Letícia Mara Peres

PROPOSTA DE UM MÉTODO DE VERIFICAÇÃO POR TEMPO GLOBAL COM REDES DE PETRI NO DESENVOLVIMENTO DE SOFTWARE EMBARCADO E EM TEMPO REAL

Texto apresentado ao Programa de Pós-Graduação em Informática do Setor de Ciências Exatas da Universidade Federal do Paraná, como requisito parcial para a obtenção do título de doutor.
Orientador: Prof. Dr. Luis Allan Künzle
Coorientador: Prof. Dr. Eduardo Todt

Curitiba
2010

Peres, Letícia Mara
Proposta de um método de verificação por tempo global com redes de Petri no desenvolvimento de software embarcado e em tempo real / Letícia Mara Peres. – Curitiba, 2010.
126 f. : il. ; tab.

Orientador: Luis Allan Kunzle
Co-orientador: Eduardo Todt
Tese (doutorado) – Universidade Federal do Paraná, Setor de Ciências Exatas, Programa de Pós-Graduação em Informática.

1. Redes de Petri. 2. Software -- Verificação
I. Kunzle, Luis Allan. II. Todt, Eduardo. III. Título.

CDD 004.0151

Ministério da Educação
Universidade Federal do Paraná
Programa de Pós-Graduação em Informática

PARECER

Nós, abaixo assinados, membros da Banca Examinadora da defesa de Tese de Doutorado em Informática, da aluna Letícia Mara Peres, avaliamos o trabalho intitulado, "PROPOSTA DE UM MÉTODO DE VERIFICAÇÃO POR TEMPO GLOBAL COM REDES DE PETRI NO DESENVOLVIMENTO DE SOFTWARE EMBARCADO E EM TEMPO REAL", cuja defesa foi realizada no dia 09 de agosto de dois mil e dez, às 09:00 horas, no Auditório do Departamento de Informática do Setor de Ciências Exatas da Universidade Federal do Paraná. Após a avaliação, decidimos pela aprovação da candidata.

Curitiba, 09 de agosto de 2010.

Prof. Dr. Luis Allan Künzle
DINF/UFPR – Orientador

Prof. Dr. Eduardo Todt
DINF/UFPR – Coorientador

Prof. Dr. Leandro Buss Becker
UFSC – Membro Externo

Prof. Dr. Evangivaldo Almeida Lima
UEB – Membro Externo

Prof. Dr. Andrey Ricardo Pimentel
DINF/UFPR – Membro Interno

Prof. Dr. Marcos Alexandre Castilho
DINF/UFPR – Membro Interno

MINISTÉRIO DA EDUCAÇÃO - UNIVERSIDADE FEDERAL DO PARANÁ - Setor de Ciências Exatas - Coordenação de Pós-Graduação em Informática

Agradecimentos

Aos orientadores e amigos Luis Künzle e Eduardo Todt pela grande ajuda e por tornarem este trabalho possível; aos membros da banca examinadora pelas contribuições; a Evan, Nacib e Erik pelas discussões teóricas e práticas sobre a técnica de tempo global; ao Marcos e Fabiano pelas revisões do texto; ao professor Hector Geffner por me receber em sua equipe na UPF/Barcelona e ao professor Paulo Stadzisz na UTFPR/Curitiba.

Agradeço especialmente ao meu marido Fabiano, cujo amor e companheirismo foram fundamentais para a conclusão desta etapa em nossas vidas; a toda minha família e à do Fabiano pelo suporte incondicional.

Aos meus amigos e colegas, da UFPR, UPF, UTFPR, Unioeste, CMC, HSBC, de Maringá, Foz do Iguaçu, Campinas, Barcelona e Curitiba, agradeço pelo apoio e alegria.

Ao apoio operacional da Jucélia, Andrea, Terezinha, Roser e Judith, e dos professores Gustavo Deco, Humberto Gamba, Valéria Arruda, Marcos Sunye, Alexandre Direne, Luis de Bona e equipe do C3SL.

Dedico este trabalho ao Fabiano e, especialmente, à tia Conceição, pela sua luta que é vitoriosa.

Sumário

Lista de Figuras

Resumo

Sistemas embarcados de tempo real são sistemas restritos quanto às funcionalidades e recursos disponíveis. Nesses sistemas, requisitos temporais são tão importantes quanto requisitos funcionais. Nas fases de análise e projeto do software embarcado, um modelo analítico pode auxiliar em atividades de verificação, reduzindo o tempo de desenvolvimento e aumentando a garantia do comportamento correto do sistema.

Este trabalho propõe a aplicação da técnica de tempo global de redes de Petri na verificação de software embarcado de tempo real. Um algoritmo da técnica de tempo global foi proposto e implementado. Um método geral de aplicação desta técnica e uma instanciação deste método foram propostos. Modelos de mapeamento de tarefas em redes de Petri foram desenvolvidos e um modelo de geração de cenários de comportamento para a análise de escalonabilidade para as políticas de prioridade fixa e *earliest deadline first* foi determinado e implementado. A execução e análise de experimentos permitiram observar o método e modelos.

O método proposto neste trabalho objetiva identificar cenários de comportamento que não respeitam as restrições temporais do sistema modelado. No caso de sistemas embarcados e em tempo real esta é uma das principais preocupações das equipes de desenvolvimento de software. Como resultado, desenvolvemos um método que permite a verificação temporal de software embarcado de tempo real com relógios global e relativo, que é um passo relevante para facilitar a aplicação de redes de Petri no contexto do desenvolvimento destes sistemas.

Abstract

Real time embedded systems are function and time constrained. In these systems, time constraints are as important as functional. An analytical model can help in the verification activities during embedded software analysis and design, reducing development time and increasing assurance of proper system behavior.

This work proposes the application of global time technique of Petri nets as an alternative to the verification of real time embedded software. An algorithm of global time technique was proposed and implemented. We proposed a general method of application of this technique and an instantiation of this method. We developed models for the mapping of tasks in Petri nets and we determined and implemented a model for the generation of scenarios for the schedulability analysis considering fixed priority and earliest deadline first policies. The implementation and analysis of experiments allowed us to observe the method and models.

The method proposed in this thesis aims to identify behavioral scenarios which do not meeting the time constraints of the modeled system. This is a major concern for software development teams about embedded and real time systems. As a result, we developed a method allowing time verification of real-time embedded software using global and relative clocks, which is an important step to facilitate the application of Petri nets in the development of these systems.

1 Introdução

1.1 Visão Geral

Em geral, um sistema embarcado é um sistema de computação com hardware e software altamente acoplados, projetados para desempenhar uma função dedicada ou uma aplicação de propósito específico e restrito quanto a funcionalidade e recursos disponíveis. A maioria dos sistemas embarcados consistem de hardware e software limitados e situados dentro de algum produto ou sistema maior.

Sistemas embarcados estão cada vez mais presentes no cotidiano das pessoas. Eles estão em dispositivos como máquinas fotográficas, filmadoras, máquinas de lavar e fornos de microondas, em dispositivos móveis como aparelhos celulares e computadores de mão, bem como em ambientes que exigem alto grau de confiabilidade como automação industrial, comercial e residencial, automóveis, aviões e trens, sistemas de saúde e de defesa (ARTEMIS, 2004; NOERGAARD, 2005).

O desenvolvimento e a utilização de sistemas embarcados têm aumentado. A indústria estima que a sua taxa de crescimento está por volta de 9% ao ano (EBERT; JONES, 2009). Esta taxa é maior que a esperada para o setor de tecnologia da informação em geral e que estes sistemas desempenham um importante papel nas economias de nações desenvolvidas e em desenvolvimento.

Estas estimativas vêm acompanhadas de reflexões a respeito de sua qualidade e de seus custos. A demanda por sistemas embarcados seguros, confiáveis e com arquitetura e componentes reusáveis, mostra-se cada vez maior.

Os sistemas embarcados, além de restrições funcionais, possuem outros tipos de restrições não-funcionais que devem ser consideradas em seu desenvolvimento. Como exemplo estão as restrições de tempo, de energia, de tamanho, de confiabilidade, de recursos de hardware, do sistema operacional e de custo (GANSSLE, 1999; NOERGAARD, 2005; PRESSMAN, 2006). Ainda no contexto de software embarcado, outros requisitos não-funcionais devem ser consi-

derados, como a necessidade de outras unidades de software aplicativo e software básico e a integração entre o software e o hardware.

Como consequência da satisfação das restrições mencionadas, os sistemas embarcados possuem as seguintes características: exigem o controle total do processador e periféricos para executar requisitos funcionais específicos; possuem facilidades para interação com hardware de propósito específico; devem ser eficientes; possuem controle de concorrência de tarefas caracterizando-se como sistemas concorrentes; e, por fim, em sua maioria são sistemas complexos por seus requisitos e por seu desenvolvimento (GANSSLE, 1999; NOERGAARD, 2005; PRESSMAN, 2006; SOMMERVILLE, 1995).

Muitos sistemas embarcados são de tempo real, que são aqueles nos quais a corretude de um comportamento ou resultado não depende somente da corretude ou integridade dos resultados, mas também do intervalo de tempo após o qual os resultados são disponibilizados (GAMBIER, 2004; BARRETO, 2005). Nesses sistemas, requisitos de tempo são tão importantes quanto restrições funcionais.

Normalmente, o atendimento a requisitos temporais está associado a um bom desempenho, i.e., o sistema deve possuir o desempenho necessário para o atendimento destes requisitos. Requisitos que se refiram ao desempenho são normalmente mais difíceis de projetar. Entre alguns requisitos relacionados ao desempenho estão: alocação de recursos e tratamento de prioridades; tempo de resposta; taxa de transferência de dados; tratamento de interrupções e chaveamento de contexto; sincronização; e comunicação entre tarefas (PRESSMAN, 2006).

Para o atendimento dos requisitos temporais, o hardware projetado deve possuir desempenho suficiente para a execução do sistema considerando que mecanismos de coordenação de tarefas são implementados em software e devem atender a condições básicas de tempo real. Alguns destes mecanismos são: semáforos, exclusões mútuas, objetos e eventos de sincronização, filas de mensagens e níveis de prioridade (SCHIEBE; PFERRER, 1992; SOMMERVILLE, 1995).

A complexidade existente nos projetos de construção de sistemas embarcados de tempo real torna esta área um campo importante de pesquisas e desenvolvimento científico e tecnológico. Uma de suas tarefas mais complexas é a garantia de que o software embarcado seja executado de maneira previsível, garantindo suas restrições temporais. São necessários métodos, técnicas e ferramentas que auxiliem nas atividades de construção e de garantia de qualidade de sistemas. Aproximadamente metade dos custos de desenvolvimento são destinados a atividades de garantia de qualidade, mais especificamente atividades de verificação e validação (ARTEMIS, 2004; GANSSLE, 1999).

1.2 Contextualização

O cotidiano das pessoas tem se tornado cada vez mais dependente de sistemas embarcados, desde os sistemas críticos como os das áreas de transporte e medicina, aos de automação residencial e eletrônicos de consumo. Sistemas e arquiteturas existentes têm sido constantemente alterados e têm sofrido evolução constante, bem como novos usos e serviços têm sido projetados e desenvolvidos por um grande número de empresas. Atividades de construção destes sistemas desempenham um importante papel nas economias de nações desenvolvidas e em desenvolvimento (STANKOVIC, 1996; ARTEMIS, 2004; NOERGAARD, 2005).

O software embarcado, como um componente fundamental destes sistemas, deve atender às suas demandas de complexidade. Esta complexidade é provocada pela heterogeneidade dos ambientes e arquiteturas, pelas restrições de projeto, pela necessidade de reatividade do sistema e pela complexidade intrínseca das tarefas. Estas características se refletem diretamente nas restrições que o software deve atender de maneira a participar do correto funcionamento do sistema, sendo o seu componente mais flexível (NOERGAARD, 2005).

Uma classe importante de restrições que deve ser considerada na construção de sistemas embarcados é a de restrições temporais, que exigem que o sistema responda a determinados requisitos funcionais dentro de um quadro de tempo limitado. Softwares inseridos neste contexto são também chamados de *softwares em tempo real* e são considerados como os de desenvolvimento mais complexo e desafiante (PRESSMAN, 2006).

A alta complexidade do projeto de software embarcado leva à necessidade de adoção de métodos de desenvolvimento para a garantia de qualidade do software e, consequentemente, do sistema embarcado. Entre alguns destes métodos de desenvolvimento está a verificação de software, que tem como objetivo a observação do código ou de modelos simplificados que descrevem ou simulam as características do projeto para a antecipação e correção de problemas e soluções.

1.3 Descrição do Problema

Atividades de garantia de qualidade devem ser conduzidas ao longo do desenvolvimento do software com o objetivo de identificar erros nos subprodutos gerados. Processos de garantia de qualidade relevantes compreendem as atividades de verificação e validação. Segundo a norma IEEE 1012 (IEEE, 2004), atividades de validação são conduzidas para se responder à pergunta: “estamos construindo o produto certo?” ou “o software faz o que o usuário requisitou?”, enquanto

que, com as atividades de verificação, pretende-se responder a: “estamos construindo o produto corretamente?” ou “o software está de acordo com sua especificação?”.

Existem duas técnicas fundamentais de *verificação* de software: a verificação dinâmica e a verificação estática. A verificação dinâmica compreende as atividades de testes, que são aquelas associadas à execução do código com o objetivo de detectar defeitos, por exemplo testes de unidade, integração, sistema, regressão. A verificação estática compreende as atividades de verificação formal e de análise. As atividades de verificação formal conferem a corretude de especificações formais, por exemplo através da conferência de modelos (*model checking*) e inferência lógica utilizando provadores de teoremas. As atividades de análise dizem respeito à conferência dos requisitos do software através de inspeção física e do uso de métricas, do código ou de modelos do sistema.

Nas fases de análise e projeto do software embarcado de tempo real, modelos formais podem ser construídos para se tornarem a base da condução de atividades de verificação, que garantam a corretude da especificação e da solução. À medida que o projeto prossegue e mais detalhes sobre a arquitetura do software são obtidos, técnicas de modelagem e verificação analíticas mais sofisticadas podem ser empregadas até a produção do código (HU; GORTON, 1997). A verificação usada nestes pontos do desenvolvimento é a estática. Quando as unidades do código estão prontas, estas podem ser exercitadas com o objetivo de revelar defeitos específicos da codificação. Neste momento se realiza a verificação dinâmica.

Uma das mais importantes restrições de sistemas embarcados de tempo real é que suas tarefas possuem características que devem ser garantidas, como precedência, exclusão, concorrência, alcançabilidade e ausência de bloqueio.

Lavagno *et al*. (LAVAGNO *et al.*, 1998) discutem o uso de modelos de computação como redes de processos de fluxo de dados (KAHN, 1974), autômatos finitos (GILL, 1962) e redes de Petri (MURATA, 1989), na representação de tarefas e de sistemas embarcados. Tais modelos, cada um com suas características, permitem explicitar restrições do sistema, facilitando seu entendimento e sua verificação.

Entre os modelos consagrados na descrição do comportamento de sistemas dinâmicos a eventos discretos, as redes de Petri (RdP) (MURATA, 1989) permitem representar concorrência e diagnosticar bloqueios e alcançabilidade de estados do sistema. São usadas em diferentes métodos de verificação de sistema e software embarcado. Entre estes trabalhos estão os de Ribeiro *et al.* (RIBEIRO; FERNANDES; PINTO, 2005), Barreto (BARRETO, 2005), Cortes (CORTES; ELES; PENG, 2000) e Lime et al. (LIME; ROUX, 2003, 2009) onde a verificação das pro-

priedades do modelo do sistema utiliza fórmulas em lógica temporal (CLARKE; GRUNBERG; PELED, 1999).

No que se refere à análise de restrições temporais de modelos em redes de Petri, duas abordagens complementares merecem atenção: a proposta por Berthomieu *et al.* (BERTHOMIEU; MENASCHE, 1982; BERTHOMIEU; VERNADAT, 2006), baseada em tempo relativo, e a baseada em tempo global, proposta por Lima *et al.* (LIMA; LÜDERS; KÜNZLE, 2008). Esta última permite um cálculo mais preciso da duração de sequências temporais.

