p_{c_1}, p_{c_2}, \ldots, p_{c_{n_3}}\}$; $\quad T_c = \{t_{c_1}, t_{c_2}, \ldots, t_{c_{m_3}}\}$.

Sejam, ainda, δ_1, δ_2 e δ_3 três conjuntos de lugares ordenados.

$\delta_1 = \langle p_1^1, p_1^2, \ldots p_1^i, \ldots, p_1^u \rangle \subseteq P_1$

$\delta_2 = \langle p_2^1, p_2^2, \ldots p_2^i, \ldots, p_2^u \rangle \subseteq P_2$

$\delta_c = \langle p_c^1, p_c^2, \ldots p_c^i, \ldots, p_c^u \rangle \subseteq P_c$.

A composição por mesclagem de lugar é representada por $N_c = \langle$`Pmerg`$\rangle$ (N_1, N_2, δ_1, δ_2, δ_c), onde N_1 e N_2 representam as redes a serem mescladas, δ_1 e δ_2 representam os conjuntos de lugares de N_1 e N_2 a serem mesclados, e δ_c o conjunto de lugares de N_c resultante da mesclagem. A rede N_c é composta do seguinte modo:

- $P_c = (P_1 \cup P_2 \cup \delta_c) - (\delta_1 \cup \delta_2)$.
- $T_c = T_1 \cup T_2$
- $\forall t \in T_c : \; I_c(t) = \begin{cases} I_1(t), & if\ t \in T_1 \\ I_2(t), & if\ t \in T_2 \end{cases}$
- $F_c = \{F_1 - (\delta_1 \times T_1 \cup T_1 \times \delta_1)\} \; \cup$

  $\{F_2 - (\delta_2 \times T_2 \cup T_2 \times \delta_2)\} \; \cup$

  $\{(p_c^i, t_{1_j}) \mid t_{1_j} \in T_1 \; \wedge \; (p_1^i, t_{1_j}) \in F_1, \; 1 \leq i \leq u, \; 1 \leq j \leq m_1\} \; \cup$

$$\{(t_{1_j}, p_c^i) \mid t_{1_j} \in T_1 \ \wedge \ (t_{1_j}, p_1^i) \in F_1,\ 1 \leq i \leq u,\ 1 \leq j \leq m_1\} \ \cup$$
$$\{(p_c^i, t_{2_k}) \mid t_{2_k} \in T_2 \ \wedge \ (p_2^i, t_{2_k}) \in F_2,\ 1 \leq i \leq u,\ 1 \leq k \leq m_2\} \ \cup$$
$$\{(t_{2_k}, p_c^i) \mid t_{2_k} \in T_2 \ \wedge (t_{2_k}, p_2^i) \in F_2,\ 1 \leq i \leq u,\ 1 \leq k \leq m_2\}$$

- $\forall p \in P_c:\ M_{0c}(p) = \begin{cases} M_{01}(p) & if\ p \in P_1, P_c \\ M_{02}(p) & if\ p \in P_2, P_c \\ \max\left(M_{01}(p_1^i), M_{02}(p_2^i)\right) & if \exists\ p_c^i = p,\ 1 \leq i \leq u. \end{cases}$

- $\forall f \in F_c : W_c(f) = \begin{cases} W_1(f) & if\ f \in F_1 \\ W_2(f) & if\ f \in F_2 \\ W_1(p_1^i, t_{1_j}) & if\ f = (p_c^i, t_{1_j}),\ p_c^i \in \delta_c,\ t_{1_j} \in T_1 \\ W_1(t_{1_j}, p_1^i) & if\ f = (t_{1_j}, p_c^i),\ p_c^i \in \delta_c,\ t_{1_j} \in T_1 \\ W_2(p_2^i, t_{2_k}) & if\ f = (p_c^i, t_{2_k}),\ p_c^i \in \delta_c,\ t_{2_k} \in T_2 \\ W_2(t_{2_k}, p_2^i) & if\ f = (t_{2_k}, p_c^i),\ p_c^i \in \delta_c,\ t_{2_k} \in T_2 \\ \multicolumn{2}{l}{tal\ que,\ 1 \leq i \leq u,\ 1 \leq j \leq m_1,\ 1 \leq k \leq m_2.} \end{cases}$

Este operador combina duas redes mesclando dois conjuntos de lugares (transformando dois conjuntos de lugares em um terceiro conjunto), resultando em uma única rede.

Definição 3.23 (Refinamento de Lugar) *Sejam N e N_c redes de Petri com temporização, representadas por $N = (P, T, F, W, M_0, I)$, e $N_c = (P_c, T_c, F_c, W_c, M_{0_c}, I_c)$. O refinamento de lugar é representado por $N_c = \langle \texttt{Pref} \rangle\,(N, p_\delta, p_\sigma, t_\sigma, p'_\delta)$, onde $p_\delta \in P$. A rede N_c é composta do seguinte modo:*

- $P_c = (P \cup \{p_\sigma, p'_\delta\}) - \{p_\delta\}$
- $T_c = T \cup \{t_\sigma\}$
- $F_c = (F - F^1) \cup F^2 \cup F^3$, *onde:*
  - $F^1 = \{(t_i, p_\delta), (p_\delta, t_j) \mid t_i \in {\bullet}p_\delta,\ t_j \in p_\delta{\bullet}\}$

– $F^2 = \{(t_i, p'_\delta), (p'_\delta, t_j) \mid t_i \in \bullet p_\delta,\ t_j \in p_\delta \bullet\}$

– $F^3 = \{(p_\sigma, t_\sigma), (t_\sigma, p'_\delta)\}$

- $\forall p \in P_c$: $M_{0_c}(p) = \begin{cases} M_0(p_j), & \textit{if } p = p_j \wedge p_j \in P \\ 0, & \textit{if } p = p_\sigma \vee p = p'_\delta \end{cases}$

- $\forall t \in T_c$: $I_c(t) = \begin{cases} I(t_j), & \textit{if } t = t_j \wedge t_j \in T \\ [0,0], & \textit{if } t = t_\sigma \end{cases}$

- $\forall f \in F_c$: $W_c(f) = \begin{cases} 1, & \textit{if } f = (g, p_\sigma),\ \forall g \in T \\ 1, & \textit{if } f = (p_\sigma, t_\sigma) \\ W(\bullet p_\delta, p_\delta), & \textit{if } f = (t_\sigma, p'_\delta) \\ W(p_\delta, \bullet p_\delta), & \textit{if } f = (p'_\delta, p_\delta \bullet) \\ W(f), & \textit{outros casos} \end{cases}$

Este operador substitui um único lugar por uma seqüência de um lugar, uma transição e outro lugar. A Figura 3.5 apresenta o resultado final de um refinamento.

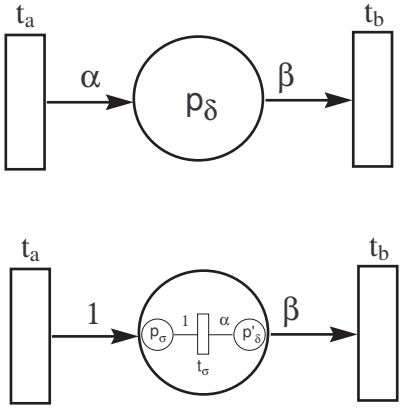

Figura 3.5: Refinamento Serial de Lugar

Definição 3.24 (Adição de Arco) *Sejam N e N_c redes de Petri com temporização, representadas por $N = (P, T, F, W, M_0, I)$ e $N_c = (P_c, T_c, F_c, W_c, M_{0_c}, I_c)$. A adição de arco é um operador que acrescenta a N um único arco (x, y), tal que, $(x \in P \wedge y \in$*

$T) \vee (x \in T \wedge y \in P)$. *Este operador é representado por* $N_c = \langle \texttt{Aadd} \rangle (N, (x, y), w)$, *onde* N *é a rede original,* (x, y) *é o arco,* w *é o peso do arco, e* N_c *é a rede resultante. A rede* N_c *é composta do seguinte modo:*

- $P_c = P; \quad T_c = T; \quad M_{0_c} = M_0; \quad I_c = I;$
- $F_c = (F \cup \{(x, y)\};$
- $\forall f \in F_c$: $W_c(f) = \begin{cases} W(f), & if\ f \in F \\ w, & if\ f = (x, y) \end{cases}$

Este operador adiciona um arco com peso w de um lugar para uma transição ou de uma transição para um lugar.

Definição 3.25 (Remoção de Arco) *Sejam* N *e* N_c *redes de Petri com temporização, representadas por* $N = (P, T, F, W, M_0, I)$, *e* $N_c = (P_c, T_c, F_c, W_c, M_{0_c}, I_c)$. *A remoção de arco é um operador que retira de* N *um único arco* (x, y). *Este operador é representado por* $N_c = \langle \texttt{Arem} \rangle (N, (x, y))$, *onde* N *é a rede original,* (x, y) *é o arco removido, e* N_c *é a rede resultante. A rede* N_c *é composta do seguinte modo:*

- $P_c = P; \quad T_c = T; \quad M_{0_c} = M_0; \quad I_c = I;$
- $F_c = (F - \{(x, y)\};$
- $\forall f \in F_c$: $W_c(f) = W(f)$

Definição 3.26 (Adição de Lugar) *Sejam* N *e* N_c *redes de Petri com temporização, representadas por* $N = (P, T, F, W, M_0, I)$, *e* $N_c = (P_c, T_c, F_c, W_c, M_{0_c}, I_c)$. *A adição de lugar é um operador que acrescenta a* N *um único lugar. Este operador é representado por* $N_c = \langle \texttt{Padd} \rangle (N, p_\delta, m_\delta)$, *onde* N *é a rede original,* p_δ *é um lugar,* m_δ *é sua respectiva marcação, e* N_c *e a rede resultante. A rede* N_c *é composta do seguinte modo:*

- $T_c = T; \quad F_c = F; \quad W_c = W; \quad I_c = I;$
- $P_c = P \cup \{p_\delta\}$

- $\forall p \in P_c,\ M_{0c}(p) = \begin{cases} M_0(p), & if\ p \in P \\ m_\delta, & if\ p = p_\delta \end{cases}$

Este operador adiciona um à rede um único lugar com uma marcação específica.

Definição 3.27 (União de Redes) *Sejam N_1, N_2 e N_c redes de Petri com temporização, representadas por $N_1 = (P_1,\ T_1,\ F_1,\ W_1,\ M_{0_1},\ I_1)$, $N_2 = (P_2,\ T_2,\ F_2,\ W_2,\ M_{0_2},\ I_2)$, e $N_c = (P_c,\ T_c,\ F_c,\ W_c,\ M_{0_c},\ I_c)$, tal que $P_1 \cap P_2 = \emptyset$ e $T_1 \cap T_2 = \emptyset$. A união de redes é um operador que unifica duas redes. Este operador é representado por $N_c = N_1 \sqcup N_2$. A rede resultante N_c é composta do seguinte modo:*

$$P_c = P_1 \cup P_2; \quad T_c = T_1 \cup T_2; \quad F_c = F_1 \cup F_2$$

$$\forall f \in F_c,\ W_c(f) = \begin{cases} W_1(f), & if\ f \in F_1 \\ W_2(f), & if\ f \in F_2 \end{cases}$$

$$\forall p \in P_c,\ M_{0c}(p) = \begin{cases} M_{01}(p), & if\ p \in P_1 \\ M_{02}(p), & if\ p \in P_2 \end{cases}$$

$$\forall t \in T_c,\ I_c(t) = \begin{cases} I_1(t), & if\ t \in T_1 \\ I_2(t), & if\ t \in T_2 \end{cases}$$

Este operador apenas junta duas redes criando uma nova.

3.3.2 Período

O modelo adotado neste trabalho não captura um tempo infinito de execução do sistema. Ao contrário disso, apenas um período representativo do tempo é considerado. Este período contempla pelo menos uma instância de execução de cada tarefa pertencente ao sistema, de modo que o tempo completo de execução do sistema (que é normalmente infinito) pode ser representado através de repetições do período modelado.

Este período finito é obtido através do mínimo múltiplo comum (MMC) entre os períodos de execução das tarefas que compõem o sistema. Este valor é denominado *período da escala*.

Dentro deste novo valor de tempo, uma ou mais instância de execução da mesma tarefa podem ocorrer. O número exato de *instância* de cada tarefa pode ser obtido por $\mathcal{N}(\tau_i) = P_S/p_i$, onde $\mathcal{N}(\tau_i)$ representa o número de instância de τ_i, P_S representa o período da escala e p_i representa o período da tarefa τ_i.

Como exemplo, a Tabela 3.2 representa as restrições temporais para duas tarefas, τ_1 e τ_2. Neste caso específico, o período de escala modelado é de $P_S = 24$ e o número de instância de τ_1 é $\mathcal{N}(\tau_1) = 3$ e para τ_2 é $\mathcal{N}(\tau_1) = 3$.

Tabela 3.2: Restrições Temporais para Duas Tarefas

task	ph	r	c	d	p
τ_1	0	0	2	7	8
τ_2	0	2	2	6	6

3.3.3 Blocos Básicos

A seguir são apresentados os blocos básicos utilizados na composição do modelo. Estes blocos são representados por redes de Petri rotulada com código, prioridade e consumo de energia - CTPNPE (Definição 3.18).

Nas redes apresentadas, os valores das restrições temporais e os pesos dos arcos são considerados em relação às definições de tarefa periódica ($\tau_i = (ph_i, r_i, c_i, d_i, p_i, e_i, proc_i)$ - Definição 3.1) e tarefa de comunicação ($\mu_m = (\tau_i, \tau_j, ct_m, ce_m\ bus_m)$ - Definição 3.2)

A semântica implícita na nomenclatura utilizada para identificadores dos lugares e das transições é a seguinte:

a) Identificadores de lugares iniciam com p;

b) Identificadores de transições iniciam com t.

c) Os índices (subscritos) para lugares e transições são acrônimos compostos com o seguinte significado:

- a: chegada;
- b: barramento;
- c: computação;
- $comm$: comunicação;
- d: *deadline*;
- dm: *deadline* perdido;
- end: final da rede;
- f: final da tarefa;
- g: concessão (*grant*)
- me: exclusão mutua;
- pc: interrupção de computação;
- ph: deslocamento;
- $proc$: processador;
- r: liberação;
- $rbuf$: repositório de recepção (*buffer*);
- s: inicio da tarefa;
- $start$: inicio da rede;
- $sbuf$: repositório de envio (*buffer*);
- w: espera;

Chegada de Tarefa

Este bloco (Figura 3.6) modela a chamada periódica para todas as instâncias de uma tarefa no período P_S. t_{a_i} modela a chamada periódica das instâncias restantes após a

primeira execução. O peso do arco (t_{ph_i}, p_{wa_i}) representa o número de instâncias restantes. A transição t_{ph_i} modela o deslocamento na execução da primeira instância.

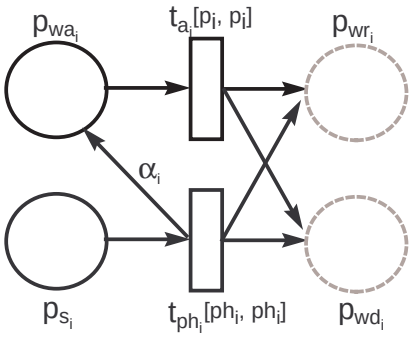

Figura 3.6: Bloco de Chegada de Tarefas

Instância de Tarefa

Este bloco modela a liberação, a concessão do processador e processamento (computação) da execução de uma instância de tarefa. Na verdade, para cada modelo de escalonamento (estático ou dinâmico), um bloco distinto é apresentado. Estes blocos são estruturalmente semelhantes, mas distintos em relação aos pesos dos arcos e atributos temporais e de energia.

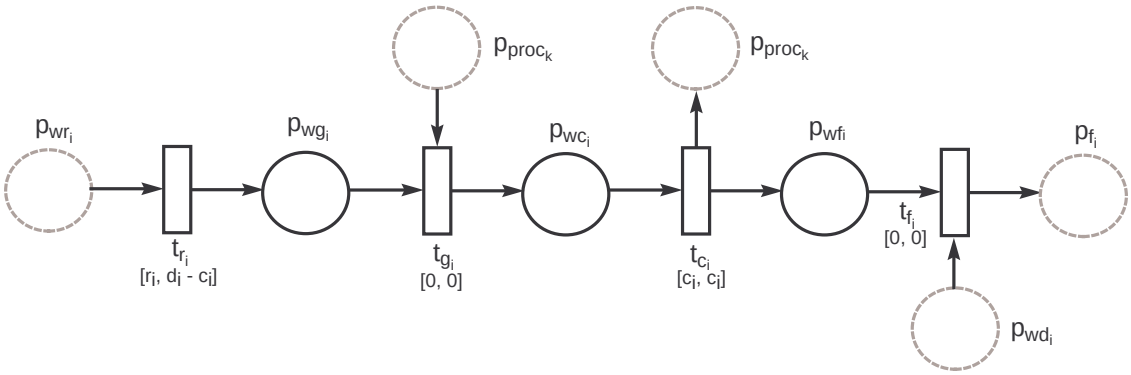

Figura 3.7: Bloco de Execução de Tarefa Não-Preemptiva

A Figura 3.7 apresenta o bloco para execução de tarefas não-preemptivas e a Figura 3.8 representa o bloco para execução de tarefas não preemptivas.

A transição t_{r_i} modela a liberação da instância de uma tarefa para execução. Seu intervalo de disparo é definido de acordo com as restrições temporais definidas para a tarefa. A transição t_{g_i} modela a concessão/reserva do processador e a transição t_{c_i} modela a computação e a posterior liberação do processador. Finalmente, a transição t_{f_i} modela a finalização da tarefa.

Para tarefas preemptivas (Figura 3.8), o ciclo computação (concessão do processador, computação e liberação do processador) é decomposto em várias unidades de tempo de tarefa (TTU), o que é representado através do intervalo de disparo transição de computação (t_{c_i}) e dos pesos dos arcos (t_{r_i},p_{wg_i}) e (p_{wf_i},t_{f_i}). Neste caso, a tarefa pode ser preemptada a cada ciclo de computação, e só será finalizada (t_{f_i}) quando todos os ciclos (c_i) de computação forem executados. O consumo de energia associado à transição de computação (t_{c_i}) é igual a e_i/c_i, que é o consumo de cada ciclo de computação.

Para as tarefas não-preemptivas (Figura 3.7), apenas um ciclo de computação é executado e consome todas as unidades de tempo de execução da tarefa de uma única vez. Em outras palavras, a tarefa não pode ser preemptada durante sua execução. Este comportamento é representado através do intervalo de disparo da transição de computação (t_{c_i}). Neste tipo de tarefa, o consumo de energia da tarefa (e_i) é diretamente associada à transição de computação.

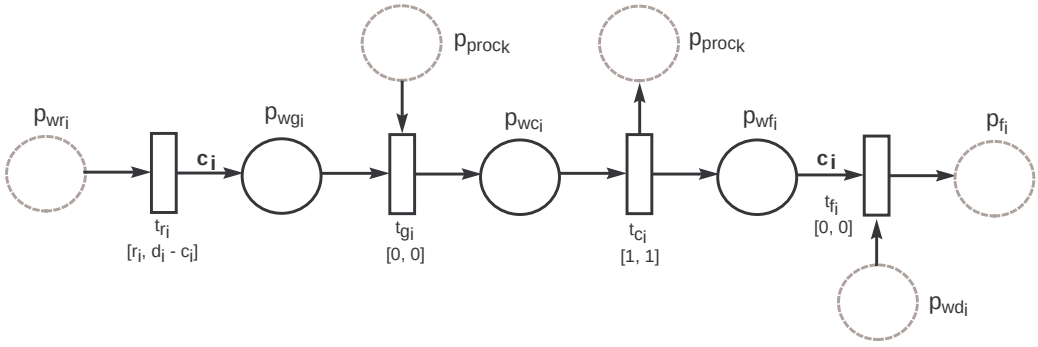

Figura 3.8: Bloco de Execução de Tarefa Preemptiva

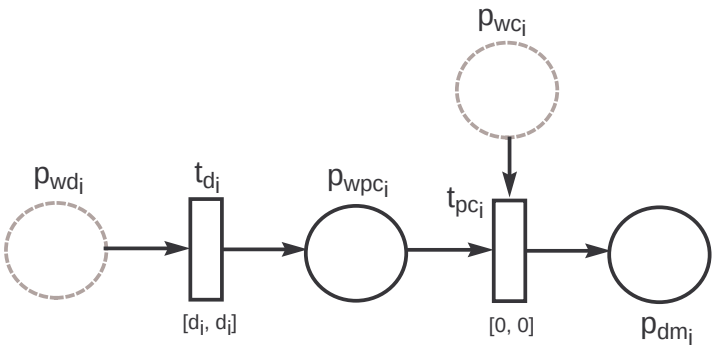

Figura 3.9: Bloco de Verificação de *Deadline*

Verificação de *Deadline*

Este bloco (Figura 3.9) modela a perda do tempo limite para execução de uma tarefa. A restrição temporal de *dealine* é representada no intervalo de disparo da transição t_{d_i}. Adicionalmente, a transição t_{pc_i} remove qualquer marca que possa habilitar a transição de computação.

Recurso

Os únicos recursos explicitamente modelados neste trabalho são os processadores e os barramentos de comunicação (Figura 3.10). Ambos são compostos apenas de um único lugar e, como descrito anteriormente, o relação entre tarefas e recursos (alocação) é efetuada em tempo de projeto.

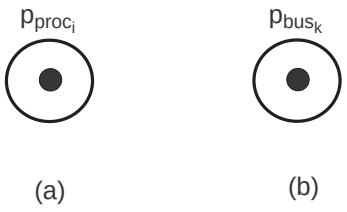

Figura 3.10: Bloco de Recursos: (a) Processador; (b) Barramento

Disjunção

Como apresentado anteriormente (Seção 2.4.3), o bloco de disjunção (Figura 3.11) modela a criação de n processos concorrentes, onde n representa a quantidade de tarefas existentes no sistema.

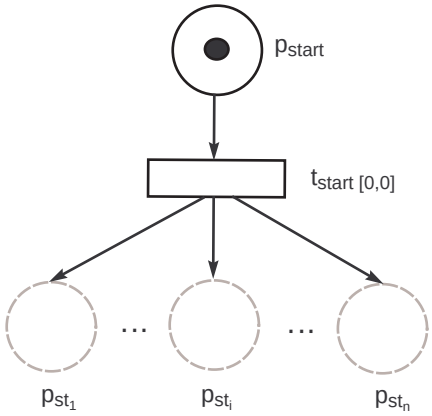

Figura 3.11: Bloco de Disjunção

Junção

Este bloco modela (Figura 3.12) a sincronização de todas (n) as tarefas ao final do período modelado (período da escala). Os pesos dos arcos de entrada de t_{end} modelam a necessidade da execução de todas as instância de cada tarefa do sistema. A marcação em p_{end} representa a marcação final do sistema, indicando que todas as instância das tarefas foram executadas obedecendo suas restrições.

Envio de Mensagem

Este bloco (Figura 3.13) modela o envio de uma mensagem entre tarefas alocadas em processadores distintos. A transição t_{gb_i} modela a alocação do barramento de comunicação para o envio da mensagem. A transição t_{send_i} modela o envio da mensagem para o *buffer* de comunicação. Após esse envio a tarefa emissora já pode retornar ao seu fluxo de execução (envio não bloqueante). Finalmente, a transição t_{comm_i} modela o envio da mensagem

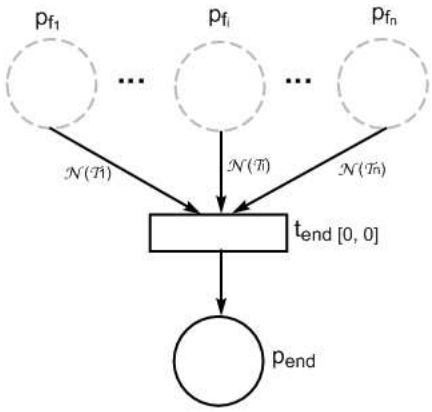

Figura 3.12: Bloco de Junção

através do barramento, com posterior liberação deste. As especificações de consumo de energia na comunicação e tempo de comunicação estão associadas à transição t_{comm_i}.

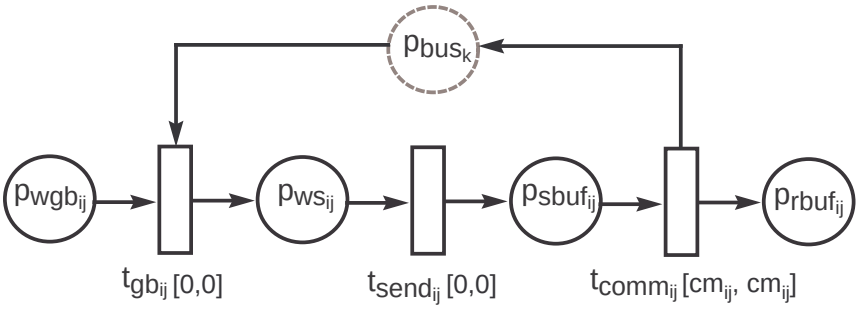

Figura 3.13: Bloco de Envio de Mensagem

3.3.4 Modelando Tarefas

A seguir, baseado nos bloco básico apresentados anteriormente, são apresentadas as composições para a representação da execução de uma tarefa e a composição para representação de um sistema mono processado com duas tarefas.

Modelo para uma Tarefa

O modelo para execução de uma tarefa é gerado através de uma composição envolvendo os seguintes blocos básicos: bloco de chegada de tarefa, bloco de instância de tarefa e bloco de verificação de *deadline*. Desse modo, um modelo para a tarefa T_0 da Tabela 3.3,

Tabela 3.3: Exemplo de Especificação com 3 Tarefas - Sem Consumo de Energia

Tarefa	ph	r	c	d	p	proc/bus
T0	0	0	10	100	250	proc1
T1	0	0	15	100	250	proc1
T2	0	0	20	150	250	proc1
Relações entre Tarefas						
T0 EXCLUDES T1, T0 PRECEDES T2						

instancia, inicialmente, os blocos N_{a_0}, N_{np_0} e N_{d_0}, representando os respectivos blocos básico.

Para que a composição possa ser efetuada, os seguintes conjuntos são considerados:

$\delta_1 = \langle p_{wr_0} \rangle \in P_{a_0}$ (de N_{a_0});

$\delta_2 = \langle p_{wr_0} \rangle \in P_{np_0}$ (de N_{np_0});

$\delta_{m_1} = \langle p_{wr_0} \rangle \in P_0$ (de N_0);

$\delta_3 = \langle p_{wc_0}, p_{wd_0} \rangle \in P_0$ (de N_0);

$\delta_4 = \langle p_{wc_0}, p_{wd_0} \rangle \in P_{d_0}$ (de N_{d_0});

$\delta_{m_2} = \langle p_{wc_0}, p_{wd_0} \rangle \in P_0$ (de N_0);

A rede $N_0 = (P_0, T_0, F_0, W_0, M_{0_0}, I_0)$, representando T_0, é então definida por:

a) $N_0 = \langle \texttt{Pmerg} \rangle (N_a, N_{np}, \delta_1, \delta_2, \delta_{m_1})$

b) $N_0 = \langle \texttt{Pmerg} \rangle (N_0, N_d, \delta_3, \delta_4, \delta_{m_2})$

A composição resultante, representando o modelo completo para a tarefa T_0, é apresentada na Figura 3.14.

Sistema Monoprocessado com Duas Tarefas

Um sistema composto por duas tarefas não preemtivas (T_0 e T_1 da Tabela 3.3) compartilhando um único processador é apresentado a seguir.

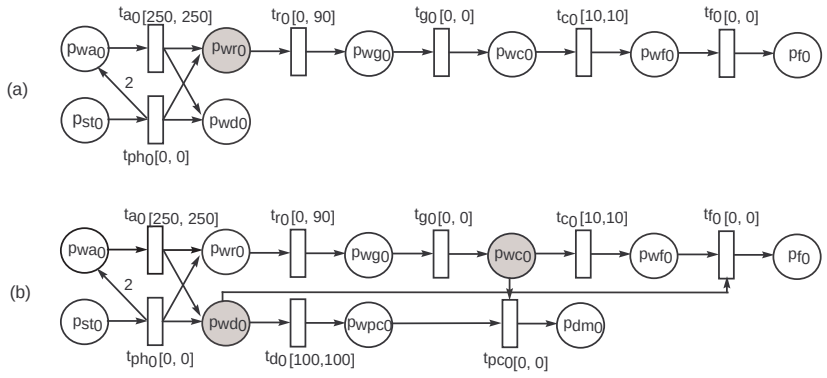

Figura 3.14: Modelo para Execução da Tarefa T_0

A rede representando o sistema é chamada $N_{aux} = (P_{aux}, T_{aux}, F_{aux}, W_{aux}, M_{0_aux}, I_{aux})$ e as duas tarefas são representadas por N_0 e N_1. Adicionalmente, também são instanciados os blocos de disjunção e de junção, representados por N'_f e N'_j, com o número de tarefas $n = 2$.

Para que a composição possa ser efetuada, os seguintes conjuntos são considerados:

$\delta_{st} = \langle p_{st_0}, p_{st_1} \rangle \in P_{aux}$;

$\delta_f = \langle p_{st_1}, p_{st_2} \rangle \in P_f$ (de N'_f);

$\delta_{m_1} = \langle p_{st_0}, p_{st_1} \rangle \in P_{aux}$;

$\delta_{end} = \langle p_{f_0}, p_{f_1} \rangle \in P_{aux}$;

$\delta_j = \langle p_{f_1}, p_{f_2} \rangle \in P_j$ (de N'_j);

$\delta_{m_2} = \langle p_{f_0}, p_{f_1} \rangle \in P_{aux}$;

p_{proc} é um único lugar, onde $M(p_{proc}) = 1$.

A rede representando o sistema (N_{aux}) é composta do seguinte modo:

a) $N_{aux} = (N_0 \sqcup N_1)$;

b) $N_{aux} = \langle \texttt{Padd} \rangle (N_{aux}, p_{proc}, 1)$

c) $N_{aux} = \langle \texttt{Aadd} \rangle (N_{aux}, (p_{proc}, t_{g0}), 1)$

d) $N_{aux} = \langle \texttt{Aadd} \rangle (N_{aux}, (p_{proc}, t_{g1}), 1)$

e) $N_{aux} = \langle \texttt{Aadd} \rangle (N_{aux}, (t_{c0}, p_{proc}), 1)$

f) $N_{aux} = \langle \texttt{Aadd} \rangle (N_{aux}, (t_{c1}, p_{proc}), 1)$

g) $N_{aux} = \langle \texttt{Pmerg} \rangle (N_{aux}, N'_f, \delta_{st}, \delta_f, \delta_{m_1})$

h) $N_{aux} = \langle \texttt{Pmerg} \rangle (N_{aux}, N'_j, \delta_{end}, \delta_j, \delta_{m_2})$

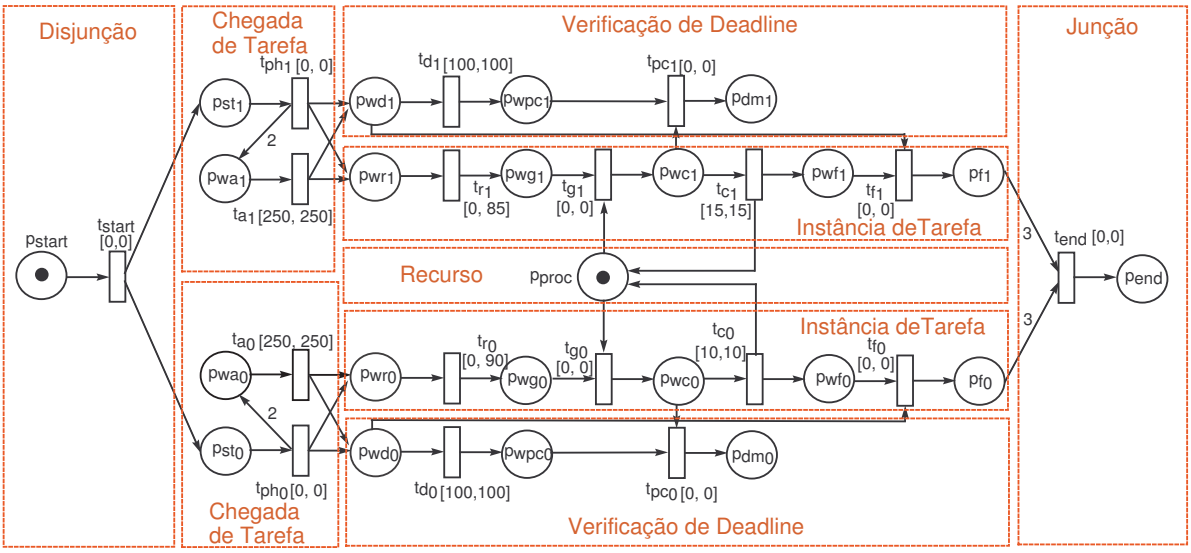

Figura 3.15: Modelo de um Sistema Monoprocessado com Duas Tarefas Não-preemptivas

A composição resultante, representando o modelo completo para a o sistema, é apresentada na Figura 3.15.