Considerando, portanto, a necessidade imperativa de verificação das restrições temporais impostas ao software de sistemas embarcados e o desenvolvimento de técnicas de análise temporal adaptadas a modelos com forte concorrência, o problema a ser considerado neste trabalho de pesquisa é expressado na seguinte questão: "*é possível verificar restrições temporais de tarefas de software embarcado de tempo real, por meio da técnica de análise de tempo global de redes de Petri?*"

1.4 Motivação

A alta flexibilidade do software, a insuficiência ou complexidade dos processos atuais de projeto, validação e manutenção de sistemas embarcados fazem do software embarcado a parte mais custosa e menos confiável desses sistemas. O aumento de sua ubiquidade constitui, devido à necessidade de expansão de áreas de conhecimento e aplicação, uma oportunidade única para a ciência da computação e para a engenharia de software (HENZINGER; SIFAKIS, 2006).

Estima-se que, no mínimo, 40% a 50% do esforço de desenvolvimento destes sistemas sejam empregados em atividades de verificação e validação (GANSSLE, 1999; ARTEMIS, 2004; EBERT; JONES, 2009). Isto se deve ao fato dos sistemas embarcados possuírem fortes restrições de qualidade e confiabilidade, pois compõem sistemas geralmente críticos, que são aqueles que podem provocar perdas materiais significativas como, por exemplo, os que demandam fabricação em massa como os de eletrônica de consumo, ou vitais, que são aqueles dos quais dependem vidas humanas como, por exemplo, os aeronáuticos e médicos.

Projetos de desenvolvimento de sistemas embarcados sofrem constantemente com o aumento da pressão do "tempo-para-mercado" [1] e custos de projeto e manufatura. Atividades exploratórias

[1] "tempo-para-mercado" é a tradução para *time-to-market*, que significa o tempo que um produto demanda para ser disponibilizado para consumo, desde o início de sua criação, ou ideia, até seu efetivo lançamento, ou disponibilidade para comercialização. A "pressão" citada no texto refere-se à necessidade de lançar o produto o quanto antes

de pesquisa na redução de custos e no aumento da qualidade e da confiabilidade no desenvolvimento e no produto de software são necessárias e vêm de encontro às expectativas da comunidade de engenharia (GANSSLE, 1999). Além disso, características específicas de software embarcado, como as análises de tempo real e de concorrência, requerem modelagens e simulações específicas que habilitem o engenheiro de sistemas a avaliar estas questões demandando mais tempo do projeto (HU; GORTON, 1997).

Embora a disciplina de Engenharia de Software tenha sido motivada sobretudo por sistemas complexos de larga escala, a maioria dos quais envolvem requisitos substanciais de tempo real, a maior parte das pesquisas e produtos neste campo endereça-se a temas funcionais (STANKOVIC, 1996).

Pode-se dizer que existem diversas técnicas de modelagem analítica (PRESSMAN, 2006), no contexto de verificação de software de computação pessoal, já consolidadas. Entretando, estudos em computação e engenharia ainda são necessários para a criação ou adaptação de técnicas de verificação, como a simulação e teste de software, para o contexto de software embarcado concorrente e de tempo real.

1.5 Objetivos

Considerando as características de contexto e a descrição de problema apresentados nas seções anteriores, a tese a ser defendida neste trabalho é a seguinte:

"É possível realizar a análise temporal de tarefas do software embarcado, verificando sua corretude, através de um modelo em redes de Petri utilizando a técnica de tempo global de Lima (LIMA, 2007; LIMA; LÜDERS; KÜNZLE, 2008).

Para provar tal tese, foi desenvolvido um método que se propõe a efetuar a análise de tempo global de redes de Petri que representam tarefas no contexto de verificação de software embarcado. Em particular, propõe-se gerar um modelo matemático que permita a aplicação da técnica de tempo global com redes de Petri temporais que representam tarefas do software; adequar o uso de técnicas de redução do espaço de estados ao modelo proposto e implementar a técnica de tempo global e o modelo matemático, verificando as possibilidades de utilização do método.

no mercado, antes que seus concorrentes, produtos ou empresas, se beneficiem ou desgastem a ideia.

1.6 Organização do Texto

Este capítulo introduziu o trabalho de pesquisa descrito neste texto. Foram discutidos alguns conceitos introdutórios para o entendimento do tema, o contexto, o problema a ser resolvido, as motivações e os objetivos do trabalho de doutoramento.

No capítulo 2 são expostas revisões dos temas relacionados à arquitetura de sistemas embarcados e alguns de seus componentes, ao desenvolvimento do software embarcado, a métodos de escalonamento e a modelos de computação. São discutidas as relações entre esses temas e o presente trabalho de pesquisa.

No capítulo 3 é apresentado o resultado da compilação e organização dos conceitos e definições da técnica de tempo global. Inicia-se expondo uma visão geral da técnica, passando pelas definições de redes de Petri temporais, operações de álgebra intervalar e as próprias definições da técnica. O algoritmo criado para a implementação deste técnica também é apresentado neste capítulo, bem como um exemplo de seu uso.

A aplicação da técnica de tempo global como uma ferramenta de verificação das restrições temporais e escalonamento de tarefas do software embarcado é apresentada no capítulo 4. São expostos o método de modelagem e análise de modelos de sistemas embarcados, os exemplos de sua aplicação e os resultados dos experimentos conduzidos para validação da aplicação e implementação.

O capítulo 5 apresenta um modelo de aplicação da técnica de tempo global no contexto do método proposto e estudos de caso tanto deste modelo quanto do método.

Os resultados deste trabalho, bem como possíveis pesquisas futuras estão discutidas no capítulo 6. Nos apêndices são apresentados alguns dos resultados alcançados com atividades realizadas.

2 Sistemas Embarcados e de Tempo Real

Neste capítulo apresentam-se conceitos básicos e trabalhos relacionados ao tema proposto. Iniciamos o capítulo apresentando conceitos de sistemas de tempo real e as estratégias de escalonamento mais usadas. Após apresentarmos o tema de software embarcado, seus principais problemas, uma arquitetura genérica e os processos mais usados no seu desenvolvimento, introduzimos os modelos de computação mais utilizados na modelagem de software embarcado, entre eles as redes de Petri. Nas últimas seções aprofundamos alguns conceitos referentes à engenharia de software, como atividades de prototipação e de verificação, abordando verificação de modelos, análise quantitativa do software embarcado e padrões de projeto.

2.1 Modelos Formais de Computação

Um modelo é uma abstração de um sistema, ou de algum seu elemento, com o propósito de entendê-lo antes de construí-lo. Como um modelo omite detalhes não essenciais, ele é mais fácil de manipular do que a entidade original (RUMBAUGH *et al.*, 1991).

Modelos de computação são meta-modelos que definem componentes e um modelo de execução para as computações que representam. Um meta-modelo é um conjunto de elementos funcionais ou estruturais de composição e de regras de composição que permitem construir uma representação conceitual para o sistema. O meta-modelo deve ser preciso e rigoroso, i.e. formal, evitando ambiguidades na interpretação da representação do sistema. Também deve ser completo, permitindo a construção de uma representação que descreva o sistema (EDWARDS *et al.*, 1997; SGROI *et al.*, 2000). Para meta-modelos que são utilizados na computação de sistemas, alguns itens importantes a serem considerados são: os modelos de comunicação para a troca de informações entre componentes, por exemplo memória compartilhada e troca de mensagens; e modelos de tempo e de troca de eventos (SGROI *et al.*, 2000).

Usualmente, modelos de computação descrevem o sistema como uma coleção hierárquica

de processos, também chamados blocos; uma coleção de atores; uma coleção de tarefas; uma coleção de transições; e a comunicação entre estes elementos sendo realizada por meio de eventos, ou marcas, carregados por sinais, ou canais (SGROI *et al.*, 2000).

Nas próximas seções serão apresentados os modelos de computação mais utilizados no contexto de desenvolvimento de sistemas embarcados, tanto hardware quanto software embarcado.

Eventos Discretos (DE)

No modelo de computação a eventos discretos (DE - *discrete events*), apesar do tempo ser fundamental, este não é visto como um fluxo constante e sim como o intervalo de tempo que transcorre entre eventos significativos. Uma operação do sistema é representada como uma sequência cronológica de eventos, na qual cada evento ocorre em um instante no tempo e marca uma alteração do estado no sistema. Eventos normalmente carregam uma indicação de tempo (*time stamp*) totalmente ordenada, indicando o momento em que ocorrem. Um simulador de eventos discretos usualmente mantém uma fila global de eventos ordenados de acordo com sua indicação temporal.

Um exemplo de um modelo de computação a eventos discretos é aquele realizado pelas linguagens de especificação do comportamento de processos de hardware digital, como a VHDL (IEEE, 2008) ou a Verilog (IEEE, 2001), que foram projetadas como linguagens de descrição de hardware e que também são usadas como linguagens de entrada para um simulador a eventos discretos (SGROI *et al.*, 2000).

Os modelos DE são usualmente derivados de outros modelos de computação sintetizáveis e usados somente com o propósito de verificação funcional e de desempenho por simulação. Ainda, no caso de eventos simultâneos, o modelo DE é ambíguo, pois a ordem de execução de múltiplos processos que tenham eventos com a mesma indicação de tempo não é especificada. Alguns simuladores a eventos discretos resolvem este problema proibindo eventos com a mesma indicação de tempo e inserindo um tempo de atraso mínimo infinitesimal. Esta técnica garante comportamento determinístico, mas não prevê os problemas decorrentes da inserção deste atraso mínimo, como o "comportamento Zeno" no qual o tempo real não avança e uma sequência infinita de tempos de atraso mínimo é produzida (EDWARDS *et al.*, 1997; SGROI *et al.*, 2000).

O modelo DE pode ser representado visualmente por grafos, como é a proposta do trabalho Simulation Graphs de (BUSS, 1995).

Redes de Processos de Kahn (KPN)

Redes de Processos de Kahn (KPN - *Kahn Process Network*), ou redes de processos, constituem-se em um modelo de computação distribuída no qual um grupo de processos sequenciais determinísticos se comunica através de canais FIFO (*first-in first-out* - primeiro a entrar, primeiro a sair) não-limitados. São representadas por um grafo dirigido onde os nós são os processos produtores/consumidores e os arcos são os canais FIFO (AMATRIAIN, 2004; JANTSCH, 2001). Processos leem e escrevem elementos de dados atômicos, ou marcas, de e para canais. Normalmente, a escrita para um canal é não-bloqueada, i.e., a escrita é sempre realizada e não bloqueia o processo, enquanto a leitura do canal é bloqueada, i.e., um processo que lê de um canal vazio o bloqueará e somente poderá continuar quando o canal contiver itens de dados suficientes. Esta restrição de leitura garante o comportamento determinístico da rede. Não é permitido a um processo testar se existe algum item de dado em um canal de entrada sem consumir tal canal. Em um determinado ponto, um processo está "habilitado" ou "bloqueado" esperando por dados sobre um, e apenas um, de seus canais. (SGROI *et al.*, 2000; AMATRIAIN, 2004)

A KPN resultante exibe comportamento determinístico e a única restrição a ser satisfeita quando de sua execução é a ordem parcial naturalmente imposta pelo consumo/produção de marcas. Entretanto, não existe representação explícita de tempo e sua implementação captura apenas a ordem relativa entre computações. O tempo ou mesmo a ordem de execução dos processos não deve afetar o resultado da saída (AMATRIAIN, 2004).

Apesar de as KPN serem desenvolvidas para modelar sistemas distribuídos, têm sido mostrada sua conveniência para modelagem de sistemas de processamento de sinais, onde infinitas sequências de dados (*streams*) são incrementalmente transformadas por processos executados sequencial ou paralelamente. Aplicações também têm sido encontradas em modelagem de sistemas embarcados, que devem operar pelo maior tempo possível com recursos limitados, e sistemas de computação de alto desempenho (SGROI *et al.*, 2000; AMATRIAIN, 2004).

Redes de processos são representadas visualmente como grafos dirigidos onde os nós representam os processos e os arcos são filas FIFO infinitas que conectam estes processos.

Redes de Processos de Fluxo de Dados (DFN)

Redes de processos de fluxo de dados (DFN - *data flow nets*) são um caso especial de KPNs, que têm seu comportamento especificado como uma sequência de disparos de cada processo, ou "ator". Durante cada disparo atômico, o processo ator consome marcas, executa algum com-

portamento usualmente especificado em uma linguagem sequencial e produz marcas. O número de marcas requeridas para cada canal de entrada para um disparo e produzido em cada canal de saída pode ser fixado, para um modelo DFN estático, ou ser dinamicamente escolhido antes de cada disparo. Como nas KPN's, não é a ordem de disparo dos atores que determina seu comportamento determinístico, mas a restrição de teste de leitura, explicada na seção 2.1.

A implementação pode ser muito eficiente, tanto de hardware quanto de software, devido à liberdade de escolha da ordem de disparo dos atores que é garantida pelas restrições de ordem parcial e pelo comportamento determinístico. Entretanto, pode haver um crescimento potencialmente não limitado de filas requeridas para implementar a comunicação sem perda. De acordo com (SGROI *et al.*, 2000) isto pode ser superado através do escalonamento e dimensionamento estático (apenas para a DFN estática) ou impondo um número máximo de filas, o que pode introduzir possíveis bloqueios.

Máquinas de Estados Finitos (FSM)

Este modelo de computação, também chamado "*Automata*", é um modelo que consiste de um conjunto de estados, um estado inicial, um alfabeto de entrada e uma função de transição que mapeia símbolos de entrada e estados correntes em um próximo estado (BOOTH, 1967). A computação começa no estado inicial com uma cadeia de símbolos de entrada. O sistema evolui para novos estados dependendo da função de transição (LAVAGNO *et al.*, 1998). São especialmente úteis para especificar protocolos de controle e o controle sequencial de tarefas.

Este modelo exprime bem a noção de evento, e parcialmente a de atividade, que é um estado entre dois eventos. Entretanto, não exprime a noção de processo, que é a evolução simultânea de diversos processos paralelos. Uma máquina de estados finitos descreve, de fato, apenas um único processo sequencial (CARDOSO; VALETTE, 1997).

Há muitas variações para estas máquinas, por exemplo: máquinas que possuem ações (ou saídas) associadas a transições, como as máquinas tradutoras tipo Mealy; máquinas que possuem ações associadas a estados, como as máquinas tradutoras tipo Moore; máquinas com um ou mais estados projetados como estados de aceitação, como máquinas detectoras de sequência ou reconhecedoras e aceitadoras; ou máquinas com múltiplos estados iniciais, com transições condicionadas sem símbolos de entrada ou símbolos nulos; e máquinas com mais de uma transição para um determinado conjunto de símbolos de entrada e estado, que são máquinas de estado finito não determinísticas (HOPCROFT; MOTWANI; ULLMAN, 2000; LAVAGNO *et al.*, 1998).

Uma máquina de estados finitos pode ser representada visualmente usando um diagrama de estados (BOOTH, 1967). Este modelo é baseado em um grafo onde estados são associados a nós e transições são associadas a arcos representados por flechas e rotuladas com a condição correspondente.

No contexto do desenvolvimento de software e da UML - Linguagem de Modelagem Unificada (BOOCH; RUMBAUGH; JACOBSON, 1998; MARTIN, 1997), um conjunto de linguagens de modelagem de propósito geral, uma máquina de estados finitos pode ser representada pelo diagrama de transição de estados (*Statecharts*)(HAREL, 1987). Nós e arcos possuem notação padronizada, sendo que nós são rotulados com nomes significativos ao estado correspondente, incluindo ações ou atividades possíveis de serem realizadas quando o sistema estiver neste estado; e arcos associados a transições são rotulados com nomes dos eventos, condições ou ações.

Redes de Petri (RdP)

As redes de Petri (RdP) (MURATA, 1989) são um modelo gráfico e matemático para descrever sistemas com características concorrentes, assíncronas, distribuídas, paralelas, não-determinísticas e estocásticas (JANTSCH, 2001). Elas descrevem causalidade e sequenciamento, escolhas não-determinísticas de conflito e concorrência e têm sido aplicadas em diferentes áreas tais como computação distribuída, manufatura, controle, redes de comunicação e transporte (SGROI *et al.*, 2000).

Redes de Petri são uma poderosa técnica de modelagem com as quais é possível descrever uma grande variedade de problemas presentes na maioria dos sistemas concorrentes e de tempo real. RdP não são somente restritas à modelagem de projetos, mas são úteis também na verificação ou análise de propriedades do sistema modelado.