3.3.5 Modelando Relações entre Tarefas

A seguir são apresentadas as composições para as relações de precedência e exclusão.

Relação de Precedência

Como definido anteriormente, a relação de precedência é uma relação estabelecida entre duas tarefas, de modo que uma das tarefas só possa executar após a finalização da outra.

Desse modo, considerando a rede N_{aux} representando duas tarefas (τ_i e τ_j), a relação τ_i `PRECEDE` τ_j é modelada pelas seguintes operações.

a) $N_{aux} = \langle \texttt{Padd} \rangle (N_{aux}, p_{prec_{ij}}, 0)$

b) para a tarefa predecessora τ_i:

1. $N_{aux} = \langle \texttt{Aadd} \rangle (N_{aux}, t_{f_i}, p_{prec_{ij}})$

c) para a tarefa sucessora τ_j:

1. $N_{aux} = \langle \texttt{Pref} \rangle (N_{aux}, p_{wg_j}, p_{wp_{ij}}, t_{prec_{ij}}, p_{wg_j})$.
2. $N_{aux} = \langle \texttt{Aadd} \rangle (N_{aux}, t_{prec_{ij}}, p_{prec_{ij}}, 1)$

Considerando as tarefas T_1 e T_2 apresentadas na Tabela 3.3, a composição resultante, representando a precedência de T_1 sobre T_2 é apresentada na Figura 3.16.

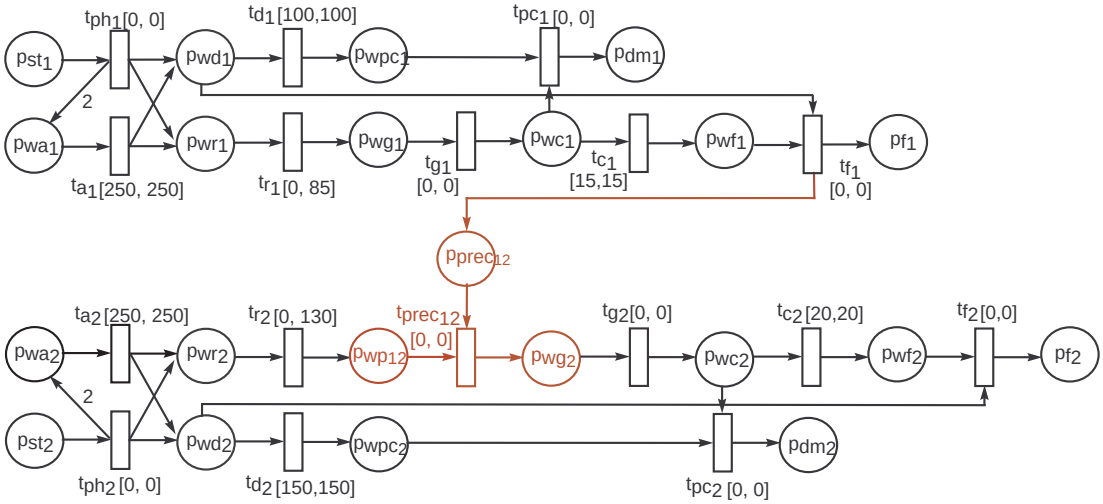

Figura 3.16: Modelo para Relação de Precedência para as Tarefas T_1 e T_2

Relação de Exclusão

Do mesmo modo que a relação de precedência, a relação de exclusão é definida entre um par de tarefas, de modo que estas tarefas não possam ser executadas concorrentemente. Assim, considerando a rede N_{aux} representando duas tarefas (τ_i e τ_j), a relação τ_i `EXCLUI` τ_j é modelada pelas seguintes operações.

a) $N_{aux} = \langle \texttt{Padd} \rangle (N_{aux}, p_{excl_{ij}}, 1)$;

b) para a tarefa τ_i:

1. $N_{aux} = \langle \texttt{Pref} \rangle (N_{aux}, p_{wg_i}, p_{wexcl_{ij}}, t_{excl_{ij}}, p_{wg_i})$;
2. $N_{aux} = \langle \texttt{Aadd} \rangle (N_{aux}, p_{excl_{ij}}, t_{excl_{ij}})$;
3. $N_{aux} = \langle \texttt{Aadd} \rangle (N_{aux}, t_{f_i}, p_{excl_{ij}})$;

c) para tarefa τ_j:

1. $N_{aux} = \langle \texttt{Pref} \rangle (N_{aux}, p_{wg_j}, p_{wexcl_{ji}}, t_{excl_{ji}}, p_{wg_j})$;
2. $N_{aux} = \langle \texttt{Aadd} \rangle (N_{aux}, p_{excl_{ij}}, t_{excl_{ji}})$;
3. $N_{aux} = \langle \texttt{Aadd} \rangle (N_{aux}, t_{f_j}, p_{excl_{ij}})$;

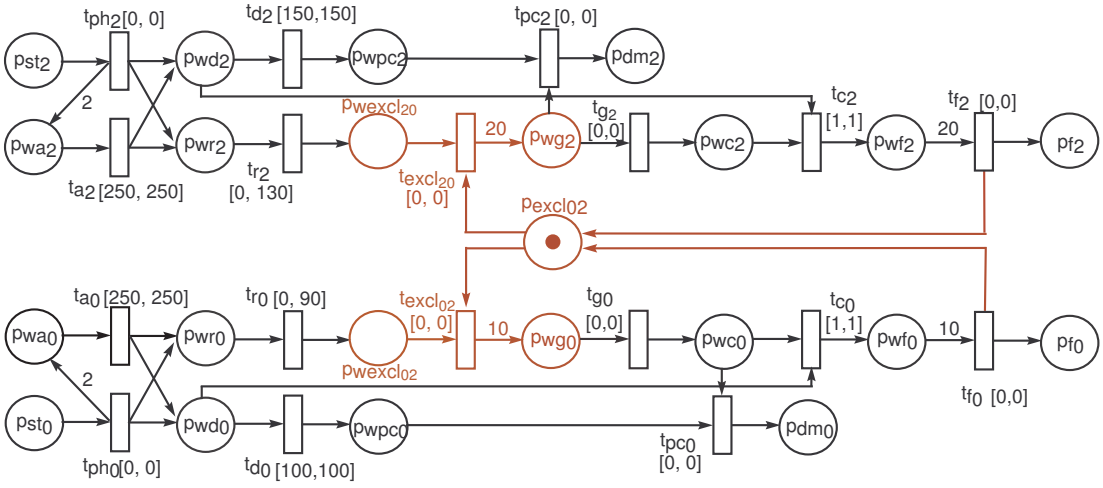

Figura 3.17: Modelo para Relação de Exclusão entre T_0 e T_2

Considerando as tarefas T_0 e T_2 apresentadas na Tabela 3.3, a composição resultante, representando a exclusão de T_1 sobre T_2 é apresentada na Figura 3.16.

3.3.6 Modelando Comunicação entre Tarefas

Esta trabalho considera a comunicação entre tarefas alocadas no mesmo processador como uma relação de precedência, uma vez que esta é normalmente realizada através de memória

compartilhada. Entretanto, a comunicação entre tarefas alocadas em processadores distintos é representada como uma nova tarefa de comunicação.

O modelo para comunicação entre duas tarefas é representado por uma composição que considera: (i) uma tarefa emissora e uma tarefa receptora, representadas pelas rede N_i e N_j, respectivamente; (ii) uma tarefa de comunicação representada pela rede $N_{sm_{ij}}$; e (iii) um barramento representado pelo lugar P_{bus_m}.

A rede N_{aux} representando a comunicação é composta do seguinte modo:

a) $(N_{aux} = ((N_i \sqcup N_j) \sqcup N_{sm_{ij}}))$

b) para a tarefa emissora (N_i):

   1. $t_{prev} = {\bullet}p_{f_i}$
   2. $(N_{aux} = \langle \texttt{Arem} \rangle (N_{aux}, t_{prev}, p_{f_i}))$;
   3. $(N_{aux} = \langle \texttt{Aadd} \rangle (N_{aux}, t_{prev}, p_{wgb_{ij}}))$.
   4. $(N_{aux} = \langle \texttt{Aadd} \rangle (N_{aux}, t_{send_{ij}}, p_{f_i}))$.
   5. $(N_{aux} = \langle \texttt{Aadd} \rangle (N_{aux}, (p_{bus_m}, t_{gb_{ij}}), 1))$;
   6. $(N_{aux} = \langle \texttt{Aadd} \rangle (N_{aux}, (t_{comm_{ij}}, p_{bus_m}), 1))$.

c) para a tarefa receptora (N_j):

   1. $(N_{aux} = \langle \texttt{Pref} \rangle (N_{aux}, p_{wc_j}, p_{recv_{ij}}, t_{recv_{ij}}, t_{wc_j}))$;
   2. $(N_{aux} = \langle \texttt{Aadd} \rangle (N_{aux}, p_{rbuf_{ij}}, t_{recv_{ij}}))$

A parte da composição relacionada ao envio da mensagem é apresentada nas Figuras 3.18. A Figura 3.19, apresenta os elementos envolvidos na recepção.

3.3.7 Modelando Sobrecarga do Despachante

Uma situação freqüentemente negligenciada na síntese de software é sobrecarga resultante da execução do despachante e do manipulador de interrupção de tempo. Uma solução,

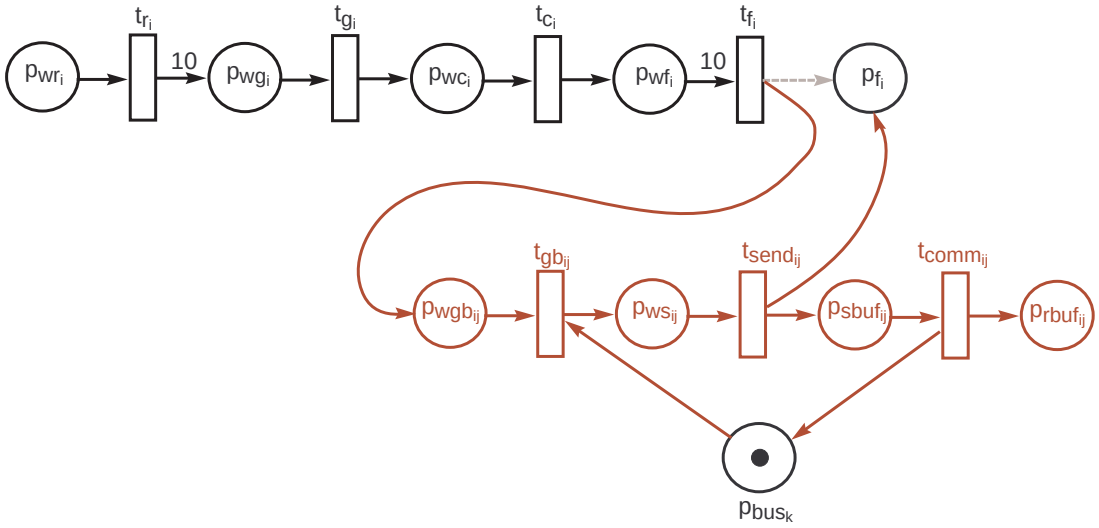

Figura 3.18: Modelo para Envio de Mensagem a partir de τ_i

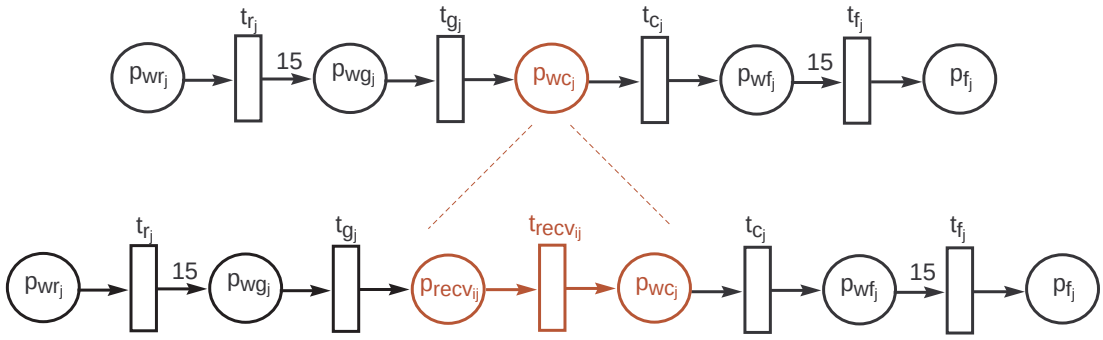

Figura 3.19: Modelo para Recepção de Mensagem em τ_i

adotada em vários trabalhos, como por exemplo [57], considera que esta sobrecarga já está incluída no pior tempo de execução da tarefa. No entanto, essa solução é bastante pessimista, uma vez que não é possível determinar quantas preempções ocorrerão antes de uma escala ser gerada. Nesse trabalho, a sobrecarga do despachante e do manipulador de interrupção é simplesmente chamada de *sobrecarga do despachante*.

Em adição aos blocos básicos propostos em [6] e apresentados anteriormente, este trabalho propõem um novo bloco para representação de tarefas considerando a sobrecarga do despachante. A sobrecarga é modelada pela adição de novos elementos (lugares e transições) ao bloco de instância de tarefas (preemptivas ou não-preemptivas). Esses novos elementos capturam as situações onde a chamada do despachante é obrigatória.

O bloco para instância de tarefas com sobrecarga do despachante é apresentado na Figura 3.20. Os pesos dos arcos e intervalos indicados nesta figura são relativos a uma tarefa preemptiva. Entretanto, da mesma forma que ocorre nos blocos originais, o modelo para tarefas não-preemptivas pode ser obtido apenas modificando os valores dos arcos (t_{r_i},p_{wg_i}) e (p_{wf_i},t_{f_i}), e do intervalo da transição de computação (t_{c_i}).

No contexto deste novo bloco, n representa o número de tarefas alocadas para o processador $proc_k$, não importando se estas são tarefas preemptivas ou não-preemptivas. A transição de sobrecarga (t_{o_i}) modela o consumo de tempo (através do intervalo [a,a]) e energia relacionada a sobrecarga do despachante. Os lugares $p_{proc_k}T_j$, $1 \leq j \leq n$, representam *memórias* que indicam que a tarefa T_j está em execução no processador $proc_k$. O lugar p_{proc_k-idle} modela o fato de o processador $proc_k$ estar ocioso.

O modelo proposto considera três situações a partir das quais o processador $proc_k$ pode ser alocado para tarefa T_i: (i) quando não há nenhuma tarefa utilizando o processador; (ii) quando a tarefa T_i já esta em execução no processador; e (iii) quando há uma outra tarefa diferente de T_i em execução no processador.

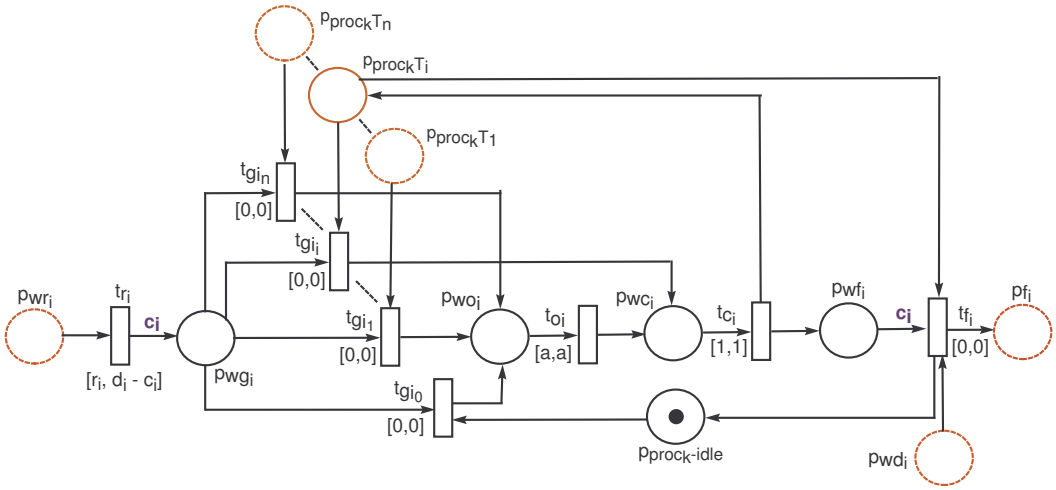

Figura 3.20: Bloco Básico para Tarefa com Sobrecarga do Despachante

A primeira situação é representada pela transição de concessão de processador $t_{g_{i_0}}$ e seu lugar de entrada p_{proc_k-idle}. Neste caso, como não há nenhuma tarefa em execução, o processador é alocado para a tarefa T_i e a sobrecarga do despachante é capturada na

transição de sobrecarga t_{o_i}.

A segunda situação é representada pela transição de concessão de processador $t_{g_{i_i}}$ e seu lugar de entrada $p_{proc_k}T_i$, que estando marcado, indica que a tarefa T_i já está em execução pelo processador. Nessa situação, o processador é reservado pela tarefa T_i, porém, a sobrecarga do despachante não é considerada, uma vez que esta situação representa apenas a continuação do processamento de T_i.

A terceira e última situação é representada pelas transições de concessão de processador $t_{g_{i_j}}$ e seus respectivos lugares de entrada $p_{proc_k}T_j, 1 \leq j \leq n, i \neq j$. Uma marca no lugar $p_{proc_k}T_j$ indica que a tarefa T_j está utilizando o processador e pode ser preemptada pela tarefa T_i. Nesta situação, caso a preempção de fato ocorra ($t_{g_{i_j}}$ dispare), a sobrecarga do despachante é capturada na transição seguinte (t_{o_i}).

As três situações analisadas podem ser generalizadas do seguinte modo: a sobrecarga do despachante precisa ser considerada sempre que o processador é alocado por uma tarefa diferente da tarefa atualmente em execução.

Finalizada a execução da tarefa T_i, a transição final (t_{f_i}) marca o processador novamente como ocioso e remove a marca em $p_{proc_k}T_i$

E importante obeservar que o modelo mantem a mesma estrutura (número de lugares e transições), indepententemente do tipo de tarefa modelado. Isso permite capturar situações onde tarefas preemptivas e não-preemptivas disputam o mesmo processador.

Como o objetivo de evitar trocas de contextos (preempções) desnecessárias, todas as transições $t_{g_{i_i}}, 1 \leq i \leq n$, têm maior prioridade que as demais transições de concessão de processador.

Por uma questão de clareza, a Figura 3.20 não mostra o processador representado pelo lugar p_{proc_k}, que é pré-condição para todas as tarefas de concessão de processador ($t_{g_{i_j}}$, $0 \leq j \leq n$) e pós-condição de tarefa de computação (t_{c_i}).

Capítulo 4

Síntese de Escalonamento Estático

Dadas as características altamente restritivas dos sistemas embarcados de tempo real crítico, a síntese de software para este tipo de sistema tem como objetivo a geração de um código fonte (programas) com o mínimo de sobrecarga possível e atendendo todas as restrições temporais. Um modo eficaz de atingir este objetivo é através da geração de código escalonado. Esta estratégia elimina a sobrecarga resultante da necessidade da execução de um escalonador dinâmico (escalonamento em tempo de execução) e aumenta significativamente a previsibilidade na execução do sistema, garantindo previamente, que todas as restrições serão atendidas.

A geração de código escalonado, pressupõe o conhecimento prévio da ordem em que as tarefas serão executadas (escala). Desse modo, este trabalho apresenta um método para síntese de escalonamento estático, baseado no modelo para sistema embarcados de tempo real apresentado no Capitulo 3. Este método consiste na busca por uma escala viável (Definição 3.16), sobre um sistema de transições rotuladas com tempo (Definição 3.15), gerado a partir da rede de Petri resultante do modelo para um sistema embarcado de tempo real crítico. Esta busca se baseia na *exploração do espaço de estados* em um sistema de estados finitos [16], que consiste na verificação recursiva de todos os estados sucessivos, gerados a partir do estado inicial, por meio do disparo de cada um dos passos possíveis em cada estado.

Duas abordagens são avaliadas para a geração do espaço de estados. Na primeira, um

novo estado é gerado a partir do disparo de uma única transição, entre as transições aptas a disparar, ou seja, os passos são compostos apenas por uma transição (*passos unitários*). Esta abordagem é conhecida como *semântica de transição* ou *interleaving*. A segunda abordagem, considera o disparo simultâneo do maior número possível de transições aptas a disparar, em outras palavras, todos os passo são *passos máximos*.

Esta seção inicia pela discussão de algumas questões relacionadas à síntese de escalonamento estático. Em seguida, é apresentada a abordagem para síntese de escalonamento utilizando *interleaving* (passos unitários) e posteriormente, a abordagem utilizando passos máximos

4.1 Questões Iniciais

Essa seção discute, brevemente, as questões relacionadas a (i) eliminação de estados indesejados durante a busca por uma escala (ii) e a aplicabilidade das estimativas de consumo de energia na síntese de escalonamento estático, sendo ambas, referenciadas durante o restante deste capítulo.

4.1.1 Eliminação de Estados Indesejados

O modelo proposto no Capítulo 3, explicitamente, captura situações não desejadas na execução de um sistema de tempo real crítico, como por exemplo, a perda de um *deadline*. Como resultado disso, o espaço de estados gerado na análise deste modelo, certamente inclui estados que representam situações indesejadas.

Como o método de escalonamento está interessado em uma escala que não leve a um estado indesejado, tais estados são simplesmente descartados durante a busca por uma escala viável. Este descarte é realizado eliminando-se do conjunto de transições disparáveis todas as transições que conduzam a algum estado desse tipo. No modelo proposto, as transições de *deadline* são exemplos de transições que levam a estados indesejado, uma vez que seu disparo indica que uma determinada restrição temporal não foi cumprida.

Logo, qualquer seqüência de disparo que envolva uma transição de *deadline* não pode ser considerada uma escala viável (Definição 3.16).

4.1.2 Escalonamento Estático e Consumo de Energia

A previsibilidade proporcionada pela adoção de escalonamento estático melhora a precisão da estimativa de consumo de energia de um sistema. A mesma exatidão não pode ser alcançada em sistemas que utilizem escalonamento dinâmico, uma vez que neste caso, a quantidade de preempções durante a execução do sistema depende da ordem imprevisível de chegada das tarefas. Tal aleatoriedade pode levar o escalonador a executar um número muito maior de preempções, na tentativa de cumprir todas as restrições temporais.

De fato, estimativas relacionadas a consumo de energia só fazem sentidos em um sistema com tarefas que suportem preempção. Se este não for o caso, é suficiente um simples cálculo baseado na quantidade de instâncias de cada tarefa executada no período modelado.

Mesmo contento otimizações que evitam troca de contexto desnecessária, os algoritmos propostos neste trabalho, não substituem outras técnicas de redução e controle de consumo de energia, como por exemplo graduação dinâmica de voltagem (*dynamic voltage scaling*) [33].

4.2 Abordagem Baseada em *Interleaving* - Passo Unitário

Na abordagem *interleaving*, cada disparo está associado a uma única transição o que permite que o conceito de *passo unitário*, anteriormente apresentado (Definição 3.12), possa ser diretamente utilizado. Assim, neste contexto, as referências ao *disparo de um passo unitário* devem ser compreendidas como o *disparo da transição pertencente ao passo unitário*.

Embora uma escala viável possa, certamente, ser encontrada por meio do disparo de uma transição a cada instante, a natureza combinatória dessa solução resulta em um crescimento exponencial no espaço de estados. Este crescimento exponencial é conhecido como *problema de explosão de estados* [16, 52] , e pode limitar a aplicação deste método.

Com o objetivo de minimizar este problema, Barreto [6] propõem um modo de reduzir o tamanho do espaço de estados gerado, aplicando uma técnica de redução baseada em ordem parcial e na relação de causa e efeito entre as transições do modelo. Como esta técnica está fundamentada na relação entre as transições, o método de redução proposto é específico para o modelo apresentado.

4.2.1 Redução por Ordem Parcial

A redução por ordem parcial está baseada na independência entre atividades, o que na prática significa que não importando a ordem em que estas atividades ocorram, sempre se alcançará o mesmo estado final.

Desse modo, a redução por ordem parcial consiste em considerar o disparo de apenas um subconjunto de todas as transições habilitadas no estado. Uma discussão detalhada sobre a aplicação de ordem parcial na análise de sistemas concorrentes pode ser encontrada em [16]. Lilius [24] apresenta uma semântica para a aplicação dessa técnica em redes de Petri com temporização.

Para que a redução por ordem parcial possa ser aplicada, é necessária a definição de quais transições serão consideradas para disparo e quais serão descartadas. A análise do conjunto de todas as transições pertencentes à rede de Petri gerada para o modelo proposto, conduz a percepção de que as transições de chegada são independentes umas das outras. A mesma conclusão é válida para as transições de liberação, precedência, computação, final e envio e recepção de mensagem. De fato, quando uma transição pertencente a uma dessas *classes de transição* é disparada, as demais permanecem habilitadas. O mesmo não acorre com as transições de exclusão e concessão de processador. Logo, para a definição do conjunto de transições persistentes, uma classificação baseada

na independência entre as transições e na relação de causa e efeito (quando o disparo de uma transição é obrigatório para o habilitação de uma outra transição) existente entre as *classes de transição* é considerada. Desse modo, uma *ordem de escolha* (CO) é atribuída a cada classe de transição segundo esses fatores.

Assim, somente os passos unitários compostos por transições pertencentes à classe com menor *ordem de escolha* (maior prioridade) necessitam ser analisada nas mudanças de estado. A Tabela 4.1 apresenta a *ordem de escolha* das classes de transições que compõem o modelo.

Tabela 4.1: Ordem de Escolha para Cada Classe de Transição

Ordem de Escolha	Tipo de Transição
1	Chegada
2	Liberação
3	Precedência
4	Computação
5	Envio de Mensagem
6	Recebimento de Mensagem
7	Final
8	Outras
9	Exclusão
10	Concessão de Processador
11	Concessão de Barramento

A definição formal para redução por ordem parcial é a seguinte:

Definição 4.1 (Redução por Ordem Parcial) *Seja $FT(s, e_{max})$ o conjunto das transições disparáveis no estado s. A* redução por ordem parcial *é um método de poda que gera o conjunto de transições persistentes $PT(FT(s, e_{max}))$ tal que:*

$$PT\ (FT(s, e_{max})) = \{t \in FT(s, e_{max}) \mid CO(t) = \min\left(CO(t_k)\right), \forall t_k \in FT(s, e_{max})\}$$

Como destacado anteriormente, este é um método específico para o modelo proposto (não genérico), que através da redução do espaço de estados, diminui o volume de pro-

cessamento e melhora o desempenho na busca por uma escala viável.

4.2.2 Heurísticas para Ordenação

Em adição aos métodos propostos para a redução do espaço de estado (remoção de estados indesejados e redução por ordem parcial), a busca por uma escala viável pode ainda ser aperfeiçoada por meio de heurísticas de ordenação que guiem a seqüência de disparo dos passos disponíveis em cada estado.

a) **Ordenação por prioridade**. Em um modelo que considere a sobrecarga do despachante, a ordenação por prioridade tem o objetivo de reduzir o consumo de energia do sistema, evitando trocas de contexto (preempções) desnecessárias. Quando o conjunto de transições disparáveis possui transições de concessão de processador, as transições responsáveis por manter a tarefa atual em processamento são privilegiados em relação às demais, uma vez que possuem maior prioridade (informações adicionais de como a seleção privilegiada de certas transições de concessão de processador afeta o chaveamento de contexto podem ser obtidos na Seção 3.3.7).

b) **Ordenação por Tempo Dinâmico de Disparo Inicial (TDI_D)**. A ordenação por TDI_D tem o objetivo de reduzir atrasos desnecessários no disparo de transições já habilitadas. Quando conjunto de transições disparáveis possui, por exemplo, duas transições de liberação t_{r_1} e t_{r_2} com os intervalos dinâmicos de disparos igual a [2,5] e [0,5] repectivamente. a transição t_{r_2} é previlegiada, uma vez que caso t_{r_1} seja primeiramente disparado, t_{r_2} permanecerá habilitada mas sofrerá um atraso desnecessário de duas unidades de tempo. Desse modo, transições com o mesmo valor de prioridade são ordenadas segundo seu TDI_D. Assim, sempre que possível, a seqüência de disparos produz uma evolução natural dos relógios das transições que permanecem habilitadas no estado seguinte.

c) **Ordenação por *deadline* mais eminente (EDF)**. A ordenação por EDF tem o objetivo de reduzir a ocorrência (e conseqüentemente a análise) de situações que

possam conduzir a perda de um *deadline*. A lógica por trás dessa heurística é bastante simples: tarefas com pouco tempo para finalizar sua execução, devem ser privilegiadas. Desse modo, caso duas transições possuam a mesma prioridade e o mesmo valor de TDI_D, a transição pertencente à tarefa com o *deadline* mais eminente é privilegiada.

Para produzir um conjunto de transições ordenadas, essas heurísticas são combinadas em um único método de ordenação, na seguinte ordem: prioridade, TDI_D e EDF.

É importante observar que as heurísticas de ordenação introduzidas nesta seção, não têm por objetivo contornar situações não capturadas no modelo proposto, mas somente guiar o método de busca com o objetivo de (i) melhorar o desempenho do algoritmo e (ii) obter uma escala com o mínimo consumo de energia.

A melhoria no desempenho é perseguida aumentando-se as chances de uma escala ser encontrada sem a necessidade de recuo (*backtracking*) em relação a um caminho tomado (ordenação por EDF e TDI_D), uma vez que a ocorrência de tal situação manifesta que estados desnecessários à escala final foram visitados. Uma escala com consumo mínimo de energia é perseguida pela redução do número de preempções (ordenação por prioridade).

4.2.3 Algoritmo para Síntese de Escalonamento Estático - *Interleaving*

O algoritmo apresentado nesta seção (Figura 4.1) é uma adaptação do algoritmo apresentado em [6]. Este algoritmo utiliza busca em profundidade (*depth-first*) para percorrer um sistema de transições rotuladas com tempo (TLTS), com o critério de parada determinado pelo alcance da marcação final desejada M^F (Definição 3.16). O espaço de estados é gerado sob demanda, de acordo com a evolução do algoritmo e o processamento sempre tem fim, uma vez que a rede de Petri resultante do modelo proposto é limitada e as restrições temporais são discretas, o que resulta em um TLTS finito.

Uma escala viável, satisfazendo as restrições temporais e de energia, é encontrada quando o algoritmo alcança a marcação final M^F (linha 2). De acordo com a Definição

```
1 Sintese-Interleaving(S, M^F, e_max)
2    Se (S.M = M^F) Retorne VERDADEIRO
3    Faça rotule S como estado já visitado
4    Faça PT := heuristicas-ordenação(ordem-parcial(estados-indesejados(disparaveis(S,e_max))))
5    Faça SST := Conjunto-Passos-Unitários(PT))
6    Se (|SST| = 0) Retorne FALSO
7    Para cada (st ∈ SST) Faça
8         Para cada (θ ∈ dominio-disparo-passo(st)) Faça
9              S' := disparar(S, st, θ)
10             Se (S' ainda não foi visitado) e (Sintese-Interleaving(S', M^F, e_max)) Então
11                  Faça inclua (S, S', st, θ) na TLTS
12                  Retorne VERDADEIRO
13             Fim
14        Fim
15   Fim
16   Retorne FALSO
17 Fim
```

Figura 4.1: Algoritmo para Síntese de Escalonamento Estático - *Interleaving*

3.20, a função *disparáveis* (linha 4), leva em consideração as restrições temporais e de energia. O conjunto de transições disparáveis é então modificada pela remoção das transições que levam a *estados indesejados* e pela redução por *ordem parcial* (também na linha 4), e em seguida é ordenado de acordo com as heurísticas apresentadas anteriormente (ainda na linha 4). Com base no conjunto ordenado de transições persistentes, o conjunto de passos unitários é computado (linha 5). Após isso, para cada instante no intervalo de disparo (linha 8) de cada um dos passos unitários (linha 7), a geração de um novo estado é considerada através da função *disparar* (linha 9). O esquema de identificação de estados já visitados assegura que nenhum estado será visitado mais de uma vez (linhas 3 e 10). A escala viável é representada através do sistema de transições rotuladas com tempo (linha 11) e o espaço de estados reduzido é completamente visitado somente quando o sistema não possui uma escala viável, situação, essa em que o algoritmo retorna FALSO (linha 16).