Este modelo consiste de um grafo bipartido com lugares e transições (nós) e arcos que os conectam entre si, representando a possibilidade de ocorrência de um evento que altera o estado da RdP. Uma rede de Petri é executada por regras de disparo que transmitem as marcas, em quantidade definida nos arcos, de um lugar a outro. Tal disparo é habilitado em uma transição quando cada lugar de entrada possui ao menos o número de marcas definidos nos arcos (PETERSON, 1977).

Uma transição pode disparar quando existir um número previamente especificado de marcas em seus lugares de entrada. Se ela dispara, ela consome aquelas marcas de entrada e produz um número fixo de marcas em seus lugares de saída. Desde que lugares possuem múltiplas transições

destino, uma transição pode impedir o disparo de outra removendo marcas de um lugar de entrada compartilhado por ambas as transições. No nível mais abstrato, o comportamento de uma rede de Petri é dado somente pela sequência de disparos de transições que sua estrutura e conteúdo inicial de marcas permitir. Outros modelos, como redes de Petri coloridas, associam valores a marcas e comportamentos mais complexos a transições, mas sempre atômicos e baseados em disparos.

Segundo Amatriain em (AMATRIAIN, 2004) as máquinas de estado finito, as redes de processos de Kahn e as redes de fluxo de dados DFN são subclasses de redes de Petri. As redes de Petri podem ser consideradas um modelo matemático onde é possível configurar equações de estado, entre outros modelos, que governam o comportamento do sistema (AMATRIAIN, 2004).

2.2 Software Embarcado

Um sistema embarcado é um sistema de computação de propósito específico e restrito quanto às funcionalidades e recursos dos elementos que o compõem, ou seja hardware e software. A expressão "sistema embarcado" se deve ao fato de que este tipo de sistema normalmente é inserido em um sistema maior, que pode ou não ser móvel. Algumas das inúmeras aplicações de sistemas embarcados se dão em sistemas industriais, dispositivos móveis, como telefones celulares e computadores de mão, equipamentos em ambientes privados e infraestrutura pública, bem como em contextos que exigem maior grau de confiabilidade, como transporte, saúde e sistemas de defesa. (ARTEMIS, 2004; NOERGAARD, 2005)

Um aspecto complicador na construção de sistemas embarcados é a sua heterogeneidade de ambiente e de aplicação. Esta heterogeneidade é intrínseca a estes sistemas, afetando tanto a eletrônica quanto o software embarcado, e é consequência das inúmeras possibilidades e necessidades de aplicação no mundo real e da diversidade de tecnologias. Ela implica que os sistemas atendam a diferentes requisitos de desempenho, integração, capacidade de armazenamento e energia, exigindo soluções especializadas para cada projeto (STADZISZ; RENAUX, 2007).

Um projeto de sistema embarcado pode envolver desde o desenvolvimento de um hardware único, à customização de um hardware de referência existente e até mesmo a um misto destas duas técnicas (HENNESSY; PATTERSON, 2006).

Os sistemas embarcados possuem outras características que não podem ser relevadas: exigem o controle total do processador e periféricos para executar requisitos específicos; exigem facilidades para interação com o hardware; devem ser eficientes, com implementações voltadas

para melhor desempenho; exigem controle de concorrência de tarefas; são, em grande parte, orientados a eventos discretos; e, por fim, em sua maioria são sistemas complexos em seu desenvolvimento ou pela sua criticidade (GANSSLE, 1999; NOERGAARD, 2005; PRESSMAN, 2006; SOMMERVILLE, 1995).

O software embarcado possui uma participação fundamental no atendimento às restrições de sistemas desse tipo. A complexidade dos sistemas embarcados tem provocado que projetistas considerem implementações mais flexíveis. O aumento do poder dos processadores com ciclos de manufatura do hardware mais caros e gastando mais tempo, tem tornado a implementação baseada em software uma alternativa mais factível. Seu maior grau de flexibilidade e sua possibilidade de satisfazer as restrições do tempo de disponibilidade para o mercado, ou "tempo-para-mercado", tem permitido mover mais e mais requisitos funcionais e não-funcionais para o software (GANSSLE, 1999; NOERGAARD, 2005; PRESSMAN, 2006; BARRETO, 2005; STADZISZ; RENAUX, 2007).

Obviamente, a passagem da implementação de requisitos do hardware para o software provoca um aumento de complexidade no produto e no processo de software. Outro fator que contribui para essa complexidade é a ausência de padrões, métodos, técnicas e ferramentas específicas, como as encontradas para os sistemas baseados em computadores de propósito geral.

Embora a implementação de requisitos em software embarcado possua vantagens quando comparado com o hardware, ele também possui algumas desvantagens. A principal delas é o desempenho. Para reduzi-la, normalmente se utilizam linguagens de montagem ou C. Entretanto esta política afeta a produtividade da equipe e o tempo-para-mercado, bem como itens de qualidade intrínsecos ao software como manutenabilidade e facilidade de leitura do código. Mesmo uma atividade tão crítica quanto a verificação da corretude do projeto, à medida que o software aumenta de importância e de tamanho, aumenta sua dificuldade (BARRETO, 2005).

O software embarcado também necessita de suporte de hardware para a depuração e avaliação de desempenho. Osciloscópios, analisadores lógicos e placas de depuração muitas vezes são as únicas ferramentas para a análise do sistema (BAILEY, 2007). Pesquisas e desenvolvimentos têm sido realizados para aumentar o número de técnicas e ferramentas baseadas em software e diminuir o impacto dos problemas de desenvolvimento mencionados.

2.2.1 Arquitetura do Software Embarcado

Apesar da alta heterogeneidade dos sistemas embarcados e, como consequência, do software embarcado, em (STADZISZ; RENAUX, 2007) propõe um modelo refinado de uma arquitetura geral de sete camadas para sistemas embarcados. Apresentamos resumidamente este modelo de arquitetura com a finalidade do entendimento dos elementos que compõem o software de sistemas embarcados.

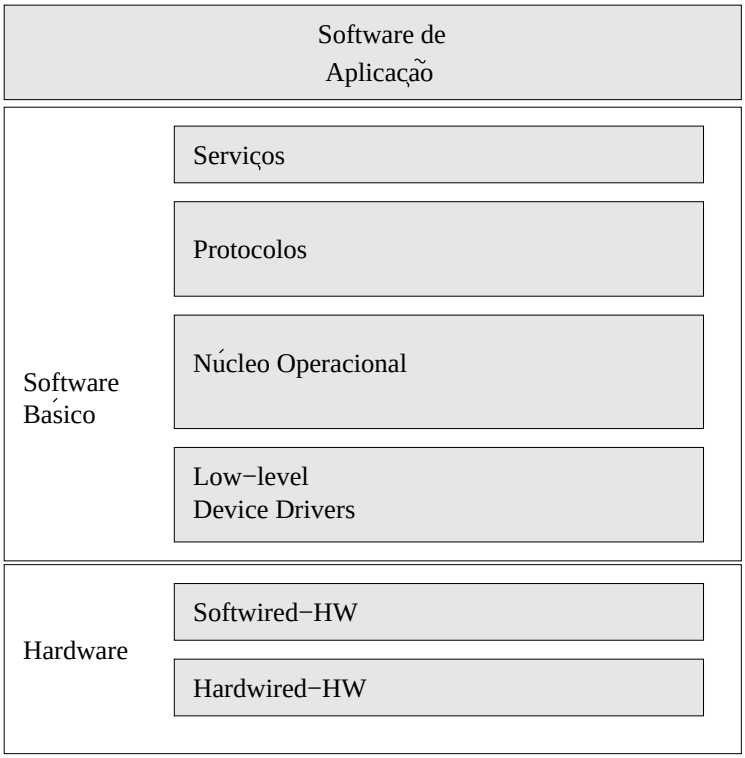

Figura 1: Modelo de sistemas embarcados em sete camadas (STADZISZ; RENAUX, 2007).

Conforme ilustrado na Figura 1, as camadas que compõem este modelo são:

1. *Hardwired-HW*: camada dos dispositivos físicos, onde sinais elétricos são comandados por semicondutores interligados com o objetivo de realizar alguma atividade. Transistores formam portas lógicas e blocos funcionais que, por sua vez, formam processadores, memórias e dispositivos de entrada e saída. As interligações são fixas, em parte realizadas por circuitos integrados e em parte por trilhas em placas de circuito impresso.

2. *Softwired-HW*: camada de software armazenada em dispositivos lógicos programáveis. Esta camada controla a ativação ou não das portas lógicas da camada anterior. Esta conexão se dá via programação, usualmente em VHDL (IEEE, 2008) ou Verilog (IEEE, 2001).

3. *Device drivers de baixo nível*: camada do software básico que interage diretamente com os dispositivos das camadas de hardware (HW) inferiores. É tipicamente estruturada na forma de módulos de software que encapsulam cada um dos dispositivos de hardware. Em geral, as camadas superiores não acessam os dispositivos de hardware diretamente, mas através dos serviços oferecidos pelos elementos desta camada.

4. Núcleo operacional (*kernel*): camada que oferece serviços típicos de um sistema operacional, como escalonamento de tarefas, tratamento de concorrência entre tarefas, gerenciamento de memória e gerenciamento de arquivos. Sistemas operacionais comerciais de sistemas embarcados implementam em maior ou menor grau tais serviços.

5. Protocolos: camada de componentes de software que oferecem serviços de comunicação às camadas superiores, atendendo ao padrão estabelecido pelos protocolos de comunicação utilizados no sistema.

6. Serviços: camada de componentes de software que se referem a serviços que são comuns em diversos sistemas embarcados, o que justifica a implementação destes componentes que são reutilizados em outros sistemas. Exemplos destes componentes são módulos de criptografia e comunicação.

7. Software de aplicação: camada de software que implementa a funcionalidade específica do sistema embarcado.

Destaca-se que nem todos os níveis de serviço devem estar obrigatoriamente presentes na implementação do software embarcado. Ainda, as camadas de software podem ser implementadas em uma ou mais camadas, dependendo das restrições do sistema embarcado, como quantidade e desempenho das funções, celeridade e qualidade de desenvolvimento.

2.2.2 Desenvolvimento de Software Embarcado

Visão Geral

Um processo genérico de desenvolvimento de sistema embarcado inicia-se com a fase de concepção do produto e com a elaboração de um estudo de viabilidade técnica, econômica e comercial. Confirmada a viabilidade deste produto, procede-se à fase de engenharia de requisitos do sistema, que tem como principal objetivo promover o entendimento de seus requisitos e o seu refinamento para a elaboração da especificação do sistema embarcado. Esta fase envolve a

análise de domínio, a análise, modelagem e validação de requisitos e produz a especificação que descreve propriedades funcionais e não-funcionais do sistema a ser desenvolvido, consolidando os diversos resultados das atividades realizadas (STADZISZ; RENAUX, 2007).

Uma vez que os requisitos do sistema foram estabelecidos, passa-se à fase de engenharia de sistema, na qual se inicia o planejamento da solução. No planejamento deve-se observar quais as partes, ou subsistemas, comporão o produto, por exemplo hardware, software, mecânica, ergonomia, entre outros, e quais as dependências entre estes subsistemas. Para isto, realizam-se o particionamento dos requisitos, a identificação e definição dos subsistemas envolvidos, a definição das interfaces entre estes e, por fim, a associação de requisitos aos subsistemas. Como resultados, esta fase produz a identificação e determinação de quais partes serão implementadas em quais componentes, ou subsistemas.

Após a fase de particionamento dos requisitos e mapeamento destes aos subsistemas, certas decisões referentes à ordem de execução das tarefas e processos devem ser tomadas. Este passo do projeto é chamado de escalonamento.

Uma representação formal do sistema permite que ele seja refinado para que novas decisões de projeto sejam incluídas no modelo do sistema. Isto é possível porque um modelo de computação com uma semântica bem definida permite um raciocínio formal sobre cada passo de refinamento durante o processo de projeto (CORTES, 2005).

De posse das especificações de cada subsistema, procede-se o desenvolvimento destas partes, cada qual por uma equipe técnica responsável. Após o desenvolvimento, síntese e validação necessários para cada subsistema, procede-se a sua integração. Em conjunto e posteriormente à integração, devem ocorrer testes para todo o sistema onde é possível verificar se os requisitos definidos anteriormente foram corretamente implementados e se sistema atende às restrições. O teste e depuração de software de tempo real são difíceis devido às restrições temporais impostas a ele e também devido ao seu comportamento concorrente e muitas vezes não-determinístico (PRESSMAN, 2006).

A adoção de métodos e processos rigorosos para estas atividades é extremamente importante (EBERT; JONES, 2009). Correções e ajustes durante o processo de integração e testes podem ser necessários, podendo provocar o retorno a uma das fases anteriores afetando custos e até a qualidade do sistema. Por serem trabalhosas e custosas, as adaptações necessárias muitas vezes são realizadas sobre o software, pois geralmente é o componente mais flexível. Atividades de validação e documentação dos subprodutos do sistema e de gerenciamento do projeto e de configuração devem ser realizadas ao longo das atividades apresentadas.

Uma atividade comum no desenvolvimento de sistemas embarcados é a prototipação, que é a criação de protótipos ou versões incompletas do sistema tanto de software quanto de hardware. As atividades de prototipação de hardware e de software possuem objetivos distintos. Quando desenvolvemos subsistemas de hardware, o protótipo normalmente é usado para validar o projeto, antes da manufatura, já que os custos associados à construção do hardware são altos. Um protótipo de software é normalmente usado para auxiliar a fase engenharia de requisitos, no desenvolvimento e verificação dos requisitos reais do sistema (SOMMERVILLE, 1995).

Algumas metodologias como *Hardware-Software Co-design* (CHIODO *et al.*, 1994), *Platform-Based Design* citada por (BARRETO, 2005) e *Harmony* (TELELOGIC, 2009), têm sido usadas para o desenvolvimento de sistemas embarcados, organizando as atividades apresentadas.

Processos de Desenvolvimento de Software Embarcado

Como apresentado na seção anterior, a partir das especificações de cada subsistema, procede-se ao seu desenvolvimento, cada qual por uma equipe técnica responsável. O desenvolvimento do software como subsistema emprega fases genéricas que podem ser classificadas como apresentado a seguir.

- **Engenharia de requisitos.** Os requisitos do sistema embarcado vêm sendo refinados desde as fases de engenharia de requisitos e engenharia de sistema. Na fase de análise de requisitos do software, os desenvolvedores realizam a alocação de requisitos do sistema ao software, gerando uma documentação da especificação dos requisitos do software. Esta especificação descreve um modelo funcional do sistema que deverá ser validado e será a base das próximas evoluções do processo de desenvolvimento.

  É possível descrever a especificação de requisitos em linguagens textuais, gráficas ou formais. O nosso trabalho está no contexto de linguagens gráficas e formais. As linguagens formais possuem sustentação matemática e permitem expressar requisitos abstratos e concretos (MARZO, 1999), como a notação Z (WORDSWORTH, 1992), máquinas de estados finitos (BOOTH, 1967), Esterel (BERRY, 2000), Lustre (CASPI *et al.*, 1987), uma linguagem que modela redes de fluxo de dados, e redes de Petri.

  Alguns autores têm proposto a utilização de linguagens gráficas que são fracamente formais, por exemplo a UML (BOOCH; RUMBAUGH; JACOBSON, 1998; MARTIN, 1997), para a geração de modelos iniciais e posterior evolução para notações mais formais, por exemplo redes de Petri. Técnicas têm sido desenvolvidas para tal finalidade, por exem-

plos as propostas por (DELATOUR; PALUDETTO, 1998; STAINES, 2008; SOARES; VRANCKEN, 2008) e para posterior verificação formal (SCHMIDT; VARRÓ, 2003), (BARRETO, 2005).

- **Projeto de software.** Nesta fase os desenvolvedores trabalham sobre a especificação dos requisitos gerando uma solução tecnológica que será posteriormente implementada. Pode ser subdividido em projeto arquitetural e detalhado. No primeiro, a solução é definida e representada em termos de seus componentes e de sua organização estrutural, e no segundo, o projeto arquitetural é refinado, sendo possível descrever em detalhes todos os aspectos estruturais e de comportamento dinâmico do software (STADZISZ; RENAUX, 2007).