4.2.4 Aplicação do Algoritmo

O exemplo apresentado a seguir utiliza o modelo gerado para as tarefas da Tabela 3.2. A rede de Petri representando o modelo pode ser vista na Figura 4.2. Para que o processo de *backtracking* possa ser ilustrado, este exemplo não considera a sobrecarga do despachante e não guia a busca por nenhuma heurística de ordenação.

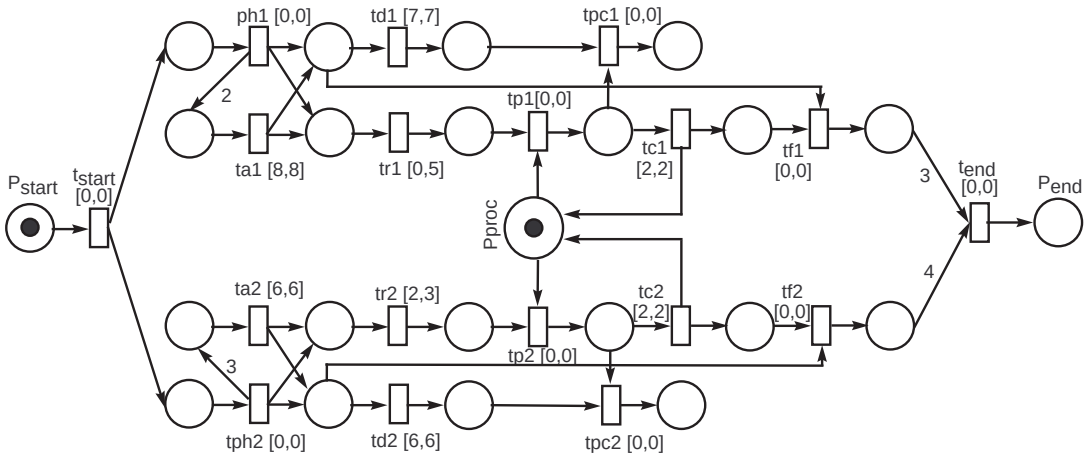

Figura 4.2: Rede para as Tarefas da Tabela 3.2

A Tabela 4.2 apresenta o número dos estados visitados, o conjunto de transições habilitadas, o conjunto de transições disparáveis, o conjunto de transições persistentes (pós-redução por ordem parcial), o passo disparado com o respectivo tempo de disparo e a energia acumulada em cada estado. O exemplo considera que ambas as tarefas consomem $2nJ$ de energia.

Como pode ser observado, no estado 15, duas transições de concessão de processador (tp_1 e tp_2) estão habilitadas. A possível execução da tarefa T_1 (escolha de tp_1) mostra-se uma escolha errada, uma vez que, após alguns minutos, T_2 perde seu deadline (estado 20). O algoritmo então recua (*backtrack*) para o estado 15 e tenta a segunda (e última) alternativa, isto é, reservar o processador para a tarefa T_2 (disparo de tp_2). Após o disparo de mais alguns passos, uma escala viável é encontrada, uma vez que no estado 36 a marcação final M^F é alcançada. Desse modo, uma escala viável é encontrada após a análise de 43 estados, quando o mínimo seriam somente 37. Se a heurística de ordenação

Tabela 4.2: Execução do Algoritmo - *Interleaving*

#	Estado	Habilitadas	Relógios	Disparáveis	Persistentes	Tempo,Passo	Energia
1	0	{tstart}	{0}	{tstart}	{tstart}	0,{tstart}	0
2	1	{tph1,tph2}	{0,0}	{tph1,tph2}	{tph1,tph2}	0,{tph1}	0
3	2	{tph2,tr1,ta1,td1}	{0,0,0,0}	{tph2,tr1}	{tph2}	0,{tph2}	0
4	3	{tr1,ta1,ta2,td1,td2}	{0,0,0,0,0}	{tr1}	{tr1}	0,{tr1}	0
5	4	{tp1,ta1,ta2,td1,td2}	{0,0,0,0,0}	{tp1}	{tp1}	0,{tp1}	0
6	5	{tr2,tc1,ta1,ta2,td1,td2}	{0,0,0,0,0,0}	{tr2,tc1}	{tr2}	2,{tr2}	0
7	6	{tc1,ta1,ta2,td1,td2}	{2,2,2,2,2}	{tc1}	{tc1}	0,{tc1}	2
8	7	{tf1,ta1,ta2,td1,td2}	{0,2,2,2,2,2}	{tf1}	{tf1}	0{tf1}	2
9	8	{tp2,ta1,ta2,td2}	{0,2,2,2}	{tp2}	{tp2}	0,{tp2}	2
10	9	{tc2,ta1,ta2,td2}	{0,2,2,2}	{tc2}	{tc2}	2,{tc2}	4
11	10	{tf2,ta1,ta2,td2}	{0,5,5,5}	{tf2}	{tf2}	0,{tf2}	4
12	11	{ta1,ta2}	{5,5}	{ta2}	{ta2}	2,{ta2}	4
13	12	{ta1,ta2,tr2,td2,}	{6,0,0,0}	{ta1,tr2}	{ta1}	2,{ta1}	4
14	13	{ta1,ta2,tr1,tr2,td1,td2}	{0,2,0,2,0,2}	{tr1,tr2}	{tr1,tr2}	0,{tr1}	4
15	14	{ta1,ta2,tr2,td1,td2,tp1}	{0,2,2,0,2,0}	{tr2,tp1}	{tr2}	0,{tr2}	4
16	**15**	**{ta1,ta2,td1,td2,tp1,tp2}**	**{0,2,0,2,0,0}**	**{tp1,tp2}**	**{tp1,tp2}**	**0,{tp1}**	4
17	16	{ta1,ta2,td1,td2,tc1}	{0,2,0,2,0}	{tc1}	{tc1}	2,{tc1}	6
18	17	{ta1,ta2,td1,td2,tf1}	{0,2,0,2,0}	{tf1}	{tf1}	0,{tf1}	6
19	18	{ta1,ta2,td2,tp2}	{2,4,4,0}	{tp2}	{tp2}	0,{tp2}	6
20	19	{ta1,ta2,td2,tc2}	{2,4,4,0}	{ta2,td2}	{ta2}	2,{ta2}	6
21	**20**	**{ta1,ta2,td2,tr2}**	**{4,0,6,2}**	**{td2}**	**{td2}**	**0,{td2}**	6
22	**15**	**{ta1,ta2,td1,td2,tp1,tp2}**	**{0,2,0,2,0,0}**	**{tp1,tp2}**	**{tp2}**	**0,{tp2}**	4
23	16	{ta1,ta2,td1,td2,tc2}	{0,2,0,2,0}	{tc2}	{tc2}	2,{tc2}	6
24	17	{ta1,ta2,td1,td2,tf2}	{0,2,0,2,0}	{tf2}	{tf2}	0,{tf2}	6
25	18	{ta1,ta2,td1,tp1}	{3,5,3,0}	{tp1}	{tp1}	0,{tp1}	6
26	19	{ta1,ta2,td1,tc1}	{3,5,3,0}	{ta2}	{ta2}	2,{ta2}	6
27	20	{ta1,ta2,td1,tc1,tr2}	{4,0,4,1,0}	{tc1}	{tc1}	1,{tc1}	8
28	21	{ta1,ta2,td1,tr2,tf1}	{4,0,4,0,0}	{tf1}	{tf1}	0,{tf1}	8
29	22	{ta1,ta2,tr2}	{4,1,1}	{tr2}	{tr2}	1,{tr2}	8
30	23	{ta1,ta2,tp2}	{5,2,0}	{tp2}	{tp2}	0,{tp2}	8
31	24	{ta1,ta2,tc2}	{5,2,0}	{tc2}	{tc2}	2,{tc2}	10
32	25	{ta1,ta2,tf2}	{5,2,0}	{tf2}	{tf2}	0,{tf2}	10
33	26	{ta1,ta2}	{8,5}	{ta1}	{ta1}	1,{ta1}	10
34	27	{ta2,td1,tr1}	{5,0,0}	{tr1}	{tr1}	0,{tr1}	10
35	28	{ta2,td1,tp1}	{5,0,0}	{tp1}	{tp1}	0,{tp1}	10
36	29	{ta2,td1,tc1}	{5,0,0}	{ta2}	{ta2}	1,{ta2}	10
37	30	{td1,tc1,td2,tr2}	{1,1,0,0}	{tc1}	{tc1}	1,{tc1}	12
38	31	{tf1,td1,td2,tr2}	{0,1,0,0}	{tf1}	{tf1}	0,{tf1}	12
39	32	{td2,tr2}	{1,1}	{tr2}	{tr2}	1,{tr2}	12
40	33	{td2,tp2}	{2,0}	{tp2}	{tp2}	0,{tp2}	12
41	34	{td2,tc2}	{2,0}	{tc2}	{tc2}	2,{tc2}	14
42	35	{td2,tf2}	{2,0}	{tf2}	{tf2}	0,{tf2}	14
43	**36**	**{tend}**	**{0}**	**{tend}**	**{tend}**	**0,{tend}**	14

fosse considerada neste exemplo, a primeira escolha do estado 15 seria a transição tp_2, e nenhum estado desnecessário seria visitado.

A escala viável encontrada pode ser expressa pela seguinte TLTS (Definition 3.16):

$$s_0 \xrightarrow{(t_{start},0)} s_1 \xrightarrow{(t_{ph1},0)} s_2 \xrightarrow{(t_{ph2},0)} s_3 \xrightarrow{(t_{r1},0)} s_4 \xrightarrow{(t_{p1},0)} s_5 \xrightarrow{(t_{r2},2)} s_6 \xrightarrow{(t_{c1},0)} s_7 \xrightarrow{(t_{f1},0)} s_8 \xrightarrow{(t_{p2},0)} s_9 \xrightarrow{(t_{c2},2)}$$
$$s_{10} \xrightarrow{(t_{f2},0)} s_{11} \xrightarrow{(t_{a2},2)} s_{12} \xrightarrow{(t_{a1},2)} s_{13} \xrightarrow{(t_{r1},0)} s_{14} \xrightarrow{(t_{r2},0)} s_{15} \xrightarrow{(t_{p2},0)} s_{16} \xrightarrow{(t_{c2},2)} s_{17} \xrightarrow{(t_{f2},0)} s_{18} \xrightarrow{(t_{p1},0)} s_{19} \xrightarrow{(t_{a2},2)}$$
$$s_{20} \xrightarrow{(t_{c1},1)} s_{21} \xrightarrow{(t_{f1},0)} s_{22} \xrightarrow{(t_{r2},1)} s_{23} \xrightarrow{(t_{p2},0)} s_{24} \xrightarrow{(t_{c2},2)} s_{25} \xrightarrow{(t_{f2},0)} s_{26} \xrightarrow{(t_{a1},1)} s_{27} \xrightarrow{(t_{r1},0)} s_{28} \xrightarrow{(t_{p1},0)} s_{29} \xrightarrow{(t_{a2},1)}$$
$$s_{30} \xrightarrow{(t_{c1},1)} s_{31} \xrightarrow{(t_{f1},0)} s_{32} \xrightarrow{(t_{r2},1)} s_{33} \xrightarrow{(t_{p2},0)} s_{34} \xrightarrow{(t_{c2},2)} s_{35} \xrightarrow{(t_{f2},0)} s_{36} \xrightarrow{(t_{end},0)} s_{37}.$$

4.3 Abordagem Baseada em Passo Máximo

Na abordagem de passo máximo, cada passo é composto ou por um conjunto unitário ou por um conjunto de transições independentes, de modo que, em qualquer um dos casos, não se possa acrescentar mais nenhuma transição disparável a este conjunto (Definição 3.13).

Segundo esta definição, mais de um conjunto de transições pode ser caracterizado como passo máximo em relação ao conjunto de transições disparáveis. Desse modo, na abordagem de passo máximo as possibilidades de mudanças de estado em uma rede, estão condicionadas ao disparo de cada um dos passos máximos em um determinado intervalo de tempo.

Essa abordagem reduz significativamente o espaço de estado, limitando a mudança ao disparo simultâneos de um grupo de transições, o que impede o atraso de uma transição disparável em relação à outra. Dependendo do modelo representado na rede de Petri (especificamente, dos intervalos das transições), esta abordagem pode reduzir as chances de que uma escala viável seja encontrada.

Na prática, a definição de independência entre transições pertencentes a um estado (Definição 3.9), e conseqüentemente, a definição de passo máximo, apresentam duas questões a serem resolvidas: (i) a resolução de conflitos relacionados à marcação (conflitos dinâmicos) e a (ii) definição de um intervalo de disparo para o passo.

4.3.1 Resolução de Conflitos Dinâmicos

A primeira questão na composição do conjunto de passos máximo é a resolução de conflitos dinâmicos. Na rede de Petri sem temporização apresentada na Figura 4.3, as transições t_3 e t_4 encontram-se em conflito dinâmico em relação à t_5. Logo, o disparo de t_3 ou de t_4, automaticamente desabilita t_5, e desse modo, t_5 não pode estar presente em um passo máximo que contenha t_3 ou t_4.

Neste caso, o conjunto de todos os passo máximos para esta rede é $MST = \{\{t_0, t_1, t_2, t_3, t_4\}, \{t_0, t_1, t_2, t_5\}\}$. Assim, pode se dizer que no passo $\{t_0, t_1, t_2, t_3, t_4\}$ o conflito foi resolvido

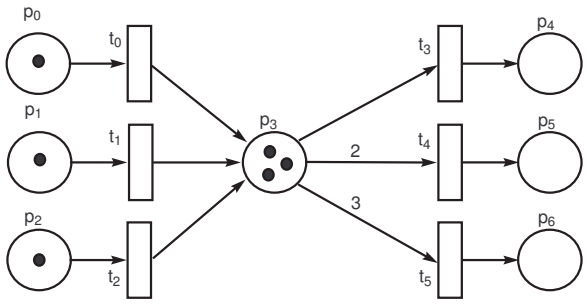

Figura 4.3: Rede de Petri com Conflitos Dinâmicos

em favor de t_3 e t_4, enquanto no passo $\{t_0, t_1, t_2, t_5\}$ o conflito foi resolvido em favor de t_5. Do mesmo modo que as heurísticas de ordenação apresentadas na abordagem de *interleaving*, a maneira como os conflitos são resolvidos (quais transições são privilegiadas) durante a criação do conjunto de passos máximos, determina a ordem em que os passos aparecem no conjunto final, e essa ordem, guia a exploração através do espaço de estados, definindo qual passo máximo será primeiramente explorado.

Um algoritmo para resolução de conflitos e criação do conjunto de passos máximos (MST), a partir de um conjunto de transições disparáveis, é apresentado na Figura 4.4. Seu funcionamento se baseia na tentativa de semi-disparo de cada transição pertencente a FT (linha 10). Neste contexto, semi-disparo significa que os *tokens* dos lugares de entrada são removidos, mas nenhum *token* é colocado nos lugares de saída [32]. Desse modo, uma transição pode ser semi-disparada quando todas as suas pré-condições são atendidas. Essa estratégia permite que os conflitos sejam resolvidos à medida que as transições são semi-disparadas, pois nenhuma transição nova é habilitada e somente transições em conflito dinâmico são desabilitadas quando uma das transições participantes do conflito semi-dispara.

A parte central do algoritmo (iteração interna - linhas 7 a 12) cria um passo máximo através de uma seqüência cíclica de tentativas de semi-disparo das transições do conjunto FT. Como a transição de início da seqüência cíclica é modificada a cada execução da iteração externa do algoritmo (linhas 3 e 17), um novo passo máximo pode ser gerado a cada iteração. Caso o passo recém gerado seja realmente um passo ainda não processado,

```
1  Conjunto-Passos-Maximo(FT)
2    st : um passo a ser computado
3    Para i := 1 até |FT(s)| Faça
4         registre a marcação atual
5         zere st
6         j := i
7         Faça
8              Se (é possível semi-disparar(t_j ∈ FT) Então
9                   Faça inclua t_j em st
10             Fim
11             incremente j ciclicamente */(j := (j % |FT|) + 1)*/
12        Enquanto (j ≠ i)
13        Se (st ∉ MST) Então
14             Faça inclua st em MST
15        Fim
16        restaure a marcação original
17   Fim
18 Retorne MST
```

Figura 4.4: Algoritmo Para Criação do Conjunto de Passos Máximo

este é adicionado ao conjunto de passos máximos (linhas 13 a 15). O mecanismo para restauração das marcações (linhas 4 e 16) garante que a marcação original não é modificada a cada ciclo de criação de um passo máximo.

É importante destacar que a heurística para resolução de conflitos desse algoritmo é determinada pela ordem das transições no conjunto de transições disparáveis. Assim, qualquer mudança na ordem em que as transições do conjunto FT são consideradas, determina qual passo máximo será o primeiro a ser gerado. Com base nisso, as mesmas heurísticas de ordenação aplicadas no algoritmo de *interleaving*, são também aplicadas sobre o conjunto de transições disparáveis, no algoritmo de passo máximo. De modo que o primeiro passo a ser considerado possua as transições com maior prioridade e pertencentes à tarefa com o *deadline* mais eminente.

4.3.2 Definição do Domínio de Disparo

O segundo ponto a ser considerado na definição de um passo máximo está relacionado com a questão temporal, ou melhor, com a definição de um domínio de disparo para o passo.

O exemplo da Figura 4.5, toma a mesma rede apresentada anteriormente, acrescentando, porém, os intervalos de disparo de cada transição.

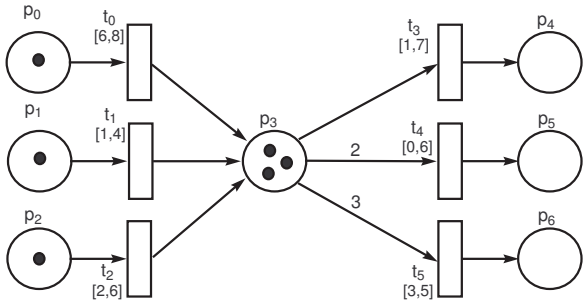

Figura 4.5: Rede de Petri com Temporização e Conflitos Dinâmicos

De acordo com os valores apresentados para os intervalos, o diagrama temporal para o disparo das transições pode ser visto na Figura 4.6. Como pode ser visto no diagrama, e segundo a Definição 3.8, o conjunto de transições disparáveis neste estado é $FT(s) = \{t_1, t_2, t_3, t_4, t_5\}$, uma vez que, devido à política de *disparo obrigatório*, após o instante 5 o estado é obrigado a mudar pelo disparo de um passo que inclua a transição t_1. Dessa forma, nenhum disparo após o instante 5 faz sentido neste estado. Esta restrição, obrigatoriamente, exclui t_0 do conjunto de transições disparáveis.

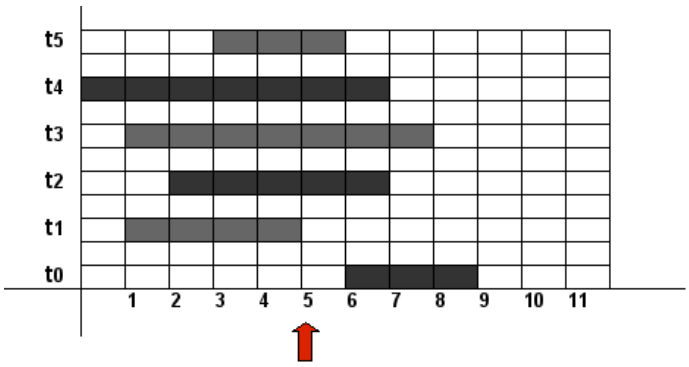

Figura 4.6: Diagrama Temporal de Disparo para Seis Transições

Com base no conjunto de transições disparáveis $FT(s)$, o conjunto de todos os passos máximos para este estado é $MST(s) = \{\{t_1, t_2, t_3, t_4\}, \{t_1, t_2, t_5\}\}$.

De acordo com a fundamentação fornecida na Seção 3.2, o domínio de disparo para o primeiro elemento do conjunto de passos máximos deste exemplo é igual a $FDS_s(st_1) = [2, 4]$, como pode ser observado na Figura 4.7(a). Para o segundo elemento, o domínio de disparo é igual a $FDS_s(st_2) = [3, 4]$ (Figura 4.7(b)). Estes limites estabelecem o intervalo máximo de disparo comum entre as transições em cada passo, o que está de acordo com os conceitos de passo máximo (Definição 3.13), independência entre transições (Definição 3.9) e domínio de disparo de um passo (Definição 3.11) apresentados anteriormente. Entretanto, não se pode deixar de observar que, embora caracterizando exatamente o que é um *passo máximo em um estado*, este conceito despreza o fato de que algumas transições, dentro de cada passo, já estão aptas a disparar muito antes do intervalo de disparo do passo a que pertencem. No passo st_1, por exemplo, a transição t_4 já está apta a disparar no instante $\theta = 0$. No instante $\theta = 1$, as transições t_1, t_3 e t_4 estão aptas a disparar, mesmo assim, somente poderão disparar no instante $\theta = 2$, quando todo o passo está apto a disparar.

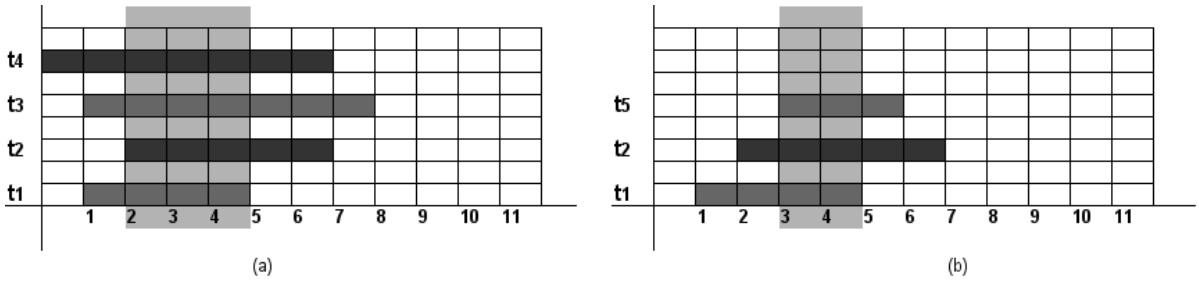

Figura 4.7: Diagrama Temporal de Disparo: (a) Passo st_1; (b) Passo st_2

Caso de deseje contornar essa situação, considerando a possibilidade de disparo de um subconjunto de transições de um passo que esteja apto a disparar em um instante inferior ao domínio de disparo do passo, duas novas definições precisam ser consideradas.

Definição 4.2 (Domínio de Disparo Estendido de um Passo) *Sejam s um estado em uma rede de Petri com temporização, $FT(s)$ o conjunto das transições disparáveis em*

s, e st um passo qualquer pertencente a $ST(s)$. O domínio de disparo estendido de st é definido pelo seguinte intervalo:

$$EFDS_s(st) = [\min(TDI_D(t_i)),\ \min(TDF_D(t_k))],\ \forall t_i \in st,\ t_k \in FT(s).$$

Esta definição estendida de domínio de disparo de um passo, toma como limite inferior, não mais o maior tempo dinâmico de disparo inicial (TDI_D) entre as transições do passo, e sim o menor. Desse modo, o intervalo de disparo é estendido em seu limite inferior para considerar o instante em que pelo menos uma das transições do passo está apta a disparar.

Definição 4.3 (Passo Máximo Local) *Sejam $st \in MST(s)$ um passo máximo do conjunto de passos máximos do estado s e $\theta \in EFDS_s(st)$ um instante no tempo pertencente ao domínio de disparo estendido do passo st. O passo máximo local no instante θ é definido por:*

$$stl_{st}(\theta) = \{t \in st \mid TDI_D(t) \leq \theta \forall t \in st\}$$

Segundo esta definição, um passo máximo local é definido em função de um passo máximo e de um instante de tempo dentro do intervalo de disparo estendido deste passo. Desse modo, um passo local é um subconjunto de transições de um passo máximo que pode disparar simultaneamente em um instante θ, onde θ pode ser menor que o intervalo de disparo comum entre todas as transições pertencentes ao passo máximo.

4.3.3 Algoritmo para Síntese de Escalonamento Estático - Passo Máximo

Nesta seção, duas versões ligeiramente diferentes do algoritmo para síntese de escalonamento utilizando passo máximo são apresentadas.

Os algoritmos propostos possuem a mesma estrutura e seguem, exatamente, o mesmo princípio de busca em profundidade encontrada no algoritmo de *interleaving* apresentado

```
1 Sintese-Passo-Maximo(S, M^F, e_max)
2    Se (S.M = M^F) Retorne VERDADEIRO
3    Faça rotule S como estado já visitado
4    Faça FT := heuristicas-ordenação(estados-indesejados(disparaveis(S,e_max))))
5    Faça MST := Conjunto-Passos-Maximos(FT)
6    Se (|MST| = 0) Retorne FALSO
7    Para cada (st ∈ MST) Faça
8         Para cada (θ ∈ dominio-disparo-passo(st)) Faça
9              S' := disparar(S, st, θ)
10             Se (S' ainda não foi visitado) e (Sintese-Passo-Maximo(S', M^F, e_max)) Então
11                  Faça inclua (S, S', st, θ) na TLTS
12                  Retorne VERDADEIRO
13             Fim
14        Fim
15   Fim
16   Retorne FALSO
17 Fim
```

Figura 4.8: Algoritmo para Síntese de Escalonamento Estático - Passo Máximo

anteriormente (Seção 4.2.3). Nas versões para passo máximo, o mecanismo de redução por ordem parcial não é utilizado. No entanto, do mesmo modo como ocorre na versão *interleaving*, o conjunto de passo a ser considerado é gerado sobre um conjunto ordenado de transições, de modo que os passos gerados também sejam ordenados segundo as heurísticas descritas na Seção 4.2.2.

Como a criação do conjunto de passos máximos necessita de um mecanismo de solução de conflitos, a função de geração do conjunto de passos a ser analisado é bem mais elaborada que a simples seleção necessária no algoritmo de *interleaving*. O algoritmo para solução de conflitos e geração do conjunto de passos máximos é apresentado na Seção 4.3.1. É importante observar que na ausência de conflitos no conjunto se transições disparáveis, um único passo com todas as transições disparáveis pode ser gerado.

A primeira versão do algoritmo (Figura 4.8), chamada simplesmente de "Síntese-Passo-Máximo", utiliza as definições de passo máximo (Definição 3.13) e domínio de disparo de um passo (Definição 3.11) na criação do espaço de estados a ser analisado. Isto pode ser observado nas linhas 8, 9 e 11 da Figura 4.8.

É importante destacar que esta vesão do algoritmo pode produzir atrasos desnecessários no disparos de algumas transições, uma vez que a definição em que esta baseada não contempla o disparo de um subconjunto de transições pertencentes ao passo máximo que já estejam habilidatas entes do início do intervalo de disparo do passo, como discutido anteriormente (Seção 4.3.2)

A segunda versão, chamada de "Síntese-Passo-Máximo-Local" (Figura 4.9), considera as definições de passo máximo local (Definição 4.3) e domínio de disparo estendido (Definição 4.2), como pode ser visto nas linhas 8, 9 e 11 da Figura 4.9.

```
1 Sintese-Passo-Maximo-Local(S, M^F, e_max)
2    Se (S.M = M^F) Retorne VERDADEIRO
3    Faça rotule S como estado já visitado
4    Faça FT := heuristicas-ordenação(estados-indesejados(disparaveis(S,e_max))))
5    Faça MST := Conjunto-Passos-Maximos(FT)
6    Se (|MST| = 0) Retorne FALSO
7    Para cada (st ∈ MST) Faça
8         Para cada (θ ∈ dominio-disparo-estendido(st)) Faça
9              S' := disparar(S, passo-maximo-local(st, θ), θ)
10             Se (S' ainda não foi visitado) e (Sintese-Passo-Máximo-Local(S', M^F, e_max)) Então
11                  Faça inclua (S, S',passo-maximo-local(st, θ),θ) na TLTS
12                  Rertorne VERDADEIRO
13             Fim
14        Fim
15   Fim
16   Retorne FALSO
17 Fim
```

Figura 4.9: Algoritmo para Síntese de Escalonamento Estático - Passo Máximo Local

Nesta versão do algoritmo, as definições utilizadas associam o conceito de passo máximo não mais a definição global do estado, mas sim a uma definição local comprometida com um instante no tempo.

Desse modo, a medida que o tempo se desloca dentro do domínio de disparo estendido, um novo passo máximo local (subconjunto de um passo máximo do estado) é considerado.

Esta semântica de passo máximo comprometida com um instante no tempo, mostra-se mais adequada a síntese de escalonamento, uma vez que sua aplicação não produz atrasos

desnecessários em transições aptas a disparar

4.3.4 Aplicação do Algoritmo

A Tabela 4.3 ilustra a aplicação do algoritmo de passo máximo, utilizando o conceito de passo máximo local, sobre o mesmo modelo utilizado no exemplo com o algoritmo de *interleaving* (tarefas da Tabela 3.2). Os valores apresentados são: o número dos estados visitados, o conjunto de transições habilitadas, o conjunto de transições disparáveis, o passo disparado com o respectivo tempo de disparo e a energia acumulada em cada estado. O exemplo considera que ambas as tarefas consomem $2nJ$ de energia.

Como pode ser observado pala seqüência de estados gerados (Tabela 4.3), neste exemplo, a ausência do mecanismo de ordem parcial evita a condição de conflito apresentada na execução do algoritmo de *interleaving*. De modo que a situação na qual o algoritmo necessita retroceder alguns estados, não ocorre quando avaliado através do algoritmo de passo máximo.

A aplicação do conceito de passo máximo local ocorre logo no estado 2 quando duas transições independentes (tr_1 e tr_2) fazem parte do conjunto de transições disparáveis. Pela definição inicialmente apresentada de passo máximo, o passo a ser disparado seria $\{tr_1, tr_2\}$ e tr_1 teria que disparar juntamente com $tr2$ no tempo 2. Entretanto, considerando a definição de passo máximo local, tr_1 pode disparar antes de tr_2, uma vez que já está apta a disparar no instante 0.

A Figura 4.10 apresenta o diagrama temporal referente a TLTS gerada pela execução do algoritmo. É importante observar que para este exemplo, mesmo utilizando uma seqüência diferente de disparo, a escala gerada pelo algoritmo de passo máximo é equivalente à escala gerada pelo algoritmo de *interleaving*, já que as transições de computação ocorrem exatamente no mesmo instante de tempo em ambas as abordagens. Isso pode ser comprovado pela avaliação dos tempos e transições contidas em cada passo disparado nas Tabelas 4.3 e 4.2.