- **Implementação.** Ao final do projeto, a fase de implementação gera, a partir das especificações de projeto, um programa fonte escrito em uma linguagem de programação. Em sistemas embarcados, esta fase também é conhecida como síntese do software. O código pode ser implementado inteiramente de forma manual ou por meio de geração automática ou semiautomática de estruturas de código a partir das especificações de requisitos em linguagens ou modelos formais ou semiformais (CHIODO *et al.*, 1994; BARRETO, 2005).

- **Verificação e Validação.** Ao longo do desenvolvimento do software embarcado atividades de garantia de qualidade devem ser conduzidas com o objetivo de identificar erros nos subprodutos gerados. Estas atividades podem ser classificadas em atividades de validação ou de verificação. Segundo a norma IEEE 1012 (IEEE, 2004), as atividades de validação referem-se às perguntas: “estamos construindo o produto certo?” ou “o software faz o que o usuário requisitou?” e as atividades de verificação referem-se às perguntas: “estamos construindo o produto corretamente?” ou “o software está de acordo com sua especificação?” (PRESSMAN, 2006).

  Existem duas técnicas fundamentais de *verificação* de software: a verificação dinâmica e a verificação estática. A verificação dinâmica compreende as atividades de testes, que são associadas à execução do código com o objetivo de detectar defeitos, por exemplo testes de unidade, integração, sistema, regressão ou testes estruturais e funcionais. A verificação estática compreende as atividades de análise, que são a conferência dos requisitos do software através de inspeção física, por exemplo a inspeção de código e uso de métricas, e as atividades de verificação formal, que conferem a corretude de especificações formais, por exemplo através da verificação de modelos (*model checking*) e inferência lógica utilizando provadores de teoremas. Estudos na área de verificação normalmente presumem que a especificação do software está correta, por isso a importância das atividades de validação

nas fases anteriores (PRESSMAN, 2006).

- **Prototipação**. Esta atividade compreende a criação de um protótipo para a identificação e conferência dos requisitos do software embarcado. Pode-se caracterizar como subfases desta atividade: identificação dos requisitos básicos, desenvolvimento inicial do protótipo, revisões e simulações, melhoria do protótipo e retorno à subfase de identificação dos requisitos básicos, se necessário. Um protótipo normalmente simula alguns aspectos específicos do software ou sistema. Em software embarcado, um protótipo pode ser usado para verificar a interação entre tarefas, a comunicação entre componentes do sistema, as capacidades de armazenamento e de desempenho, ou mesmo o atendimento do software às restrições temporais.

- **Padrões de Projeto**. Padrão de projeto pode ser entendido como a descrição de um problema ou de um tipo de problema recorrente e uma solução geral para este problema (NAEDELE; JANNECK, 1998). Teve origem no trabalho do engenheiro civil Christopher Alexander (ALEXANDER *et al.*, 1977), que compilou experiências de resolução de problemas associados a projetos de construções em geral. Segundo Christopher cada padrão descreve um problema que ocorre com frequência no ambiente de projeto e o núcleo da solução para tal problema, de maneira que se possa utilizar esta solução várias outras vezes.

  No contexto de redes de Petri, Janneck e Naedele (ALEXANDER *et al.*, 1977) descrevem um conjunto de blocos de construção como padrões de redes para a sistematização do conhecimento de projeto de RdP.

  Em (LIME; ROUX, 2003) os autores apresentam um conjunto de RdP como modelagens de serviços de sincronização. Estas modelagens são apresentadas como estruturas gerais de RdP-T que representam tarefas de software embarcado. Neste trabalho usaremos sua proposta de RdP seguras para a modelagem de tarefas e seu posterior escalonamento. As possibilidades de modelagem não devem ser restritas às estudadas neste texto e nem mesmo ao trabalho de (LIME; ROUX, 2003).

Assim como para a engenharia de software em geral, não existe um modelo de ciclo de vida de software embarcado que seja um padrão único para toda a indústria de software. Não obstante, em geral, modelos de ciclo de vida definem as fases ou atividades que devem ser conduzidas de maneira incontornável ao longo do desenvolvimento do software, determinando as dependências causais diretas entre estas atividades (STADZISZ; RENAUX, 2007), através de processos ou produtos. Alguns modelos de desenvolvimento de software mais comumente utilizados na área

de embarcados são os de modelo em cascata, prototipação, em espiral (PRESSMAN, 2006), em V (SOMMERVILLE, 1995), orientado a modelos (SELIC, 2003), processo unificado-UP (JACOBSON; BOOCH; RUMBAUGH, 1999) e COMET (GOMAA, 2000), entre outros.

2.3 Sistemas de Tempo Real

Sistemas de tempo real são aqueles nos quais o tempo de obtenção do resultado da computação é tão importante quanto sua corretude (STANKOVIC, 1988). A corretude do resultado é observada através da saída gerada pela computação e do estado interno do sistema, por exemplo não alcançando estados proibidos. A corretude do tempo é verificada através da satisfação de restrições temporais, por exemplo datas e prazos limites para a execução de atividades.

Sistemas deste tipo podem ser implementados a partir de simples microcontroladores até os altamente sofisticados e complexos sistemas distribuídos. Algumas aplicações de sistemas de tempo real são: industriais, como controles supervisórios e de aquisições de dados; médicas, como de aplicação robótica em cirurgia robotizada; automotivas e de transporte, como de injeção de combustível, de controle de voos e sistemas inteligentes de controle de tráfego; de defesa, como em sistemas de mísseis teleguiados; aplicações de internet e multimídia, como em sistemas de videoconferência (LIU, 2000).

É essencial que as restrições temporais do sistema sejam garantidamente satisfeitas. Garantir o comportamento temporal requer que o sistema seja *previsível*. Previsibilidade significa que quando uma tarefa do sistema é ativada, deve ser possível determinar seu limite de tempo de término. Também é desejável que o sistema alcance um alto grau de utilização enquanto está satisfazendo restrições temporais (MOHAMMADI; AKL, 2005).

Neste texto, uma tarefa, ou processo, é uma computação que é executada por uma unidade de processamento de uma maneira sequencial. Mais especificamente, definimos uma tarefa como um execução sequencial de código que não suspende a si mesma durante a execução (BUTTAZZO, 2004).

Uma aplicação de tempo real é normalmente composta por múltiplas tarefas com diferentes níveis críticos. Embora a não satisfação de restrições temporais, como tempos estipulados como limite (*deadlines*), não seja desejável, tarefas de tempo real não-crítico podem não satisfazer *deadlines* e ainda assim o sistema continuar trabalhando corretamente. Tarefas podem ser classificadas da seguinte forma: em tempo real crítico, em tempo real não-crítico e em tempo real

firme, dependendo das consequências da tarefa não satisfazer seu *deadline*. Os tempos podem estar em qualquer medida, por exemplo em segundos, milisegundos, microsegundos ou *clocks* do processador (LIU, 2000).

2.3.1 Tarefas em Tempo Real

Tarefas em tempo real crítico (rígido ou *hard*), ou tarefas críticas, são aquelas que devem produzir seus resultados dentro de um certo tempo limite. O sistema é considerado falho caso alguma tarefa de tempo real crítico não produza seus resultados requeridos antes do tempo especificado. Normalmente, estas tarefas controlam operações críticas cuja não satisfação das restrições temporais resulta em consequências desastrosas, em alguns casos com perda de vidas humanas. Sistemas que possuam tarefas de tempo real crítico devem empregar um alto grau de robustez e tolerância a falhas. Exemplos destes tipos de sistemas são os de navegação de aeronaves e de proteção de linhas de alta tensão, equipamentos médicos e de transporte (LIU, 2000; STANKOVIC, 1988).

Tarefas em tempo real firme (*firm*), ou tarefas firmes, são tarefas que, caso os resultados alcançados não estejam dentro do *deadline*, o sistema não falha e os resultados são simplesmente descartados. Em outras palavras, a utilidade dos resultados computados por uma tarefa de tempo real firme que não satisfaça seu *deadline* é zero. Normalmente, estas tarefas controlam aplicações multimídia. Por exemplo, quando um quadro de vídeo chega com um atraso de mais de um segundo, ele é simplesmente descartado, para não haver o risco de ser apresentado desordenadamente em relação aos outros quadros (LIU, 2000).

Tarefas em tempo real não-crítico (*soft*), ou tarefas não-críticas, possuem tempo limite associado, mas as restrições sobre este tipo de tarefa não são expressas em termos do tempo de resposta absoluto, e sim em termos do tempo médio de resposta requerido. Em sistemas que possuam apenas tarefas de tempo real não-crítico, não é catastrófico quando as restrições temporais não são satisfeitas. O não cumprimento de uma tarefa em resposta a um evento em um determinado intervalo de tempo não provoca danos irreversíveis, causando apenas a degradação no comportamento do sistema. Um exemplo deste tipo de sistema são navegadores *Web*. Nestes sistemas considera-se como falha a situação em que as páginas requisitadas levam mais que o tempo médio a que se está acostumado. Isso expressa somente que o desempenho do sistema está degradado e o pior que poderá acontecer é que os dados não serão apresentados tão rapidamente quanto desejado. Outros exemplos são comutação telefônica, jogos eletrônicos, sistemas de transação *on-line*, entre outros (LIU, 2000; BARRETO, 2005).

Cada tarefa que é realizada em sistemas de tempo real tem alguma propriedade temporal que deve ser considerada em sua atribuição ao processador. As principais propriedades temporais de uma dada tarefa se referem a um destes itens (LIU, 2000; ARTIST2, 2008; BUTTAZZO, 2004; MOHAMMADI; AKL, 2005):

- tempo de início (t_s): tempo que uma tarefa inicia sua execução no processador;
- tempo de término (t_f): tempo que uma tarefa encerra sua execução;
- tempo de ativação (*release time*) (t_r): tempo em que a tarefa é inserida na fila de pronto, ou seja, na fila das tarefas que estão prontas para executar.
- tempo de chegada (*arrival time*) (t_a): tempo em que a tarefa recebe a indicação de que deve ser executada;
- *delay* (t_d): tempo mínimo que deve passar antes da execução da tarefa ser iniciada e depois que a tarefa é liberada para execução;
- *deadline* (D): tempo limite no qual a execução da tarefa deve ser completada, depois que a tarefa é liberada para execução. Uma restrição de *deadline* D implica que a tarefa deve ocorrer dentro de D unidades de tempo. O *deadline* absoluto de uma tarefa é o valor de tempo absoluto, i.e. contado a partir do tempo zero, no qual os resultados de uma tarefa são esperados, como se medido por algum relógio físico. O *deadline* relativo é o intervalo de tempo entre o início da tarefa e o instante no qual o *deadline* ocorre ou, em outras palavras, *deadline* relativo é o intervalo temporal entre a chegada de uma tarefa e o *deadline* correspondente.
- tempo de execução ou duração (C): tempo necessário para o processador executar a tarefa sem interrupção;
- pior caso de tempo de execução (*wct*): tempo máximo gasto para a execução da tarefa, depois que esta é liberada para execução. O pior caso de tempo de execução também se refere ao pior caso de tempo de resposta; e
- peso ou prioridade (ϖ): urgência relativa da tarefa.

Considerando um sistema em tempo real consistindo de um conjunto de tarefas $T = \{\tau_1, \tau_2, ..., \tau_n\}$, onde o pior caso de tempo de execução de cada tarefa τ_1 é C_i. O sistema é dito de tempo real se existe ao menos uma tarefa $\tau_i \in T$, que satisfaça uma das seguintes condições (MOHAMMADI; AKL, 2005):

1. A tarefa τ_i é uma tarefa em tempo real crítica. Isto é, a execução da tarefa τ_i deve ser completada dentro de um determinado *deadline* D_i tal que $C_i \leq D_i$.

2. A tarefa τ_i é uma tarefa em tempo real não-crítica. Isto é, quanto mais tarde a tarefa τ_i termina sua computação após o deadline D, mais penalidade ela paga. Uma função de penalidade $P(\tau_i)$ é definida para a tarefa: se $C_i \leq D_i$ então $P(\tau_i) = 0$, senão $P(\tau_i) > 0$. O valor de $P(\tau_i)$ é uma função crescente de $C_i - D_i$.

3. A tarefa τ_i é uma tarefa em tempo real firme. Isto é, quanto mais cedo a tarefa τ_i termina sua computação após o deadline D, mais recompensa ela ganha. Uma função de recompensa $R(\tau_i)$ é definida para a tarefa: se $C_i \geq D_i$ então $R(\tau_i) = 0$, senão $R(\tau_i) > 0$. O valor de $R(\tau_i)$ é uma função crescente de $D_i - C_i$.

O conjunto das tarefas $T = \{\tau_1, \tau_2, ..., \tau_n\}$ pode ser uma combinação de tarefas de tempo real críticas, não-críticas e firmes.

Seja T_S o conjunto de todas as tarefas em tempo real não-críticas em T, i.e. $T_S = \{\tau_{S,1}, \tau_{S,2}, ..., \tau_{S,l}\}$ com $\tau_{S,i} \in T$. A função penalidade do sistema é $P(T) = \sum_{i=1}^{l} P(\tau_{S,i})$.

Seja T_F o conjunto de todas as tarefas em tempo real firmes em T, i.e. $T_F = \{\tau_{F,1}, \tau_{F,2}, ..., \tau_{F,k}\}$ com $\tau_{F,i} \in T$. A função recompensa do sistema é $R(T) = \sum_{i=1}^{k} R(\tau_{F,i})$.

Classificação de Tarefas

O problema geral de escalonamento de tarefas é determinar uma ordem para a execução das tarefas que serão executadas tal que várias restrições sejam satisfeitas. O processo de escalonamento de tarefas em tempo real crítico busca uma ordem para as tarefas tal que as execuções destas sejam completadas antes de um *deadline* geral (LIU, 2000).

Cada escalonador de tarefas é caracterizado pelo algoritmo de escalonamento que ele emprega. Um algoritmo de escalonamento apropriado deve ser projetado baseado nas propriedades do sistemas e de suas tarefas. Algumas propriedades são apresentadas a seguir (LIU; LAYLAND, 1973; STANKOVIC, 1988; LIU, 2000; BARRETO, 2005; MOHAMMADI; AKL, 2005):

Tarefas críticas, não-críticas e firmes. Estas propriedades já foram discutidas nesta seção.

Tarefas dependentes e independentes. Uma tarefa dependente é aquela cuja execução pode requerer informação originada em outra tarefa do sistema ou deve ser iniciada necessariamente

após o término da execução de outra tarefa, ou ambos. Uma tarefa independente é aquela que não possui essas restrições.

Tarefas periódicas, aperiódicas e esporádicas. Tarefas periódicas são aquelas ativadas (*released*) regularmente em intervalos fixos de tempo, chamados período e representados por *P*. Normalmente, tarefas periódicas possuem restrições que indicam quais instâncias de tarefas devem ser executadas uma vez por período.

Tarefas aperiódicas são ativadas irregularmente ao longo do tempo, a uma taxa não limitada e desconhecida de tempo. Normalmente, a restrição temporal usada é um *deadline D*.

Tarefas esporádicas são ativadas irregularmente, a uma taxa limitada e conhecida de tempo. A taxa limitada é caracterizada por um intervalo mínimo de tempo entre duas ativações sucessivas da mesma tarefa. Normalmente, a restrição temporal usada é um *deadline D*.

Uma tarefa aperiódica possui um *deadline* pelo qual ela deve completar, ou deve possuir uma restrição sobre os tempos inicial e final. A maioria do processamento baseado em sensores é periódico por natureza.

Tarefas preemptivas e não-preemptivas. As tarefas preemptivas podem sofrer interrupção do processamento, ou preempção, por outras tarefas com maior prioridade. As tarefas não-preemptivas não podem sofrer interrupção do processamento após o início de sua execução. Quando todas as tarefas do sistema admitem preempção em qualquer ponto da sua execução, diz-se que o conjunto de tarefas admite preempção total. O acesso a recursos compartilhados pode impor restrições sobre o grau de preempção de um conjunto de tarefas.

Tarefas de prioridade fixa e de prioridade dinâmica. No escalonamento dirigido a prioridades, uma prioridade é atribuída a cada tarefa. Para tarefas de prioridade fixa a atribuição das prioridades é realizada estaticamente, i.e., antes da execução do sistema. Para tarefas de prioridade dinâmica a atribuição das prioridades é realizada enquanto o sistema está em execução e pode alterar ao longo do tempo.

Sistemas estáticos e flexíveis. Sistemas estáticos assumem que muitas das informações temporais de cada tarefa, como *deadline*, *delay* e pior caso de tempo de execução, estão disponíveis *a priori* e então seu projeto é estático. Os sistemas flexíveis são aqueles baseados em projetos

dinâmicos, ou seja, estas informações estão disponíveis ou alteram-se enquanto o sistema está em execução.

2.3.2 Escalonamento de Tarefas

Para o atendimento dos requisitos temporais é necessário o escalonamento das tarefas. Um escalonamento é uma atribuição de tarefas ao processador tal que cada tarefa seja executada até sua realização e pode ser definido como a função $\sigma : \mathbb{R} \rightarrow \mathbb{N}$, onde $\sigma(t)$ denota a tarefa que é executada no tempo t. Se $\sigma(t) = 0$ então diz-se que o processador está desocupado (*idle*). Se $\sigma(t)$ altera seu valor em algum momento, então diz-se que o processador realiza uma troca de contexto. Este escalonamento pode ser baseado apenas em questões temporais ou baseado na ocorrência de eventos provocados por outros elementos do sistema.

Um escalonamento é dito factível se todas as tarefas podem ser completadas de acordo com um conjunto de restrições especificadas. Um conjunto de tarefas é dito escalonável se existe ao menos um algoritmo que pode produzir um escalonamento factível.

Escalonamento Estático

As interrupções se dão apenas através de temporizadores e o escalonamento é computado *a priori*, ou seja, anteriormente à construção do programa, quando se pode utilizar algoritmos de escalonamento sofisticados.

O comportamento em tempo de execução é determinístico e a interação com o ambiente se dá somente através de monitoração (*polling*) das interfaces, não existindo problemas no uso de recursos compartilhados.

Alguns problemas deste tipo de escalonamento podem ser: a restrição da comunicação externa somente por monitoração; pouca adaptação ao ambiente, pelo escalonamento já haver sido realizado; e problemas no caso de processos longos.

Escalonamento Dinâmico

O escalonamento é construído com o sistema em funcionamento (*on-line*) e baseia-se em um parâmetro, ou prioridade. A fila de tarefas ativadas, ou prontas para executar, é ordenada por prioridades.

Este é um tipo de escalonamento que permite tratar tarefas preemptivas, pois sempre que no topo da fila de tarefas ativadas está uma tarefa com maior prioridade do que a que está executando, esta última é suspensa e retorna à fila, de acordo com sua prioridade, e a nova tarefa é executada.

Como mencionado anteriormente, a atribuição de prioridades às tarefas pode se dar de maneira estática ou dinâmica. No escalonamento dinâmico com prioridade fixa, a atribuição de prioridades pode ser baseada na importância das tarefas (geralmente chamada Prioridade Fixa, ou *Fixed Priority*) ser inversamente proporcional ao período (baseada na Taxa Monotônica, ou *Rate Monotonic*) e inversamente proporcional ao *deadline* (baseada no Deadline Monotônico, ou *Deadline Monotonic*).

No escalonamento dinâmico com prioridade dinâmica, a prioridade é atribuída em cada ponto de escalonamento de acordo com alguma política de atribuição de prioridades e, logo a seguir à atribuição, a tarefa com maior prioridade tem seu início de execução. A seguir apresentamos o escalonamento EDF. Outros escalonamentos que também com prioridade dinâmica são LSF (LST ou LLF – *Least Slack First*), onde as prioridades são inversamente proporcionais ao tempo livre (*laxity* ou *slack*) e FCFS (First Come First Served), onde as prioridades são inversamente proporcionais ao tempo de espera por serviço.

Escalonamento *Earliest Deadline First* Neste algoritmo de escalonamento, a cada ponto de escalonamento a tarefa tendo o menor *deadline* é escolhida para ser executada. Um conjunto de tarefas é escalonável sob EDF se, e somente se, ele satisfaz a condição de que a utilização total do processador para o conjunto de tarefas é menor que 1. Para um conjunto de tarefas em tempo real periódicas $\tau_1, \tau_2, ..., \tau_n$, o critério de escalonabilidade EDF pode ser expresso por (LIU; LAYLAND, 1973):

$\sum_{i=1}^{n} \frac{C_i}{P_i} = \sum_{i=1}^{n} u_i \leq 1$

sendo a utilização $u_i = \frac{C_i}{P_i}$, onde C_i é o tempo de execução e P_i é o período de execução da tarefa τ_i, que é igual ao *deadline*.

2.4 Escalonamento e Redes de Petri

RdP temporizadas (*Timed Petri Nets*) (RAMCHANDANI, 1974) e RdP temporais (*Time Petri Nets*) (MERLIN, 1974; MERLIN; FABER, 1976) são redes de Petri com tempo associado que têm sido utilizadas para modelar e analisar, além de outros problemas, escalonamento de tarefas.

O trabalho de Aalst (AALST, 1995) propõe um método para mapear tarefas, recursos e restrições em RdP temporizadas. Neste trabalho o autor mostra que a partir deste mapeamento é possível utilizar a teoria e algumas ferramentas de RdP padrão para encontrar conflitos, e redundâncias, e limites inferiores e superiores de tempo para a execução da rede.

Trabalhos sobre escalonamento e síntese de código usando redes de Petri começaram a ser desenvolvidos usando redes de Petri de livre escolha (FCPN - *Free-Choice Petri Nets*) e suas extensões (CORTADELLA *et al.*, 2002), (SU; HSIUNG, 2002), (XU; HE; DENG, 2002) e (LIU; DONG, 2006). Os disparos de transições destas redes dependem do valor da marca (abstraído não-deterministicamente) e não de seu tempo de chegada. Isto é, são redes coloridas não necessariamente temporais. Estas redes permitem um escalonamento quase estático (*quasi-static*).

O trabalho de Hsiung e Su (HSIUNG; SU, 2003) utiliza como base extensões temporais de redes FCPN para o escalonamento. Este trabalho tem suas propostas baseadas na síntese do software embarcado de tempo real.

O trabalho de Barreto et al. (BARRETO; MACIEL; CAVALCANTE, 2003) usa Redes de Petri Temporais (RdP-T) para modelar e sintetizar escalonadores a partir da exploração do espaço de estados. Como em (HSIUNG; SU, 2003), este trabalho tem suas propostas baseadas na síntese do software.

Lime and Roux (LIME; ROUX, 2003) propõem um modelo de rede de Petri temporal, a SETPN, para modelar sistemas de tempo real. Estes autores apresentam um conjunto de padrões de projeto de RdP para modelar tarefas com escalonamento preemptivo e provêm um método usando uma representação poliédrica Matrizes de Limites de Diferença (DBM - *Difference Bounds Matrix*)(DILL, 1990).

Furfaro e Nigro em (FURFARO; NIGRO, 2007) propõem uma técnica para análise de escalonabilidade de sistemas de tempo real especificados por redes de Petri temporais (RdP-T). O foco está em padrões de sequência de disparos de transições (tarefas de execução). Um modelo de RdP-T é traduzido em autômatos temporais para a utilização da ferramenta Uppaal (YI; PETTERSSON; DANIELS, 1994). As propriedades de escalonabilidade de tarefas são verificadas através da análise de alcançabilidade. Esta técnica é considerada pelos seus autores como eficiente e escalável.

Em (LIME; ROUX, 2009), Lime e Roux usam os padrões de projeto do trabalho (LIME; ROUX, 2003) para modelar tarefas e implementar as políticas de escalonamento Prioridade Fixa e *Earliest Deadline First*. Em (LIME; ROUX, 2009) as redes são traduzidas em grafos de estados

que são autômatos lineares híbridos e propriedades temporais são verificadas usando a ferramenta de verificação de modelos Hytech (HENZINGER; HO; WONG-TOI, 1997).

2.5 Considerações do Capítulo

Modelos de computação utilizados para a representação do sistema desde a fase de engenharia de requisitos, também são úteis na atividade de prototipação, para facilitar o entendimento do que deverá ser construído. Além desta compreensão do comportamento do sistema, podem ser utilizados como base para a evolução para outros modelos até a síntese do software e do próprio sistema.

Alguns modelos de computação foram apresentados neste capítulo com a intenção de realizar um levantamento do estado da arte dos modelos empregados na verificação de sistemas embarcados. Destacamos as redes de Petri por permitirem representar concorrência e relações de sincronismo, usualmente presentes em sistemas embarcados, mas que não são facilmente representados por outros modelos.

Em sistemas de tempo real é necessária a garantia de que seus tempos serão satisfeitos. Esta é a motivação para a aplicação de técnicas de análise temporal de redes de Petri no contexto de sistemas embarcados em tempo real: a avaliação da garantia de que o software satisfaz as restrições temporais do sistema.

O escalonamento de tarefas é a atividade realizada para organizar as tarefas sobre um ou mais processadores para garantir que elas serão executadas em tempo hábil e sem bloqueio.

Nos próximos capítulos sugerimos a aplicação do método de tempo global no escalonamento estático, i.e., anterior à execução do software, como um meio de validar a política de escalonamento desejada. Esta validação implica que a política de escalonamento permitirá que o software embarcado e suas tarefas satisfaçam as restrições temporais estabelecidas.

Similarmente, métodos de verificação devem garantir que estas restrições temporais sejam satisfeitas e que o sistema seja robusto, previsível e acurado. Este problema é mais difícil em face da concorrência inerente em aplicações em tempo real (BARRETO, 2005).

Métodos formais têm sido propostos para especificar e verificar sistemas de tempo real. A maioria deles não tem sido usada pela dificuldade de aplicação de tais formalismos. Entretanto, percebe-se que acidentes poderiam ter sido evitados caso fossem usados métodos formais (BARRETO, 2005) que, por exemplo, permitissem a detecção de condições de conflito (SHA; RAJ-

KUMAR; LEHOCZKY, 1990) e inversões de prioridade (GARMAN, 1981).

Aqui está uma oportunidade para investigar ferramentas mais amigáveis ao desenvolvedor e que permitam a aplicação de conceitos de tempo real no desenvolvimento de sistemas. Consideramos que redes de Petri são um modelo formal promissor nesse contexto.

Como a aplicação do método de tempo global se dará ao longo do desenvolvimento de software embarcado, que por sua vez está no contexto da construção de sistemas embarcados, introduzimos conceitos, fases e problemáticas referentes à construção do sistema e software. Destacamos neste tema a atividade do desenvolvimento onde este trabalho está inserido: a verificação de software.

Consideramos, então, que nosso trabalho está inserido no contexto da verificação estática do software embarcado (D'SILVA; KROENING; WEISSENBACHER, 2008), mais especificamente na análise estática temporal. A análise estática abrange uma família de técnicas para computar automaticamente informações sobre o comportamento de um sistema sem executá-lo. Propomos a análise estática observando o comportamento temporal através da análise de tempo global de redes de Petri temporais que modelam as tarefas do software embarcado.

Como motivação para a adoção da análise estática temporal, Clarke aponta em (CLARKE *et al.*, 2000) a necessidade da geração de ferramentas de verificação com maior grau de usabilidade. Este autor analisa que, apesar dos avanços na área, os métodos de verificação mais conhecidos, a verificação de modelos e a prova de teoremas, são distantes da prática diária do engenheiro de sistemas embarcados de tempo real. Isto porque são métodos que utilizam linguagens mais distantes desta prática, tornando mais difícil ao desenvolvedor realizar a abstração que pode ser útil para a geração de novas ideias ao desenvolvedor.

A necessidade em prover uma descrição precisa da causa e efeito de cada evento para raciocinar sobre como cada componente se comporta isoladamente, também é apontada por Clarke (CLARKE *et al.*, 2000) como outro fator de dificuldade na utilização dos métodos de verificação de modelos e de prova de teoremas. Clarke considera que "não é imediatamente óbvio como mapear estes tipos de sistemas em um modelo apropriado"(CLARKE *et al.*, 2000).

Quanto a este problema, consideramos que redes de Petri podem auxiliar nesta descrição precisa de causa e efeito, por ser um modelo visual e matemático. A capacidade de visualização torna as RdP mais próximas da prática diária do desenvolvedor e a capacidade matemática as torna mais próximas das teorias formais e de seus benefícios.

O uso de redes de Petri na fase de projeto permite a modelagem de tarefas e suas interações.

As análises estrutural e comportamental dessas redes permitem observar e identificar alguns tipos de comportamentos e problemas na interação entre essas tarefas. A análise temporal das rede de Petri permite a verificação da ordem de execução dessas tarefas e a satisfação de seus requisitos temporais. Durante essa verificação é possível avaliar o comportamento das tarefas modeladas, analisando as configurações previstas e possibilitando uma rápida adaptação aos problemas encontrados e a mudanças que ocorram no sistema a ser produzido.

Neste trabalho as redes de Petri são utilizadas para representar os processos e suas linhas de execução, uma vez que elas têm se mostrado bastante úteis na modelagem e verificação de sistemas embarcados, tanto de hardware quanto de software (KAVI; MOSHTAGHI; CHEN, 2002) (NAEDELE, 2001). Constituem um modelo que conta com uma comunidade acadêmica muito atuante, possuindo seu uso consagrado por diversos grupos de pesquisa. Não apenas por esses fatores, mas também em continuidade às pesquisas desenvolvidas no Laboratório de Inteligência Artificial e Métodos Formais (LIAMF) do Departamento de Informática (DInf) da Universidade Federal do Paraná (UFPR).

3 *Análise por Tempo Global*

O uso de Redes de Petri (RdP) como estrutura formal de representação e modelagem de sistemas dinâmicos permite a análise e a verificação de propriedades tanto estruturais quando comportamentais desses sistemas (MURATA, 1989). As características temporais dos sistemas podem ser representadas em algumas extensões das RdP. Em especial, as Redes de Petri Temporais (RdP-T) (MERLIN, 1974) permitem a associação de uma janela temporal de ocorrência a cada evento do sistema, representado como uma transição da rede. No entanto, esta informação temporal é relativa ao estado do sistema imediatamente anterior ao evento. Este tratamento relativo do tempo nas RdP-T simplifica a representação de informações temporais locais como durações e atrasos, mas dificulta a análise temporal global onde, por exemplo, o interesse está voltado para a localização dos eventos no tempo considerando um relógio global associado ao estado inicial do sistema.

Este capítulo apresenta uma técnica de análise temporal para RdP-T que além de tratar as informações temporais relativas de cada evento os localiza em intervalos temporais de um relógio global. Para descrever esta técnica de análise, primeiramente introduzimos as estruturas formais, operações de álgebra intervalar e as RdP-T. Apresentamos então uma visão geral da técnica, o algoritmo de construção de um grafo de classes de estados da RdP-T e como as informações temporais são extraídas deste grafo. Um exemplo completo empregando esta técnica é construído e discustido e, por fim, apresentamos as considerações finais do capítulo.

3.1 Álgebra Intervalar

Os cálculos dos tempos de disparos em redes de Petri temporais exigem que se defina o conceito de intervalo, bem como um conjunto mínimo de operações sobre intervalos, uma vez que as restrições temporais são apresentadas na forma de intervalos de tempo associados às transições da rede de Petri. As definições apresentadas seguem a formalização usada no trabalho de Mattar

Jr. *et al* (MATTAR JUNIOR *et al.*, 2007).

Definição 1 (Intervalos)**:** Dados dois números racionais $a, b \in \mathbb{Q}$, define-se $[a,b]$ como o *intervalo* entre o *limite inferior* a e *limite superior* b, sendo $[a,b] = \{x \in \mathbb{Q} : a \leq x \leq b\}$. O intervalo $[a,b]$ é denominado de intervalo *próprio* caso $a < b$, intervalo *impróprio* caso $a > b$ e intervalo *degenerado* caso $a = b$. Em um intervalo próprio $[a,b]$, a e b são chamados respectivamente de *limite inferior* e *limite superior* do intervalo.