Tabela 4.3: Execução do Algoritmo - Passo Máximo

#	Estado	Habilitadas	Relógios	Disparáveis	Tempo,Passo	Energia
1	0	{tstart}	{0}	{tstart}	0,{tstart}	0
2	1	{tph1,tph2}	{0,0}	{tph1,tph2}	0,{tph1,tph2}	0
3	2	{ta1,td1,tr1,ta2,td2,tr2}	{0,0,0,0,0,0}	{tr1,tr2}	0,{tr1}	0
4	3	{ta1,td1,tp1,ta2,td2,tr2}	{0,0,0,0,0,0}	{tp1}	0,{tp1}	0
5	4	{ta1,td1,tc1,ta2,td2,tr2}	{0,0,0,0,0,0}	{tc1,tr2}	2,{tc1,tr2}	2
6	5	{ta1,td1,tf1,ta2,td2,tp2}	{2,2,0,2,2,0}	{tf1,tp2}	0,{tf1,tp2}	2
7	6	{ta1,ta2,td2,tc2}	{2,2,2,0}	{tc2}	3,{tc2}	4
8	7	{ta1,ta2,td2,tf2}	{5,5,5,0}	{tf2}	0,{tf2}	4
9	8	{ta1,ta2}	{5,5}	{ta2}	1,{ta2}	4
10	9	{ta1,ta2,td2,tr2}	{6,0,0,0}	{ta1,tr2}	2,{ta1,tr2}	4
11	10	{ta1,td1,tr1,ta2,td2,tp2}	{0,0,0,2,2,0}	{tr1,tp2}	0,{tr1,tp2}	4
12	11	{ta1,td1,ta2,td2,tc2}	{0,0,2,2,0}	{tc2}	3,{tc2}	6
13	12	{ta1,td1,tp1,ta2,td2.tf2}	{3,3,0,5,5,0}	{tp1.tf2}	0,{tp1,tf2}	6
14	13	{ta1,td1,tc1,ta2}	{3,3,0,5}	{ta2}	1,{ta2}	6
15	14	{ta1,td1,tc1,ta2,td2,tr2}	{4,4,1,0,0,0}	{tc1}	1,{tc1,tr2}	8
16	15	{ta1,td1,tf1,ta2,td2,tr2}	{5,5,0,1,1,1}	{tf1}	0,{tf1}	8
17	16	{ta1,ta2,td2,tr2}	{5,1,1,1}	{tr2}	1,{tr2}	8
18	17	{ta1,ta2,td2,tp2}	{6,2,2,0}	{tp2}	0,{tp2}	8
19	18	{ta1,ta2,td2,tc2}	{6,2,2,0}	{ta1}	2,{ta1}	8
20	19	{td1,tr1,ta2,td2,tc2}	{0,0,4,4,2}	{tr1}	0,{tr1}	8
21	20	{td1,ta2,td2,tc2}	{0,4,4,2}	{tc2}	1,{tc2}	10
22	21	{td1,tp1,ta2,td2,tf2}	{1,0,5,5,0}	{tp1,tf2}	0,{tp1,tf2}	10
23	22	{td1,tc1,ta2}	{1,0,5}	{ta2}	1,{ta2}	10
24	23	{td1,tc1,td2,tr2}	{2,1,0,0}	{tc1}	1,{tc1,tr2}	12
25	24	{td1,tf1,td2,tr2}	{3,0,1,1}	{tf1}	0,{tf1,tr2}	12
26	25	{td2,tr2}	{1,1}	{tr2}	1,{tr2}	12
27	26	{td2,tp2}	{2,0}	{tp2}	0,{tp2}	12
28	27	{td2,tc2}	{2,0}	{tc2}	3,{tc2}	14
29	28	{td2,tf2}	{5,0}	{tf2}	0,{tf2}	14
30	29	{tend}	{0}	{tend}	0,{tend}	14

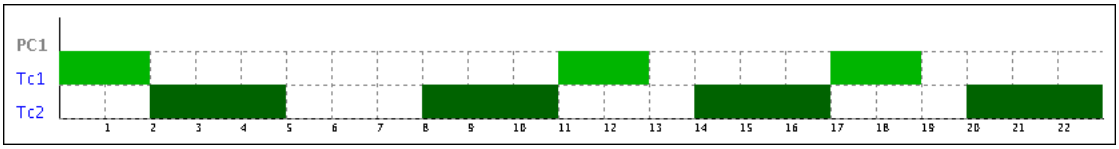

Figura 4.10: Diagrama Temporal para Duas Tarefas

Capítulo 5

Experimentos

A seguir são mostrados os resultados de um pequeno conjunto de experimentos realizados sobre especificações apresentadas em [6, 49, 50]. São apresentados os valores de tempo de execução, número de estados visitados e estados válidos, para a aplicação das duas semânticas (*interleaving* e passo máximo). Os resultados apresentados apontam para a viabilidade da aplicação da semântica de passo máximo, já que mesmo em situações conhecidamente difíceis de serem tratadas com passo máximo (alto nível de concorrência), os valores obtidos não se diferenciaram tanto dos valores para *interleaving*

Todos os experimentos foram realizados utilizando o algoritmo de passo máximo local e os valores de tempo apresentados são médias de 50 execuções, sendo que o melhor e o pior caso foram desconsiderados. Os programas utilizados foram desenvolvidos em linguagem C, de forma não recursiva. Os experimentos foram executados em um computador AMD Athlon 1800 Mhz, com 768 MB RAM, utilizando sistema operacional Linux com compilador GCC 4.0.2.

5.1 Sistema Básico Composto de Duas Tarefas

Este primeiro experimento é um exemplo básico da aplicação das semânticas de *interleaving* e passo máximo. O exemplo reavalia o conjunto de duas tarefas apresentadas na aplicação dos algoritmos no Capítulo 4. Entretanto, para esta avaliação as heurísticas de

ordenação são consideradas. A Tabela 5.1 especifica as restrições temporais para as duas tarefas que compõem o exemplo.

Tabela 5.1: Restrições Temporais para Duas Tarefas

task	ph	r	c	d	p
τ_1	0	0	2	7	8
τ_2	0	2	2	6	6

Neste contexto, a aplicação do algoritmo de *interleaving* encontrou uma escala viável após visitar 37 estados em 0,001154 segundo. O algoritmo de passo máximo local encontrou a mesma escala após 30 estados em 0,001307 segundo.

Para acrescentar uma carga maior de paralelismo a este exemplo, um segundo modelo foi elaborado duplicando-se a estrutura composta pelas duas tarefas. Desse modo, o novo modelo passou a conter um conjunto de quatro tarefas e dois processadores, onde cada processador executa um conjunto similar de duas tarefas.

Nesta situação, a aplicação da semântica de *interleaving* encontrou um escala viável após visitar 72 estados em 0,001910 segundo. O algoritmo de passo máximo local encontrou uma escala viável visitando 30 estados em 0,001575 segundo.

É importante observar que a duplicação do modelo de forma paralela, praticamente dobra o número de estados visitados pelo algoritmo *interleaving*, enquanto o número de estados visitados na semântica de passo máximo permanece inalterado.

5.2 Oxímetro

O oxímetro de pulso é um equipamento responsável por medir a saturação de oxigênio no sangue de uma forma não invasiva. Maiores detalhes sobre o funcionamento e especificações do sistema de controle do oxímetro podem ser obtidos em [6] e [49]. A especificação para o oxímetro utilizada neste experimento é a especificação fornecida em [50] que é apresentado na Tabela 5.2.

Tabela 5.2: Especificação de Tarefas para o Oximetro de Pulso

Tarefa	r	c	d	p	Proc	De	Para	Energia
TE1	0	41	1000	2500	P1	-	-	8576,79 nJ
TE2	371	41	1000	2500	P1	-	-	52,48 nJ
TE3	576	41	1000	2500	P1	-	-	8576,79 nJ
TE4	947	41	1000	2500	P1	-	-	52,48 nJ
TE5	0	45	2000	2500	P1	-	-	222,30 nJ
TA2	141	50	5000	16000	P1	-	-	222,50 nJ
TA3	191	41	5000	16000	P1	-	-	55,76 nJ
TA4	323	50	5000	16000	P1	-	-	222,50 nJ
TA5	382	41	5000	16000	P1	-	-	55,76 nJ
TA6	523	50	5000	16000	P1	-	-	222,50 nJ
TA7	573	41	5000	16000	P1	-	-	55,76 nJ
TA8	714	50	5000	16000	P1	-	-	222,50 nJ
TC1	764	60	5000	16000	P2	-	-	430,2 nJ
TC2	0	50	10000	16000	P2	-	-	444,00 nJ
TC3	0	50	10000	16000	P2	-	-	1117,5 nJ
TC4	764	45	10000	16000	P2	-	-	935,1 nJ
TC5	0	90	10000	16000	P2	-	-	935,1 nJ
TC6	0	90	10000	160000	P2	-	-	7089,3 nJ
TC7	0	90	10000	80000	P2	-	-	935,1 nJ
M1	-	7	-	-	bus1	TA8	TC1	8797,2 nJ

Adicionalmente, as relações ente as tarefas são as seguintes: TE1 PRECEDE TE2, TE2 PRECEDE TE3, TE3 PRECEDE TE4, TA1 PRECEDE TA2, TA2 PRECEDE TA3, TA3 PRECEDE TA4, TA4 PRECEDE TA5, TA5 PRECEDE TA6, TA6 PRECEDE TA7, TA7 PRECEDE TA8.

A sobrecarga do despachante é de 200 microsegundos e seu consumo de energia é de 3958166,22 nJ. Para este experimento a unidade de tempo de tarefa (TTU) adotada é de 100 microsegundos e a restrição de consumo de energia é $2J$.

Neste trabalho, a especificação do oxímetro foi dividida em três conjuntos de tarefas: excitação (TE), aquisição (TA) e controle (TC).

Utilizando a abordagem *interleaving*, uma escala viável foi encontrada em 2,120562 segundos, após a verificação de 42145 estados. Utilizando o algoritmo de passo máximo local uma escala viável foi encontrada em 1,894816 segundo, após a verificação de 36242. Nas duas situações a escala foi encontrada sem nenhuma preempção e o consumo de energia total da escala é de 1,794,314,752.32 nJ ($1.795J$). Estes resultados representam um ganho de mais de 10% no tempo utilizado para busca da escala e de mais de 16% no

número de estados visitados.

Com o objetivo construir um novo cenário onde a abordagem de passo máximo encontrasse maiores dificuldades (baixo nível de paralelismo e alto nível de concorrência), algumas modificações foram realizadas na especificação apresentada anteriormente. Desse modo, apenas um processador foi considerado e todas as relações de precedência foram removidas. Mesmo neste contexto, uma escala viável foi encontrada em 0,649513 segundo, utilizando a semântica de *interleaving*, e em 0,665073 segundos utilizando a semântica de passo máximo local. Sendo que o algoritmo de *interleaving* analisou 17663 estados, enquanto o algoritmo de passo máximo percorreu 17330 estados. O que sugere que mesmo em situação extrema a abordagem de passo máximo mostra-se viável.

5.3 Outros Resultados Práticos

A seguir são apresentados os resultados de dois outros experimentos:

Sistema de Drenagem de Mina. Este é outro exemplo de uma aplicação real, e detalhes sobre sua especificação podem ser obtidos em [11]. Este sistema é aplicado no controle do nível de água em minas. O sistema monitora o nível da água até perceber que o nível máximo foi atingido. Neste ponto o sistema ativa uma bomba de drenagem que permanece ativa até a água atingir o nível mínimo. O sistema também monitora a qualidade do ar no ambiente, condicionando parte de sua operação a situação do ambiente e acionando alarmes quando o os níveis de gás atingem limites críticos.

Este sistema não possui, de fato, restrições temporais muito rígidas mas seu escalonamento é interessante por possuir um grande número de tarefas chegando no mesmo instante de tempo. Na aplicação do algoritmo de *interleaving* para este exemplo, uma escala foi encontrada visitando-se 3255 estados, dos quais somente 3130 foram considerados na escala encontrada (estados válidos). Esta busca consumiu 0,126034 segundo. A aplicação do algoritmo de passo máximo percorreu 2747 estados dos quais 2352 eram estados válidos em um período de 0,149286.

Neste exemplo, a queda de desempenho do algoritmo de passo máximo é justificada pelo alto nível de concorrência proporcionado pelo exemplo.

Veículo Não-Tripulado. Este tipo de veiculo e projetado para percorrer terrenos perigosos e coletar uma diversidade de informações. Uma capacidade semi-automática para formular decisões quanto ao caminho a seguir é disparada quando uma situação de risco inesperado ocorre. Um veiculo não tripulado fornece dois serviços principais: sua própria mobilidade e a coleta de informações de interesse do controlador. De acordo com [43], as tarefas pertencentes a cada um desses serviços são dependentes. Este exemplo possui 14 tarefas que após a determinação do período a ser modelado (período da escala), resultam em 433 instâncias.

Para este exemplo o algoritmo de *interleaving* encontrou uma escala após analisar 16339 estados, dos quais apenas 4701 eram estados válidos. Este processamento foi realizado em 0,320879 segundo.

Pare este mesmo exemplo a estratégia de passo máximo encontrou uma escala após visitar 3755 estados (todos válidos) em um período de 0,238174 segundo.

5.4 Avaliação

	Interleaving		Passo Máximo		Tempo de Execução (s)	
	Válidos	Visitados	Válidos	Visitados	Interleaving	Máximo
Simples	37	37	30	30	0,001154	0,001307
Simples Modificado	72	72	30	30	0,001910	0,001575
Mina	3255	3130	2747	2352	0,126034	0,149286
Veículo	16339	4701	3755	3755	0,320879	0,238174
Oxímetro	42145	42145	36242	36242	2,120562	1,894816
Oxímetro Modificado	17663	17663	17330	17330	0,649513	0,665073

Figura 5.1: Resumo dos Experimentos

A Figura 5.1 mostra um resumo dos valores obtidos nos experimentos apresentados. Apesar dos números encontrados nestes experimentos não poderem ser considerados conclusivos, são bastante promissores, e sugerem que a semântica de passo máximo pode ser

utilizada no contexto da metodologia proposta sem desvantagens quando comparada a semântica de *interleaving*.

Capítulo 6

Conclusões

O disparo simultâneo de um conjunto de transições em uma rede de Petri é muitas vezes citado na literatura como uma semântica alternativa de disparo, sendo capaz de reduzir o impacto do problema de explosão de estado, que é comum à análise realizada por meio de disparos de transições individuais (*interleaving*) [6, 10, 45, 46, 47]. Entretanto, quando o contexto envolvendo restrições temporais é considerado, esta estratégia é nornalmente aplicada em redes de Petri que utilizam o conceito de duração de disparo (*Timed Petri Net*) [18, 21, 27] e não em redes de Petri com intervalo de disparo (*Time Petri Net*). Mesmo ferramentas consagradas de modelagem, análise e verificação em redes de Petri (como por exemplo, INA [46]) não contemplam a possibilidade do disparo de um passo máximo em redes de Petri com temporização (*Time Petri Net*). Assim, não existe de fato, um consenso ou definição formal do que caracteriza um passo máximo em uma rede de Petri com temporização.

Neste sentido, este trabalho propôs um conjunto de definições que caracterizam uma estratégia de disparo de passo máximo sobre redes de Petri com temporização, e aplicou estas definições na síntese de escalonamento estático para sistemas embarcados de tempo real críticos.

A interpretação mais natural do conceito de passo máximo relaciona este conceito a um estado da rede. Entretanto, quando aplicada sobre redes de Petri com temporização, esta interpretação não se mostrou adequado à análise de escalonamento segundo o modelo

proposto. Desse modo, para adequar a semântica de disparo máximo ao contexto do problema de busca por escala, definições especÍficas de passo máximo e domínio de disparo foram estabelecidas, de modo que a semântica proposta estÁ em perfeito acordo com a teoria pré-estabelecida para redes de Petri.

As definições fornecidas iniciam caracterizando *passo máximo* em relação a um estado da rede. Em seguida essas definições são estendidas para um conceito comprometido com um instante no domínio de disparo de um estado (*passo máximo local*). Desse modo, a semântica de passo máximo adotada não impõem restrições que possam caracterizar atraso ou qualquer tipo de limitação que possa influenciar na possibilidade de uma escala viável ser encontrada.

Os resultados obtidos nos experimentos apresentados indicam a viabilidade da aplicação da semântica de passo máximo proposta, no contexto da metodologia na qual este trabalho está inserido, demonstrando melhorias de desempenho relacionadas ao tempo de execução e ao volume de informação processada (número de estados visitados até que uma solução seja encontrada). Mesmo em casoS extremos, os resultados obtidos com a estratégia de passo máximo mostraram a viabilidade dessa estratégia se comparada à abordagem de *interleaving*, já que a quantidade de estado gerada é sempre menor ou igual (pior caso) a quantidade gerada em uma semântica *interleaving*.

É importante destacar ainda, que os aspectos fundamentais da metodologia para desenvolvimento de sistemas embarcados na qual este trabalho esta inserido, não foram modificados. De modo que a maior parte do referencial teórico já existente foi reutilizada e somente alguns elementos novos inseridos ou aperfeiçoados.

Finalmente, além da apresentação de uma abordagem de passo máximo para síntese de escalonamento, este trabalho ainda propôs um conjunto de melhorias e aperfeiçoamentos na metodologia e no modelo para sistema embarcados tomados inicialmente como base. Essas contribuições são apresentadas a seguir.

6.1 Contribuições

Este trabalho estende a proposta apresentada por Barreto em [6], à medida que insere em seu contexto, uma semântica de passo máximo para síntese de escalonamento. As principais contribuições diretamente relacionadas ao objetivo deste trabalho são:

- Um conjunto de definições que caracterizam a aplicação de uma semântica de passo máximo em redes de Petri com temporização;
- A utilização desta semântica na síntese de escalonamento estático;
- Um algoritmo para determinação do conjunto de passos máximo em um determinado estado de uma rede de Petri com temporização.