Definição 2 (Operadores)**:** Dados os intervalos $[a,b]$ e $[c,d]$, definem-se os seguintes operadores:

$$[a,b] + [c,d] = [a+c, b+d]$$
$$[a,b] - [c,d] = [\max\{0, a-d\}, \max\{0, b-c\}]$$
$$[a,b] \ominus [c,d] = [\max\{0, a-c\}, \max\{0, b-d\}]$$

Dados dois intervalos próprios $I_1 = [a,b]$ e $I_2 = [c,d]$, o intervalo resultante de $I_1 + I_2$ sempre é um intervalo próprio. Se $a \leq d$ e $b \leq c$ não ocorrem ao mesmo tempo, o intervalo resultante de $I_1 - I_2$ é um intervalo próprio, caso contrário o intervalo é degenerado.

Definição 3 (Amplitude do intervalo)**:** Dado um intervalo próprio $I = [a,b]$, define-se a *amplitude* do intervalo I como sendo $\alpha(I) = b - a$.

A amplitude de um intervalo próprio obtido com a soma de outros dois intervalos próprios I_1 e I_2 é dada pela soma das amplitudes de I_1 e I_2, ou seja, $\alpha(I_1 + I_2) = \alpha(I_1) + \alpha(I_2)$.

Para dois intervalos próprios $I_1 = [a,b]$ e $I_2 = [c,d]$ com $a > c$ e $b > d$, a amplitude do intervalo resultante de $I_1 \ominus I_2$ é dada pela diferença das amplitudes de I_1 e I_2, ou seja, $\alpha(I_1 \ominus I_2) = \alpha(I_1) - \alpha(I_2)$. No caso de $a > d$ e $b > c$, a amplitude do intervalo resultante de $I_1 - I_2$ é dada pela soma das amplitudes de I_1 e I_2, ou seja, $\alpha(I_1 - I_2) = \alpha(I_1) + \alpha(I_2)$.

Dados dois os intervalos próprios I_1 e I_2, tem-se necessariamente que $\alpha(I_1 \ominus I_2) \leq \alpha(I_1 - I_2)$.

Como exemplo, suponha os intervalos $I_1 = [3,4]$ e $I_2 = [0,2]$, cujas amplitudes são dadas por $\alpha(I_1) = 1$ e $\alpha(I_2) = 2$. Assim temos:

$I_3 = I_1 + I_2 = [3,6]$, cuja amplitude é $\alpha(I_3) = \alpha(I_1) + \alpha(I_2) = 1 + 2 = 3$;

$I_4 = I_1 - I_2 = [1,4]$, cuja amplitude também é dada por $\alpha(I_4) = \alpha(I_1) + \alpha(I_2) = 1 + 2 = 3$;

$I_5 = I_1 \ominus I_2 = [3,2]$, que não é um intervalo próprio.

Neste exemplo, I_3 e I_4 possuem a mesma amplitude, apesar de I_3 ser obtido pela soma de I_1 com I_2 e de I_4 ter sido obtido pela diferença entre os mesmos intervalos I_1 e I_2.

É importante notar que as duas subtrações de intervalos ($-$, $\ominus$) definidas produzem, de maneira geral, resultados muito distintos no que diz respeito à amplitude do intervalo obtido com a operação. Pode-se dizer de maneira simplificada que enquanto o operador $\ominus$ tende a "reduzir" a amplitude dos intervalos, a operação $-$ acaba por "aumentar" a amplitude dos intervalos. Estas características são grande relevância no estudo que será desenvolvido ao longo deste trabalho, pois terão relação direta com a precisão dos resultados atingidos (MATTAR JUNIOR, 2008).

3.2 Redes de Petri Temporais

Três tipos de propriedades têm sido considerados em um modelo de RdP: temporais, estruturais e comportamentais. Propriedades comportamentais são aquelas que dependem da marcação inicial da rede e propriedades estruturais são aquelas independentes de marcação (BARRETO, 2005). Algumas propriedades comportamentais são: alcançabilidade, limitação e segurança, vivacidade, reversabilidade, cobertura, persistência, justiça (*fairness*)(BARROS, 1996) e conservação (MURATA, 1989). As principais propriedades estruturais são: vivacidade estrutural, limitação estrutural, conservação, repetitividade e consistência (MURATA, 1989).

Para análise de propriedades comportamentais, como por exemplo para a análise de bloqueios, é necessária a criação de um grafo de alcançabilidade. Entretanto, a complexidade de criação deste grafo é exponencial, i.e. à medida que o tamanho da RdP aumenta, aumenta exponencialmente o tamanho do espaço de estados deste grafo de alcançabilidade. A comunidade acadêmica da área de RdP vem resolvendo este problema através da técnica de desdobramento (*unfolding*) (MCMILLAN, 1995) (ESPARZA; ROMER; VOGLER, 1996) (ESPARZA; HELJANKO, 2008). O trabalho (CORBETT, 1996) avalia alguns métodos para análise de bloqueio e o trabalho (MELZER; ROMER, 1997) propõe análise de bloqueio utilizando desdobramento.

Situações de bloqueio ocorrem quando um conjunto de processos devem compartilhar um conjunto de recursos comuns e um processo prende indefinidamente os recursos necessários para outros processos do grupo. Normalmente os tratamentos para este problema estão classificadas em detecção e recuperação de bloqueios, prevenção de bloqueios e anulação de bloqueios. Métodos baseados em redes de Petri são utilizados na modelagem e verificação de sistemas para prevenção e anulação de bloqueios. Por exemplo, a ferramenta Tina (LAAS, 2010) implementa a verificação de ausência de bloqueios.

A presença ou ausência de marcas em lugares da rede representa o estado do sistema em um determinado momento no comportamento dinâmico da rede. Baseado nisto, diferentes pro-

priedades podem ser estudadas. Por exemplo, dois lugares marcados simultaneamente podem representar uma situação de risco que deve ser evitada. Este tipo de requisito de segurança pode ser provado formalmente pela verificação de que tal estado perigoso nunca será alcançado. Ainda, pode-se estar interessado na prova de que o sistema alcança um certo estado, no qual a presença de marcas em um lugar em particular representa que uma tarefa foi completada. Este tipo de análise, de ausência ou presença de marcas em lugares da rede, é determinado pela análise de alcançabilidade.

A análise de alcançabilidade é útil mas não informa nada sobre os aspectos temporais. Entretanto, tempo é um fator essencial em aplicações embarcadas. Sobretudo em sistemas de tempo real, onde é crucial o raciocínio quantitativo sobre propriedades temporais para garantir a corretude do projeto.

Extensões temporais de redes de Petri foram desenvolvidas e existem duas grandes classes destas extensões: Redes de Petri temporizadas (*Timed Petri Nets*) e redes de Petri temporais (*Time Petri Nets*). No modelo temporizado uma duração de disparo é associada a cada transição, lugar ou arco. A RdP temporizada, na qual as transições contêm informação temporal, é obtida associando-se a cada transição da rede ordinária uma duração de disparo. Sua semântica é uma noção de atraso durante o qual as marcas usadas para disparar a transição não estão disponíveis ou até mesmo visíveis. Portanto, o disparo não é instantâneo, pois possui uma duração pré-definida na estrutura da rede. Uma rede de Petri temporizada é um par $R =< R, D >$, onde R é uma rede de Petri ordinária ($R =< P, T, Pre, Pos >$) com uma marcação inicial M_0 e $D : T \rightarrow \mathbb{Q}^+$ é a função *duração de disparo*, que associa a cada transição um número racional positivo que descreve a duração do disparo (CARDOSO; VALETTE, 1997).

No modelo temporal, uma duração de sensibilização é associada a cada transição da rede ordinária. O disparo é instantâneo, mas a transição deve permanecer sensibilizada durante o intervalo de tempo dado e o disparo pode ocorrer em qualquer momento nesse intervalo. Diferentemente do modelo RdP temporizada, durante a sensibilização de uma transição as marcas continuam disponíveis ou visíveis a outras transições de saída do lugar. Uma rede de Petri temporal pode ser definida como um par $R =< R, I >$, onde R é uma rede de Petri ordinária ($R =< P, T, Pre, Pos >$) com uma marcação inicial M_0 e $I : T \rightarrow (\mathbb{Q}^+ \times (\mathbb{Q}^+ \cup \{\infty\}))$, $(t, e(t)) \in I$ e $e(t) = [a, b]$, onde a e b são números racionais positivos. I é a função que, a cada transição t, associa um intervalo fechado racional que descreve uma *duração de sensibilização* (CARDOSO; VALETTE, 1997).

Para a análise de propriedades temporais, é necessário não somente verificar que um certo

estado será alcançado, ou um determinado de conjunto de lugares estará marcado, mas garantir que isto ocorrerá dentro das restrições temporais especificadas para a rede. Este tipo de análise é chamado análise temporal de redes de Petri e será apresentado de forma detalhada no capítulo 4.

No método de análise de tempo global apresentado no capítulo 3 a informação temporal está sempre atualizada, o que permite analisar as propriedades temporais de forma quantitativa. É possível mostrar, usando este método, que um dado lugar estará eventualmente marcado e que seu intervalo temporal atende ou não as restrições temporais.

A seguir são apresentados os conceitos e definições referentes a redes de Petri temporais que são úteis para a técnica de tempo global, apresentada mais adiante neste capítulo.

Definição 4 (Redes de Petri Temporais)**:** Uma rede de Petri temporal (RdP-T) R é dada por uma tupla (P, T, Pre, Pos, M_0, I) (BERTHOMIEU; DIAZ, 1991), onde:

- P é um conjunto finito de lugares,
- T é um conjunto finito de transições,
- $Pre : (P \times T) \rightarrow \mathbb{N}$ é a função incidência de entrada em transições, que representa o peso dos arcos de entrada da transição.
- $Pos : (P \times T) \rightarrow \mathbb{N}$ é a função incidência de saída de transições, que representa o peso dos arcos de saída da transição.
- $M_0 : P \rightarrow \mathbb{N}$ é a marcação inicial e
- $I : T \rightarrow (\mathbb{Q}^+ \times (\mathbb{Q}^+ \cup \{\infty\}))$ é a função que associa um intervalo de tempo de disparo a cada transição, sendo $\mathbb{Q}^+$ é o conjunto dos números racionais não negativos.

Uma marcação é um conjunto de atribuições de marcas a lugares da rede. A marcação de um lugar $p \in P$ é denotada $M(p)$ e a marcação de R pode ser representada como um vetor com tamanho m, onde m é o número de lugares de P. Para uma marcação em particular, um lugar é dito estar *marcado* se e somente se $M(p) > 0$.

Uma RdP-T possui um intervalo estático de disparo associado a cada transição $t \in T$, dado por $[a, b]$, $a \leq b$. Os limites a e b representam, respectivamente, o menor e o maior tempos de disparo da transição t contados a partir de sua habilitação. O intervalo de disparo representa os tempos mínimo e máximo que uma transição pode permanecer habilitada antes de disparar causando uma nova marcação na rede. O disparo da transição é instantâneo e deve estar dentro deste intervalo temporal.

Definição 5 (Intervalo de tempo estático)**:** Dada uma rede $R = (P, T, Pre, Pos, M_0, I)$. O intervalo de tempo $e \in I$ associado a cada transição $t \in T$ da rede será denominado de *intervalo de tempo estático* da transição e denotado por $e(t)$.

Definição 6 (Classe de estados)**:** Uma *classe de estados* c_k de uma RdP-T é uma tupla (M_k, W_k), onde M_k é uma marcação da rede e W_k é o conjunto das informações temporais da classe.

O conjunto de informações temporais das classes de estados tem por finalidade representar o comportamento temporal da rede em cada um de seus estados bem como registrar o andamento do relógio de tempo global. A estrutura e os elementos deste conjunto de informações serão apresentados na próxima seção.

Definição 7 (Transição habilitada)**:** Uma transição t está *habilitada* em uma classe $c_k = (M_k, W_k)$ se e somente se todos os lugares do pré-conjunto de t estão devidamente marcados em c_k, ou seja, $\forall p \in P,\ Pre(p,t) \leq M_k(p)$ ou, de forma simplificada, $Pre(.,t) \leq M_k$.

Definição 8 (Disparo de transição)**:** Se t é uma transição habilitada por uma marcação M_{k-1} e o tempo em que t permaneceu habilitada está dentro do intervalo de tempo estático $e(t)$, então t *pode* disparar. Se o tempo de habilitação de t alcançar o limite superior do intervalo $e(t)$, então t *deve* disparar. O *disparo* de t muda o estado da rede levando a uma nova marcação M_k dada por $M_k = M_{k-1} + Pos(.,t) - Pre(.,t)$

O disparo de uma transição t na rede, provoca, além de uma nova marcação, a geração de uma nova classe c_k no grafo S a partir da classe c_{k-1}, denotado por: $c_{k-1}[t\rangle c_k$. Dizemos que uma classe c_{k-1} é imediatamente anterior à c_k quando t dispara em c_{k-1} e gera a classe c_k.

Definição 9 (Grafo de classes de estados)**:** O *grafo de classes de estados* é um grafo dirigido $S = (C, A)$ onde cada nó $c \in C$ é uma classe de estados e cada arco $a \in A$ conecta uma classe c_{k-1}, no nível $k-1$, à sua classe imediatamente posterior c_k, no nível k. Cada arco a é rotulado com a transição t, cujo disparo levou a rede da classe c_{k-1} para a classe c_k. O *nó raiz* do grafo de classes é o nó c_0 que representa a marcação inicial M_0.

Definição 10 (Transição persistente)**:** Uma transição habilitada t numa classe c_k é também dita *persistente* em c_k se t estava habilitada em uma classe c_{k-1}, imediatamente anterior a c_k, e t não disparou em c_{k-1}.

Definição 11 (Transição recém-habilitada)**:** Uma transição t, habilitada em uma classe c_k, será considerada *recém-habilitada* em c_k se t satisfizer uma das condições:

i) t não estava habilitada na classe c_{k-1};

ii) o disparo de t originou a classe c_k, isto é, t disparou na classe c_{k-1} e foi reabilitada em c_k.

Definição 12 (Sequência de disparos)**:** Seja t uma transição habilitada na marcação M_k. Caso o disparo de t seja sucedido pelo disparo de uma ou mais transições levando a rede a uma nova marcação M_j diz-se que os disparos das transições formam uma *sequência de disparos s* precedida por M_k e sucedida por M_j, denotado por: $M_k[s\rangle M_j$.

Uma sequência s de disparos se reflete como um caminho no grafo de classes. O disparo sucessivo de n transições em uma RdP-T de uma classe c_k para outra classe c_{k+n} é representado por $c_k[s\rangle c_{k+n}$.

3.3 Técnica de Análise de Tempo Global

Segundo a análise por tempo global, dois tipos de informação temporal estão presentes em cada classe de estado: informação temporal relativa e informação temporal global da classe. No domínio relativo, têm-se informações temporais a respeito da classe, utilizadas para fazer comparações entre as transições habilitadas na classe. O domínio global acumula informações temporais desde a marcação inicial até a classe em questão, ou seja, possui informações temporais que levam em consideração todos os disparos que ocorreram entre a classe inicial e a classe em questão (MATTAR JUNIOR *et al.*, 2007; LIMA, 2007).

A inclusão do domínio global nas classes de estado aumenta o poder de representatividade do grafo, o que pode viabilizar a modelagem e análise de sistemas em que se pretende, entre outras coisas, verificar o tempo de duração de roteiros de comportamento.

A técnica de análise tempo global (TG) se propõe a diminuir a imprecisão do cálculo do acúmulo temporal, quando comparado ao cálculo temporal relativo, principalmente quando a rede apresenta situações de paralelismo e concorrência entre transições (LIMA, 2007).

A análise de tempo global consiste da criação de um grafo de classes baseado em RdP-T que contém intervalos temporais associados às transições.

Nesta técnica, o grafo de classes é um grafo de alcançabilidade com informações temporais associadas às classes, baseado no grafo de classes de (BERTHOMIEU; MENASCHE, 1982; BERTHOMIEU; DIAZ, 1991). Os nós são as classes e os arcos são as transições disparadas para alcançar uma nova classe, ou nó. Os cálculos das informações temporais existentes nas classes são todos baseados em álgebra intervalar.

As informações temporais das classes são calculadas a partir da execução da RdP e incluem, além das informações de tempo relativo à classe, como no trabalho de Berthomieu *et. al.* (BERTHOMIEU; DIAZ, 1991), informações de tempo absoluto, ou global.

Para o cálculo de tempo global dois ajustes são realizados: 1) aplicação de um coeficiente de ajuste baseado na persistência ou não de transições habilitadas e 2) um ajuste dos tempos máximos dos intervalos de transições habilitadas para que continuem disparáveis.

A definição do coeficiente de ajuste de persistência considera as características intrínsecas às RdP-T e é utilizado para garantir que não haja imprecisão pela simples soma dos intervalos desde o início da execução da rede.

Na análise de tempo global, quando a questão é decidir quais relógios de transição devem ser zerados quando do disparo de uma transição, é considerada a semântica intermediária para RdP-T, baseada em (BERTHOMIEU; DIAZ, 1991). O ponto chave nestas semânticas é definir quando uma transição é recém-habilitada e seu relógio deve ser zerado (LIME; ROUX, 2009). Quando a questão é a disparabilidade das transições, a análise de tempo global realiza o ajuste de tempos máximos dos intervalos de transições habilitadas, para garantir que todas as transições disparáveis serão disparadas. Entretanto, nem todas as transições habilitadas são disparáveis e definimos a noção de transições disparáveis, bem como os dois ajustes citados na seção 3.3.1.

3.3.1 Informações Temporais

O conjunto de informação temporal W_k, da classe c_k, é composto por dois tipos de informação: de domínio relativo e de domínio global. No primeiro tipo, o tempo é acumulado desde o momento em que a transição que leva à classe c_k é habilitada até seu disparo. No segundo tipo, o acúmulo do tempo se dá desde o estado inicial da rede com a marcação inicial até a classe c_k (LIMA; LÜDERS; KÜNZLE, 2008).

As definições a seguir seguem os trabalhos de Mattar Jr. et al. (MATTAR JUNIOR *et al.*, 2007) e Lima (LIMA, 2007).

Definição 13 (Intervalo de tempo relativo)**:** Sendo t_f a transição disparada, o *intervalo de tempo relativo* $r_k(t_i)$ de uma transição $t_i \in T$ é calculado em uma classe c_k, tal que $c_{k-1}[t_f\rangle c_k$, é definido como:

$$r_k(t_i) = \begin{cases} e(t_i) & \text{caso 1,} \\ r_{k-1}(t_i) - r_{k-1}(t_f) & \text{caso 2,} \end{cases}$$

onde os casos 1 e 2 são respectivamente:

1. t_i é recém-habilitada em c_k;
2. t_i é persistente em c_k.

O intervalo de tempo relativo $r_k(t_i)$ é usado para identificar quais transições são disparáveis na classe c_k.

Definição 14 (Transição disparável)**:** Uma transição t_f com $r_k(t_f) = [a_f, b_f]$, é *disparável* em c_k se, e somente se, t_f está habilitada em c_k e não há outra transição t_i, com $r_k(t_i) = [a_i, b_i]$ habilitada em c_k, tal que $b_i < a_f$.

Definição 15 (Coeficiente de persistência)**:** O *coeficiente de ajuste de persistência* $ac_k(t_i)$ de uma transição habilitada t_i, em uma classe c_k tal que $c_{k-1}[t_f\rangle c_k$, é:

$$ac_k(t_i) = \begin{cases} r_{k-1}(t_i) \ominus r_{k-1}(t_f) & \text{caso 1,} \\ ac_{k-1}(t_i) \ominus r_{k-1}(t_f) & \text{caso 2,} \\ r_{k-1}(t_i) \ominus ac_{k-1}(t_f) & \text{caso 3,} \\ ac_{k-1}(t_i) \ominus ac_{k-1}(t_f) & \text{caso 4,} \end{cases}$$

onde os casos 1 a 4 são, respectivamente:

1. t_i e t_f são recém-habilitadas em c_{k-1};
2. t_i é persistente em c_{k-1} e t_f é recém-habilitada em c_{k-1};
3. t_i é recém-habilitada em c_{k-1} e t_f é persistente em c_{k-1};
4. t_i e t_f são persistentes em c_{k-1}.

O coeficiente $ac_k(t_i)$ é usado para evitar o aumento de imprecisão no cálculo dos tempos globais da rede.

Definição 16 (Intervalo de tempo global)**:** O *intervalo de tempo global* $g_k(t_i)$ de uma transição t_i disparável em uma classe c_k tal que $c_{k-1}[t_f\rangle c_k$ é:

$$g_k(t_i) = \begin{cases} e(t_i) & \text{caso 1,} \\ g_{k-1}(t_f) + r_k(t_i) & \text{caso 2,} \\ g_{k-1}(t_f) + ac_k(t_i) & \text{caso 3,} \end{cases}$$

onde os casos 1 a 3 são, respectivamente:

1. $k = 0$, i.e. c_k é a classe inicial;
2. $k \neq 0$ e t_i é recém-habilitada em c_k;
3. $k \neq 0$ e t_i é persistente em c_k.

O intervalo de tempo global $g_k(t_i)$ é um intervalo temporal contado desde a marcação inicial até o instante do disparo de t_i na classe c_k.

Definição 17 (Ajuste do limite superior)**:** Seja t_f uma transição disparada na classe c_k tal que $c_k[t_f\rangle c_{k+1}$. O limite superior do intervalo de tempo global $g_k(t_f)$ de t_f, deve ser ajustado pelo menor limite superior dos intervalos $g_k(t_i)$ de todas as transições t_i disparáveis na classe c_k, gerando um novo $g_k(t_f)$, tal que:

$$g_k(t_f) = [a_f, b],$$

onde a_f, $g_k(t_f) = [a_f, b_f]$, e $b = min\{b_i \mid g_k(t_i) = [a_i, b_i], \forall t_i \text{ disparável em } c_k\}$.

Definição 18 (Informações temporais da classe)**:** O *conjunto de informações temporais* de uma classe $c_k = (M_k, W_k)$ é dado por $W_k = (H, P, F, R, G)$, onde:

- H é o conjunto das transições habilitadas em c_k;
- P é uma tupla $\langle P_0, P_1, P_2\rangle$ que contém a informação de persistência em c_k, onde: P_0 é o conjunto de transições recém-habilitadas em c_k; P_1 é o conjunto de transições persistentes em c_k e recém-habilitadas em c_{k-1}; e P_2 é o conjunto de transições persistentes em c_k e em c_{k-1};
- F é o conjunto das transições disparáveis em c_k;
- R é o conjunto de intervalos de tempo relativo $r_k(t_i)$ de todas as transições habilitadas em c_k.
- G é uma tupla ($ac_k(t_i)$,$g_k(t_i)$) de cada transição disparável em c_k.

Para a implementação da técnica TG, desenvolvemos um algoritmo que constrói o grafo de classes, calculando suas informações temporais. A partir do grafo de classes pronto é possível extrair diretamente algumas informações que são usadas na análise da rede.

3.3.2 Construção do Grafo de Classes

A seguir propomos um algoritmo que constrói um grafo S de classe de estados de acordo com a técnica TG para uma rede RdP-T segura.

Dados de entrada

- a estrutura da rede: uma matriz de incidência $Pre = [pre(p_i,t_i)]_{m\times n}$ e uma matriz de incidência $Pos = [pos(p_i,t_i)]_{m\times n}$;
- o vetor I_e de tempo estático $e(t_i)$, $I_e = [(t_1,e(t_1)),(t_2,e(t_2)),...,(t_n,e(t_n))]^t$;
- o vetor M_0 de marcação inicial $M_0 = [marc(p_1),marc(p_2),...,marc(p_m)]^t$, onde $marc(p_i)$ é um inteiro não negativo que representa o número de marcas no lugar p_i.

Corpo do algoritmo

p0) definir $k = 0$;

Iniciando a construção da classe c_k:

p1) determinar a marcação da rede M_k: se $k = 0$, $M_k \leftarrow M_0$; senão M_k já foi determinada pelo disparo da transição t_f no passo p7 do ciclo anterior do algoritmo;

p2) determinar o conjunto H das transições habilitadas t_i a partir de M_k, de acordo com a definição 7;

p3) determinar a tupla $\langle P_0,P_1,P_2\rangle$;

p4) calcular o conjunto R dos intervalos de tempo relativo $r_k(t_i)$ $\forall t_i \in H$, de acordo com a definição 13;

p5) determinar o conjunto F das transições disparáveis t_i comparando elementos de R, de acordo com a definição 14;

p6) determinar a tupla G, calculando: a) $ac_k(t_i)$, $\forall t_i \in \{P_1 \cup P_2\}$, de acordo com a definição 15, e b) $g_k(t_i)$, $\forall t_i \in F$, de acordo com a definição 16;

Disparo de todas as transições $t_i \in F$ da classe c_k:

p7) $\forall t_f \in F$ disparar t_f: a) ajustar o intervalo $\overline{g}_k(t_f)$, de acordo com a definição 17; b) criar classes sucessoras com nível $k+1$; c) atualizar a nova marcação M_{k+1}; e d) atribuir o novo tempo global da classe c_{k+1} $g_{k+1} = g_k(t_i)$;

p8) incrementar $k \leftarrow k+1$ e $\forall t_f \in F$ e executar os passos p1 a p8;

Fim do algoritmo: O algoritmo é executado até que todas as transições em todas as classes tenham sido disparadas.

3.3.3 Extração das Informações do Grafo de Classes

É possível extrair, a partir do grafo de classes construído pelo algoritmo mostrado na seção 3.3.2, algumas informações que são úteis para a análise da RdP-T.

Tempo Total de uma Sequência de Disparos

O tempo global de uma sequência de disparos s de $c_0[s > c_k$, é encontrado na construção do grafo de classes e é o resultado do intervalo de tempo global $g_{k-1}(t_f)$, sendo t_f a última transição disparada para alcançar a classe c_k, que é, $c_{k-1}[t_f > c_k$.

Tempo de Permanência em um Estado

O tempo de permanência em uma classe refere-se ao tempo mínimo e máximo que o sistema, representado pela RdP-T, mantém-se no estado representado pela classe.

O tempo de permanência em uma marcação refere-se ao tempo mínimo e máximo que o sistema, representado pela RdP-T, mantém-se no estado representado pela marcação. O cálculo de permanência em uma classe é útil quando se deseja obter o tempo de um estado, representado por uma marcação, e se sabe qual a sequência de transições que levou a rede até esta marcação.

Uma marcação pode ocorrer em mais de uma classe. Então, é possível que sejam obtidos diferentes tempos de permanência para a mesma marcação, um para cada classe em que a marcação ocorre. O cálculo de permanência em uma marcação é útil quando se deseja o tempo em um estado, representado por uma marcação, mas não se sabe qual a sequência de transições que levou a rede até esta marcação.

Definição 19 (Tempo de permanência em uma classe)**:** Considerando que o disparo de todas as transições disparáveis é obrigatório, o tempo de permanência de uma rede em uma classe

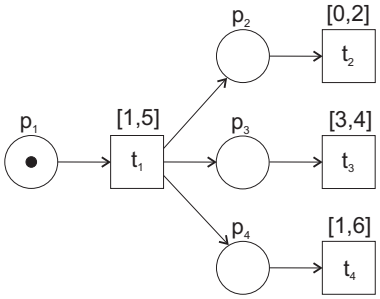

Figura 2: Exemplo de uma RdP-T com transições concorrentes.

alcançável c_k é dado por:

$$ic_{c_k} = [\overline{a}, \overline{b}]$$

onde: $\overline{a} = min\{a_i \mid r_k(t_i) = [a_i, b_i]\}$ e $\overline{b} = min\{b_i \mid r_k(t_i) = [a_i, b_i]\}$, $\forall t_i$ disparável em c_k.

Definição 20 (Tempo de permanência em uma marcação)**:** O tempo de permanência de uma rede em uma marcação alcançável M_x considera todos os tempos de permanência nas classes que possuem a marcação M_x e é dado por:

$$im_{M_x} = [\alpha, \beta]$$

onde: $\alpha = min\{\overline{a} \mid ic_{c_k} = [\overline{a}, \overline{b}]\}$ e $\beta = max\{\overline{b} \mid ic_{c_k} = [\overline{a}, \overline{b}]\}$, $\forall c_k$ que possua a marcação M_k.

Esta definição pode ser estendida para uma submarcação $M_y \subseteq M_x$.

3.3.4 Exemplos

Geração do Grafo de Classes

A figura 3 apresenta o grafo de classes baseado na técnica TG referente à RdP-T da figura 2. Cada nó do grafo representa uma classe $C_{k\_n} \in C$, cada arco uma transição disparada na classe C_{k_n} e que gera a classe sucessora $C_{(k+1)\_n}$. A figura 3 mostra cada nó com um cabeçalho separado por vírgulas: o nome da classe, C_{k_n}, onde k é o nível e n é um número identificador da classe; o vetor de marcação da classe M_k; e o tempo global da classe. As próximas linhas do nó representam a informação de disparo de uma transição t_i que será usada para a geração da próxima classe no caso de t_i ser disparada: o nome da transição t_i; seu tempo relativo $r_k(t_i)$; seu coeficiente de ajuste $ac_k(t_i)$; e seu tempo global $g_k(t_i)$.

Para melhor ilustrar a construção das classes e os cálculos de intervalos da técnica TG, ex-

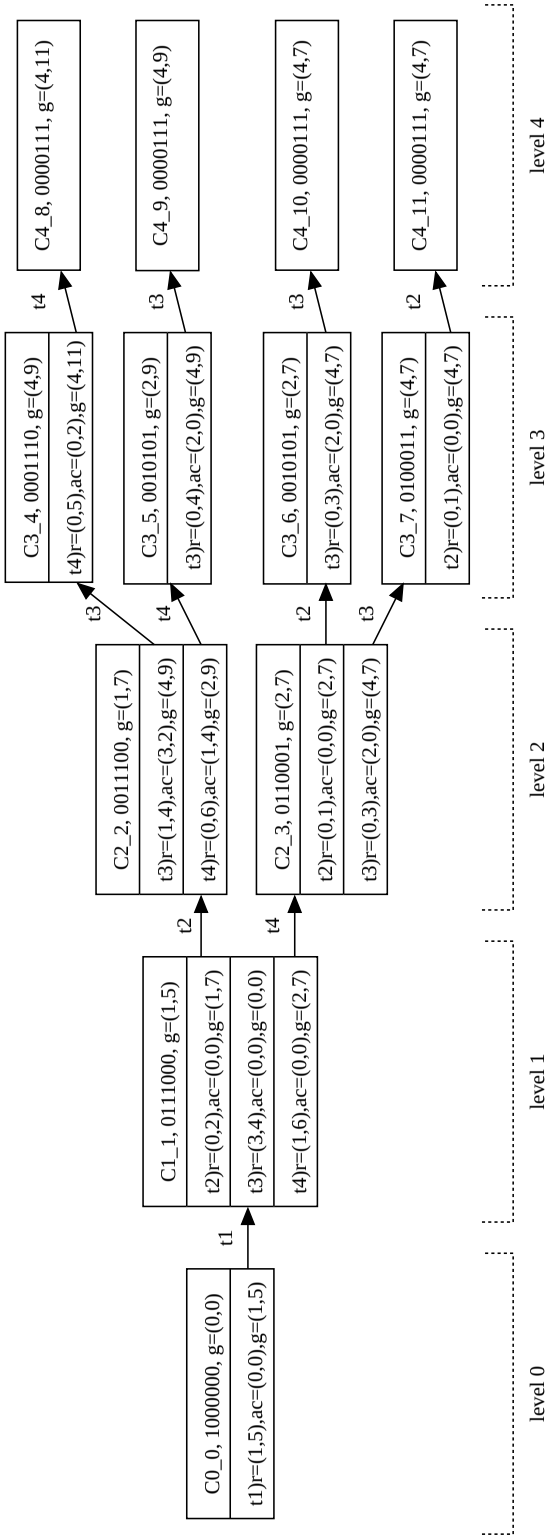

Figura 3: Classes de estado baseadas na técnica TG da rede de Petri da figura 2.

plicaremos a construção da classe C_{2_2}, considerando que t_2 foi disparada na classe C_{1_1}.