Adicionalmente, algumas contribuições foram realizadas no aprimoramento do modelo utilizado. Essas contribuições fazem parte do modelo utilizado nesta dissertação, sendo estas:

- Formalização de um conjunto de heurísticas de ordenação que auxiliam na busca por uma escala viável, reduzindo o número de estados visitados e conduzindo a uma escala com o mínimo consumo de energia;
- A definição de consumo de energia como uma função total sobre as transições de uma rede de Petri. Desse modo, toda transição tem um valor de consumo de energia associado, ainda que este valor seja zero. Em relação à definição anterior, que considerava energia como função parcial, esta definição simplifica a análise e representação do modelo;
- Um novo bloco básico representando tarefas com sobrecarga do despachante. Este bloco apresenta as seguintes vantagens em relação à proposta originalmente apresentada no modelo:
  - Representação adequada da possibilidade de preempção. O modelo anterior, de fato não considera esta possibilidade, uma vez que as transições de concessão

de processador de uma tarefa possuem intervalos distintos, e jamais poderão ser consideradas aptas a disparar em um mesmo estado;

- Representação das restrições relacionada a sobrecarga do despachante em uma transição específica. No modelo anterior, essas restrições são representadas separadamente e associadas a outras transições;
- Possibilidade de representação de alocação de tarefas preemptivas e não preemptivas no mesmo processador. O modelo anterior utiliza estruturas distintas para tarefas preemptivas e não-preemptivas quando considerando sobrecarga do despachante;
- Representação das características de computação em uma única transição. O modelo anterior considerava várias transições para representar a computação de uma única tarefa;
- Utilização do mecanismo de prioridade restrita a composição de escala com consumo mínimo de energia. No modelo anterior, a prioridade das transições é essencial para aplicação do modelo.

6.2 Limitações

A seguir são apresentadas algumas limitações presentes no modelo original que este trabalho não contorna, sendo estas:

- O escalonamento estático assume que todas as informações relacionadas ao sistema são conhecidas antes da execução, o que em muitos casos, pode não ser verdadeiro. Porém se as informações sobre o sistema não são previamente conhecidas não é possível fornecer garantias quanto à execução;
- A abordagem trata somente tarefas periódicas e esporádicas;
- A definição do período de tempo a ser modelado (período da escala) é baseada em uma relação de multiplicidade entre os períodos individuas de cada tarefa. Esta

definição pode resultar em períodos muito longos que inviabilizem a análise do modelo;

- O modelo utilizado para a comunicação de tarefas entre processadores distintos não representa adequadamente as restrições impostas à comunicação no tipo de sistema modelado. As limitações estão relacionadas ao fato do modelo não poder garantir a execução sucessiva e imediata de duas tarefas comunicantes. Mesmo o modelo apresentado por Tavares [49], apresenta as mesmas limitações;

- O mecanismo para determinar se um estado já foi visitado consome mais recursos de armazenamento e processamento à medida que o número de estados visitados cresce. Isso pode representar uma limitação para a busca de escalas que exijam um grande número de estados visitados;

- A associação de código ao modelo para tarefas preemptivas, encontra algumas dificuldades semânticas, uma vez que neste tipo de tarefa a transição de computação representa apenas uma unidade de tempo de computação e não a execução da tarefa completa.

6.3 Trabalhos Futuros

Como destacado anteriormente a semântica de passo máximo aplicada neste trabalho reduz o espaço de estados de uma rede de Petri. Considerando este fato, uma extensão possível a este trabalho é a confecção de provas formais de que esta redução não afeta a possibilidade de que uma escala viável seja encontrada.

Um segundo ponto a ser explorado é a experimentação da abordagem em um universo maior de especificações de sistemas. De modo que estas especificações apresentem grau de dificuldade mais elevado para o escalonamento e que estejam mais próximas de sistemas do mundo real.

A metodologia no qual este trabalho esta inserido permite a representação de sistemas muito complexo, com um grande número de tarefas mantendo algum tipo de relação entre

si. Mesmo estando oculto ao projetista, este contexto pode resultar em um modelo com um elevado grau de complexidade. Dessa forma um ponto a ser explorado na síntese de escalonamento estático é a aplicação de mecanismo de redução que considerem as questões temporais na análise do modelo proposto.

Outro ponto a ser explorado é a extensão do modelo para captura um número maior de características presentes em aplicações do mundo real. Neste sentido a proposta de novas relações entre tarefas, como por exemplo *distância mínima*, agregariam bastante valor ao modelo proposto. Nesta mesma área, se encontra o aperfeiçoamento do modelo de comunicação entre tarefas de processadores distintos. Como apresentado anteriormente, a representação atual não captura, adequadamente, as características encontradas nos sistemas do mundo real. Adicionalmente, diversas formas de comunicação são permitidas em sistemas embarcados de tempo real crítico. Formalizar um modelo capaz de representar estas formas distintas de comunicação é um grande desafio para um futuro trabalho.

6.4 Comentário Final

Mesmo sendo adequada a geração de escala, a redução de estados proporcionada pelo passo máximo pode não ser viável quando o interesse não for síntese de escalonamento, já que um grande número de estados é desconsiderado

Referências Bibliográficas

[1] T. F. Abdelzaher and K. G. Shin. Optimal combined task and message scheduling in distributed real-time systems. In *Proc. of the IEEE Real-Time Systems Symposium*, pages 162–171, December 1995.

[2] T. F. Abdelzaher and K. G. Shin. Combined task and message scheduling in distributed real-time systems. *IEEE Trans. Parallel and Distributed Systems*, 10(11):1179–1191, November 1999.

[3] T. A. AlEnawy and H. Aydin. On energy-constrained real-time scheduling. *Proceedings of the 16th EuroMicro Conference on Real-Time Systems (ECRTS 04)*, June 2004.

[4] K. Altisen, G. Göbler, A. Pnueli, J. Sifakis, S. Tripakis, and S. Yovine. A framework for scheduler synthesis. *IEEE Real-Time System Symposium*, pages 154–163, December 1999.

[5] F. Balarin, L. Lavagno, P. Murthy, and A. Sangiovanni-Vincentelli. Scheduling for embedded real-time systems. *IEEE Design and Test of Computers*, pages 71–82, Jan-Mar 1998.

[6] R. Barreto. *A Time Petri Net-Based Methodology for Embedded Hard Real-Time Systems Software Synthesis*. Phd. thesis, Centro de Informática. Universidade Federal de Pernambuco, April 2005.

[7] G. Berthelot. Checking Properties of Nets Using Transformations. In G. Rozenberg, editor, *Advances in Petri Nets*, volume 222 of *Lecture Notes in Computer Science*, pages 19–40. Springer-Verlag, 1986.

[8] G. Bruno, A. Castella, G. Macario, and M. Pescarmona. Scheduling hard real time systems using high-level petri nets. In *Lecture Notes in Computer Science; 13th International Conference on Application and Theory of Petri Nets 1992, Sheffield, UK*, volume 616, pages 93–112. Springer-Verlag, June 1992.

[9] G. Bucci, A. Fedeli, L. Sassoli, and E. Vicario. Modeling flexible real time systems with preemptive time petri nets. *Proceedings of the 15th Euromicro Conference on Real-Time Systems*, pages 279–286, 2003.

[10] G. Bucci, L. Sassoli, and E. Vicario. Mcorrectness verification and performance analysis of real-time systems using stochastic preemptive time petri nets. *Transaction on Software Engineering*, 31(11):913–926, NOVEMBER 2005.

[11] A. Burns and A. Wellings. HRT-HOOD: A structured design method for hard real-time systems. *Real-Time Systems Journal*, 6(1):73–114, 1994.

[12] M. Cornero, F. Thoen, G. Goossens, and F. Curatelli. Software synthesis for real-time information processing systems. *Code Generation for Embedded Processors*, pages 260–279, 1995.

[13] J. Desel and W. Reisig. Place/transition nets. *Lectures on Petri Nets I: Basic Models, LNCS 1491*, pages 122–173, June 1998.

[14] D. Gajski, J. Zhu, and R. Domer. Essential issues in codesign. Technical Report ICS-97-26, Department of Information and Computer Science. University of California at Irvine, June 1997.

[15] M. Garey and D. Johnson. *Computer and Intractability: a Guide to the Theory of the NP-Completeness*. W. H. Freeman and Company, 1979.

[16] P. Godefroid. *Partial Order Methods for the Verification of Concurrent Systems: An Approach to the State-Explosion Problem.* PhD Thesis, University of Liege, Nov. 1994.

[17] D. Harel. Statecharts: A visual formalism for complex systems. *Science for Computer Programming*, 1987.

[18] M. HEINER and P. POPOVA-ZEUGMANN. Worst-case analysis of concurrent systems with duration interval petri nets; btu cottbus, 1996.

[19] Thomas A. Henzinger, Christoph M. Kirsch, and Slobodan Matic. Schedule carrying code. In *Proceedings of the Third International Conference on Embedded Software (EMSOFT)*. Lecture Notes in Computer Science, Springer-Verlag, 2003.

[20] C. Hoare. *Communicating Sequential Process.* Prentice-Hall, 1985.

[21] T. Jucan and C. Vidrascu. Concurrency-degrees for petri nets. *In Proc. of the 1st Conference on Theoretical Computer Science and Informatics Technologies CITTI 2000*, pages 108–114, May 2000.

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

[23] J. Leung and J. Whitehead. On the complexity of fixed-priority scheduling of periodic, real-time tasks. *Performance Evaluation*, 2(4):237–250, December 1982.

[24] J. Lilius. Efficient state space search for time petri nets. In *Electronic Notes in Theoretical Computer Science*, volume 18. Elsevier Science, 1998.

[25] C. Liu and J. Layland. Scheduling algorithms for multiprogramming in a hard real-time environment. *ACM Journal*, 20(1):46–61, January 1973.

[26] P. Maciel. *Petri Net Based Estimators for Hardware/Sofware Codesign.* PhD Thesis, Centro de Informática. Universidade Federal de Pernambuco, Dec 1999.

[27] Louchka Popova-Zeugmann Matthias. A state equation for timed petrinets.

[28] P. Merlin and D. J. Faber. Recoverability of communication protocols: Implicatons of a theoretical study. *IEEE Transactions on Communications*, 24(9):1036–1043, Sept. 1976.

[29] A. K. Mok. *Fundamental Design Problems of Distributed Systems for the Hard-Real-Time Environment.* PhD Thesis, MIT, May 1983.

[30] A. K. Mok. The design of real-time programming systems based on process models. *IEEE Real-Time Systems Symposium*, pages 5–17, 1984.

[31] T. Murata. Petri nets: Properties, analysis and applications. *Proc. IEEE*, 77(4):541–580, April 1989.

[32] R. Pais. *Geração de Executores e Avaliadores de Redes de Petri.* Msc. thesis, Faculdade de Ciências e Tecnologia - Universidade Nova de Lisboa, February 2004.

[33] T. Pering and R. Broderson. Energy efficient voltage scheduling for real-time operating systems. *In Proceedings of the 4th IEEE Real-Time Technology and Applications Symposium RTAS*, June 1998.

[34] J. L. Peterson. Petri nets. *ACM Computing Surveys*, 9(3):223–252, September 1977.

[35] C. A. Petri. *Kommunikation mit Automaten.* PhD Dissertation, Darmstad University, Germany, 1962.

[36] C. Ramchandani. *Analysis of Asynchronous Concurrent Systems by Petri Nets.* PhD Thesis, MIT, Cambridge, USA, February 1974.

[37] W. Reisig. On the semantics of petri nets. *Formal Models in Programming, IFIP 1985*, pages 347–372, 1985.

[38] A. Sangiovanni-Vincentelli and G. Martin. Platform-based design and sofware design methodology for embedded systems. *IEEE Design and Test of Computers*, pages 23–33, November-December 2001.

[39] A. Sangiovanni-Vincentelli and G. Martin. A vision for embedded software. In *Proceedings of the International Conference on Compilers, Architectures and Synthesis for Embedded Systems (CASES'01)*, pages 1–7. Atlanta, Georgia, November 16-17 2001.

[40] L. Sha, R. Rajkumar, and J. Lehoczky. Priority inheritance protocols: An approach to real-time synchonization. *IEEE Transactions on Computers*, 39(9):1175–1185, September 1990.

[41] L. Sha, R. Rajkumar, J. Lehoczky, and K. Ramamritham. Mode change protocols for priority-driven preemptive scheduling. *Real-Time Systems*, 1(3):243–265, December 1989.

[42] T. Shepard and J. A. Gagné. A pre-run-time scheduling algorithm for hard real-time systems. *IEEE Trans. Soft. Engineering*, 17(7):669–677, July 1991.

[43] L. Sieh, P. Haniak, and P. Richardson. Implementing transient fault tolerance in embedded real-time systems. *IEEE Electronics and Information Technology Conference*, 2001.

[44] A. Singhal. Real time systems: A survey. Technical report, Computer Science Department. University of Rochester, December 1996.

[45] P. Starke. *A Memo on Time Constraints in Petri Nets.* Humbolt Universität zu Berlin - Institut für Informatik, 1995.

[46] P. Starke and S. Roch. *INA - Integrated Net Analyzer - Version 2.2.* Humbolt Universität zu Berlin - Institut für Informatik, 1999.

[47] L.J. Steggles. Algebraic prototyping tools for petri nets with time.

[48] V. Swaminathan and K. Chakrabarty. Pruning-based, energy-optimal, deterministic i/o device scheduling for hard real-time systems. *ACM Trans. Embedded Comput. Syst.*, 4(1):141–167, 2005.

[49] E. Tavares. *A Time Petri Net Based Approach for Software Synthesis in Hard Real-Time Embedded Systems with Multiple Processors.* Msc. thesis, Centro de Informática. Universidade Federal de Pernambuco, March 2005.

[50] E. Tavares, P. Maciel, M. Oliveira Jr, B. Souza, R. Barreto, R. Freitas, and M. Custódio. Pre-runtime scheduling considering timing and energy constraints in embedded systems with multiple processors. *In 5th IFIP Working Conference on Distributed and Parallel Embedded Systems (DIPES'2006)*, October 2006.

[51] J. Tsai, S. Yang, and Y.-H. Chang. Timing constraint petri nets and their application to schedulability analysis of real-time system specifications. *IEEE Trans. Software Enginenring*, 21(1):32–49, January 1995.

[52] A. Valmari. The state explosion problem. *LNCS: Lectures on Petri Nets I: Basic Models*, 1491:429–528, June 1998.

[53] W. Wang, A. Mok, and G. Fohler. Pre-scheduling. In *IEEE Transactions on Computers*. IEEE, 2004.

[54] T. Wilmshurtz. An introduction to the design of small-scale embedded systems. 2001.

[55] D. Xu, X. He, and Y. Deng. Compositional schedulability analysis of real-time systems using time petri nets. *IEEE Trans. Soft. Engineering*, 28(10):984–996, October 2002.

[56] J. Xu. Multiprocessor scheduling of processes with release times, deadlines, precedence, and exclusion relations. *IEEE Trans. Soft. Engineering*, 19(2):139–154, February 1993.

[57] J. Xu. On inspection and verification of software with timing requirements. *IEEE Transactions on Software Engineering*, 29(8):705–720, August 2003.

[58] J. Xu and K. Lam. Integrating run-time scheduling and pre-run-time scheduling of real-time processes. In *23rd IFAC/IFIP Workshop on Real-Time Programming*. Shantou, China, june 1998.

[59] J. Xu and D. Parnas. Scheduling processes with release times, deadlines, precedence, and exclusion relations. *IEEE Trans. Soft. Engineering*, 16(3):360–369, March 1990.

[60] J. Xu and D. Parnas. On satisfying timing constraints in hard real-time systems. *IEEE Trans. Soft. Engineering*, 19(1):70–84, January 1993.

[61] J. Xu and D. Parnas. Priority scheduling versus pre-run-time scheduling. In *Real-Time Systems*, volume 18, pages 7–23. Kluwer Academic Publishers, January 2000.

[62] M. Young and L-C. Shu. Hybrid online/offline scheduling for hard real-time systems. *2nd International Symposium on Real-Time and Media Systems*, pages 231–240, July 1996.