Início da construção da classe C_{2_2}

p1) determinar a marcação corrente da rede M_{2_2}: M_{2_2} já está determinada pelo disparo da transição t_2 na classe C_{1_1}, que é $M_{2\_2} = [0111000] + [0000100] - [0100000] = [0011100]$

p2) determinar o conjunto H de transições habilitadas: $H = \{t_3, t_4\}$.

p3) determinar $\langle P_0, P_1, P_2 \rangle$: $P_0 = \{\}$, $P_1 = \{t_3, t_4\}$ e $P_2 = \{\}$.

p4) calcular o conjunto R de intervalos de tempo relativo $r_{2\_2}(t_i)$ das transições habilitadas t_3 and t_4: $r_{2\_2}(t_3) = r_{1\_1}(t_3) - r_{1\_1}(t_2) = [3,4] - [0,2] = [1,4]$ e $r_{2\_2}(t_4) = r_{1\_1}(t_4) - r_{1\_1}(t_2) = [1,6] - [0,2] = [0,6]$

p5) determinar o conjunto F de transições disparáveis: $F = \{t_3, t_4\}$.

p6) determinar o conjunto G, calculando:

a) o coeficiente de ajuste das transições persistentes: $ac_{2\_2}(t_i)$, para $\{\forall t_i \in t_3, t_4\}$, Assim: $ac_{2\_2}(t_3) = r_{1\_1}(t_3) \ominus r_{1\_1}(t_2) = [3,4] \ominus [0,2] = [3,2]$ e $ac_{2\_2}(t_4) = r_{1\_1}(t_4) \ominus r_{1\_1}(t_2) = [1,6] \ominus [0,2] = [1,4]$;

b) intervalo de tempo global $g_{2\_2}(t_i)$ para $\forall t_i \in \{t_3, t_4\}$. Assim: $g_{2\_2}(t_3) = g_{1\_1}(t_2) + ac_{2\_2}(t_3) = [1,7] + [3,2] = [4,9]$ e $g_{2\_2}(t_4) = g_{1\_1}(t_2) + ac_{2\_2}(t_4) = [1,7] + [1,4] = [2,11]$

Disparo de $\{t_3, t_4\} \in F$ a partir da classe C_{2_2}:

p7) disparar t_3 e t_4: a) ajustar intervalos $\overline{g}_{2\_2}(t_3)$ e $\overline{g}_{2\_2}(t_4)$ pelo menor limite superior, resultando em $g_{2\_2}(t_3) = [4,9]$ e $g_{2\_2}(t_4) = [2,9]$; b) criar novas classes sucessoras C_{3_4} e C_{3_5}; c) atualizar M_{3_4} e M_{3_5}; e d) atribuir o novo tempo global das novas classes C_{3_4} e C_{3_5} respectivamente $g_{3\_4} = g_{2\_2}(t_3) = [4,9]$ e $g_{3\_5} = g_{2\_2}(t_4) = [2,9]$;

p8) $k \leftarrow 3$ e executar os passos p1 a p8 para a classe C_{3_4} com $t_f = t_3$ e C_{3_5} com $t_f = t_4$.

Tempo Total de uma Sequência de Disparos

De acordo com a definição 3.3.3, o tempo total para uma sequência é o intervalo de tempo global $g_{k-1}(t_f)$. Para este exemplo e considerando o disparo da sequência t_1, t_2, t_3, t_4 o tempo global é $g_3(t4) = [4,11]$. Esta informação é encontrada diretamente no grafo de classes, neste caso na classe C_{4_8}.

Tempo de Permanência em um Estado

De acordo com a seção 3.3.3, tem-se dois tipos observação de permanência em um estado. A permanência na classe e a permanência na marcação.

Os tempos de permanência nas classes C_{1_1}, C_{2_2} e C_{2_3} são, respectivamente, $[0,6]$, $[0,4]$ e $[0,1]$. Os tempos de permanência nas classes C_{3_5} e C_{3_6} são respectivamente $[0,4]$ e $[0,3]$. Todos estes tempos são encontrados de acordo com a definição 19.

O tempo de permanência na marcação 0010101, marcação das classes C_{3_5} e C_{3_6}, é $[0,4]$ encontrado de acordo com a definição 20.

Discussão

Considerando o exemplo dado, as transições t_2, t_3 e t_4 na figura 2, são concorrentes e habilitadas no mesmo instante depois o disparo de t_1.

Iniciando na classe C_{0_0} e assumindo o disparo das transições t_1 e depois t_2, obtém-se a classe C_{2_2} apresentada na figura 3. Se t_2 disparou instantaneamente na classe C_{1_1}, t_3 se mantém habilitada sem disparar por 3 unidades de tempo na classe C_{2_2}. De outra maneira, se t_2 disparou 2 unidades de tempo depois de sua habilitação em C_{1_1}, t_3 pode ainda parmanecer habilitada sem disparar por 2 unidades temporais na classe C_{2_2}. Então, na classe C_{2_2}, o coeficiente de ajuste de t_3 é dado por $ac_{2\_2}(t_3) = [3,4] \ominus [0,2] = [3,2]$, um intervalo impróprio. O coeficiente $ac_{2\_2}(t_3)$ significa que t_3 adiciona 3 unidades de tempo ao limite inferior e 2 unidades de tempo ao limite superior dos intervalos de tempo global.

Na figura 2, observamos que existirá um aumento na amplitude do intervalo, e consequentemente da imprecisão, se os intervalos de tempo estático $e(t_i)$ das sequências de disparo forem simplesmente somados para encontrar o tempo total da sequência, como em $e(t_1)+e(t_2) = [1,7]$, $e(t_1)+e(t_3) = [4,9]$ e $e(t_1)+e(t_4) = [2,11]$. Além da imprecisão, a existência de interseções entre os intervalos $[1,7],[4,9]$ e $[2,11]$ pode sugerir que t_3 após o disparo de t_1 seja disparável, o que não é verdade, porque quando t_3 puder disparar ($e(t_3) = [3,4]$), t_2 já terá disparado($e(t_2) = [0,2]$). Por estes motivos, devem ser feitos ajustes usando um coeficiente sobre o intervalo de tempo de disparo das transições que são persistentes na classe. Desta maneira, o intervalo de tempo global da sequência de disparos t_1,t_2,t_3,t_4, por exemplo, é dado por $r_{0\_0}(t_1)+r_{1\_1}(t_2)+ac_{2\_2}(t_3)+ac_{3\_4}(t_4) = [1,5]+[0,2]+[3,2]+[0,2] = [4,11]$, que é o intervalo de tempo de disparo calculado para a transição t_4 no domínio global da classe C_{3_4}.

Os problemas abordados nesta discussão foram os motivadores para o uso do coeficiente de ajuste *ac*, que garante o intervalo de tempo global exato para uma sequência de disparos que acumula tempos em uma RdP-T, conforme demonstrado em (MATTAR JUNIOR, 2008).

3.3.5 Teorema do Tempo Global

Nesta seção a definição 21 introduz o conceito de tempo concorrente calculado para uma sequência de disparos com transições em paralelo. Este conceito é associado ao ajuste de tempo calculado baseado em transições habilitadas simultaneamente.

O teorema 1, apresentado inicialmente em (MATTAR JUNIOR, 2008) e melhor estruturado a seguir, mostra que o coeficiente de ajuste de persistência *ac* calculado pela técnica TG é equivalente ao coeficiente de ajuste de persistência apresentado na definição 21. Apresentamos tal teorema por sua importância em mostrar que a técnica TG, mesmo quando da análise de sistemas com forte concorrência (o que implica em grande interlaçamento de eventos concorrentes), encontra um intervalo temporal sem acréscimo de imprecisão.

Definição 21: Sejam t_i e t_j duas transições em paralelo em uma classe c_k, e o disparo de t_i precede o disparo de t_j. O tempo corrente da sequência $t_i t_j$ é dado por:

$$\begin{cases} [a_i, b_i] + [a, b], & \text{se } b_i \leq b_j \\ [a_i, b_j] + [a, b], & \text{se } b_i > b_j \end{cases}$$

Sendo $[a_i, b_i]$ e $[a_j, b_j]$ o intervalo de tempo global de t_i e t_j respectivamente calculado na classe c_k; e o intervalo $[a, b]$ o coeficiente de ajuste do tempo $ac_k(t_j)$ de t_j calculado na classe c_k.

Allen (ALLEN, 1983) classifica treze relações mutuamente exclusivas entre duas possíveis ocorrências de eventos associados com intervalos de tempo. Estas relações que compreendem todas as possibilidades de relacionamentos entre os intervalos temporais, são explicitadas na tabela 3.3.5. Nota-se que as relações inversas, ou a troca de ordem entre X e Y, devem ser consideradas, excetuando-se a primeira relação, para que se verifiquem os treze casos possíveis.

Teorema 1: Sejam t_i e t_j duas transições em paralelo em uma classe c_k. O tempo global da sequência $s = t_i t_j$, obtido de acordo com a definição 21, representa o intervalo de tempo em que a sequência de disparos s pode ocorrer. Os valores obtidos nos limites inferiores e superiores desse intervalo são os menores e maiores tempos, respectivamente, em que o disparo da sequência pode ocorrer.

Tabela 1: As possíveis relações entre dois eventos temporais X e Y

Relação	Exemplo
X é igual a Y	XXX YYY
X ocorre antes de Y	XXX YYY
Y ocorre imediatamente após X	XXXYYY
X sobrepõe parcialmente Y	XXX YYY
X ocorre durante Y	XXX YYYYYYY
X começa com Y	XXX YYYYYYY
X termina com Y	XXX YYYYYYY

Tabela 2: Casos a serem estudados

$x_i < x_j$ $y_i < y_j$	$x_i < x_j$ $y_i = y_j$	$x_i < x_j$ $y_j < y_i$
$x_i = x_j$ $y_i < y_j$	$x_i = x_j$ $y_i = y_j$	$x_j = x_i$ $y_j < y_i$
$x_j < x_i$ $y_i < y_j$	$x_j < x_i$ $y_i = y_j$	$x_j < x_i$ $y_j < y_i$

Na demonstração a seguir é necessário considerar somente os nove casos listados na tabela 2 para considerar todas as diferentes possibilidades de ocorrência dos eventos "disparo de t_i e t_j". Neste texto $[x_i, y_i]$ e $[x_j, y_j]$ são respectivamente os intervalos de tempo global de t_i e t_j calculados na classe c_k.

Demonstração: Pretende-se mostrar que o coeficiente de ajuste de tempo $ac(t_j)$ calculado para t_j na classe c_k é o ajuste de tempo exato para que o resultado obtido no cálculo do tempo global seja coincidente com o tempo real do sistema. Existem quatro possibilidades que consideram o disparo de t_i anterior ao disparo de t_j, devido à sequência $s = t_i t_j$.

$x_i \leq x_j$ **and** $y_i \leq y_j$ Como t_i pode disparar em qualquer instante contido no intervalo $[x_i, y_i]$, deve-se somar um intervalo $[a, b]$ a $[x_i, y_i]$ para se obter o tempo global da sequência s, que necessariamente deve ser $[x_j, y_j]$. Assim, pode-se afirmar que $[x_i, y_i] + [a, b] = [x_j, y_j]$. Por outro lado, pode-se também obter o coeficiente de ajuste do tempo $ac(t_j)$ para t_j pela equação $[x_j, y_j] \ominus [x_i, y_i] = [a', b']$. Então, $x_i + a = x_j$, $y_i + b = y_j$, $x_j - x_i = a'$ e $y_j - y_i = b'$ e conclui-se que $a = a' \quad \text{e} \quad b = b'$.

$x_i > x_j$ **and** $y_i \leq y_j$ O intervalo de tempo da sequência é dado por $[x_i, y_j]$ e então deve-se obter $[a,b]$ tal que $[x_i,y_i]+[a,b]=[x_i,y_j]$. Por outro lado, o coeficiente de ajuste do tempo $ac(t_j)$ pode ser obtido por $[x_j,y_j] \ominus [x_i,y_i] = [a',b']$. Assim, $x_i + a = x_i$, $y_i + b = y_j$, $x_j - x_i = a' = 0$ e $y_j - y_i = b'$ e conclui-se que $a = a' = 0 \quad \text{e} \quad b = b'$.

$x_i \leq x_j$ **and** $y_i > y_j$ O disparo de t_i deve ocorrer dentro do intervalo $[x_i,y_j]$, uma vez que $y_i > y_j$, e o intervalo de tempo real para este caso é dado por $[x_j,y_j]$; portanto, deve-se obter $[a,b]$ tal que $[x_i,y_j]+[a,b]=[x_j,y_j]$ que resulta nas equações $x_i+a=x_j$ e $y_j+b=y_j$. Analogamente aos casos anteriores, o coeficiente de ajuste do tempo $ac(t_j)$ pode ser obtido por $[x_j,y_j] \ominus [x_i,y_i] = [a',b']$. A partir das equações $x_i + a = x_j$, $y_j + b = y_j$, $x_j - x_i = a' = 0$ e $y_j - y_i = b' = 0$ conclui-se que $a = a' \quad \text{e} \quad b = b' = 0$.

$x_i > x_j$ **and** $y_i > y_j$ O intervalo de tempo real é dado por $[x_i,y_j]$ e, analogamente ao caso anterior, t_i deverá disparar em $[x_i,y_j]$. Assim, pode-se obter as seguintes equações intervalares $[x_i,y_j]+[a,b]=[x_i,y_j]$ e $[x_j,y_j] \ominus [x_i,y_i] = [a',b']$ de onde pode-se concluir que $a = a' = 0 \quad \text{e} \quad b = b' = 0$. □

Desta forma, o incremento dado pelo coeficiente de ajuste do tempo *ac* ao intervalo de tempo global $g_k(t_j)$ de uma sequência de disparos em paralelo evita o aumento de imprecisão no cálculo dos tempos acumulados da rede. Isto se deve porque a técnica TG fornece exatamente o tempo real em que a sequência pode disparar, ou seja, seus limites inferiores e superiores são os tempos mínimo e máximo possíveis para o disparo da sequência.

3.4 Considerações do Capítulo

Além das definições básicas referentes à técnica de tempo global (TG), foram apresentados neste capítulo um algoritmo, um exemplo gerado por este algoritmo e uma discussão a respeito deste exemplo. A partir desta discussão percebe-se a importância de um teorema a respeito da precisão da técnica TG e seu coeficiente que chamamos de ajuste de persistência. O teorema mostra que a técnica de tempo global, devido a seu coeficiente de ajuste, é mais precisa que os métodos clássicos que realizam o acúmulo da informação relativa para totalizar as sequências de disparos através das classes do grafo.

O método clássico de análise para a construção de grafos de classes (BERTHOMIEU; MENASCHE, 1982; BERTHOMIEU; DIAZ, 1991) objetiva apresentar o comportamento dinâmico

de uma rede de Petri temporal.

Se por um lado o método clássico possui certa eficiência para a verificação de propriedades e análises temporais referentes a uma determinada classe e cenário que representam, por outro lado, não foram concebidos com a intenção de apresentar informações temporais referentes ao domíno global da rede, i.e., o encadeamento de eventos representados por sequências de disparos da rede.

O uso do relógio global, i.e. do relógio que acumula o tempo a partir do instante inicial da execução da rede, aparece em trabalhos que tratam o tempo de forma contínua, como em (YONEDA; SCHLINGLOFF, 1997) e (LILIUS, 1999), e em trabalhos que tratam o estado de forma compacta, como em (XU *et al.*, 2002), (WANG; DENG; XU, 2000) e (LIMA; LUDERS; KUNZLE, 2005). Segundo (LIMA, 2007), a abordagem da técnica de tempo global se difere destas nos seguintes pontos:

- A técnica TG associa o tempo absoluto, ou global, ao intervalo de disparo dinâmico de cada transição habilitada.
- Para a diminuição da imprecisão, a verificação da disparabilidade de cada transição habilitada é aplicada diferentemente para as transições persistentes e para as recém-habilitadas.
- O domínio dos intervalos de disparos dinâmicos pode utilizar o tempo absoluto ou relativo. Em nossa implementação utilizamos o tempo relativo para verificar a condição de disparabilidade das transições.
- Trata da múltipla habilitação das transições. Na construção do grafo de classes ajustamos intervalos de transições habilitadas em conjunto.

Para a utilização da técnica de tempo global, propomos a utilização de redes de Petri temporais lugar/transição k-limitadas com todos os arcos limitados a peso 1. Para as transições que se mantém habilitadas devido a existência de mais de uma marca em um lugar, consideramo-las persistentes.

O grafo de classes implementado é infinito para RdP-T cíclicas. Isto porque o cálculo do tempo global é infinito para uma rede cíclica. Entretanto, é possível estabelecer um dos seguintes critérios como critério de parada para a construção do grafo:

- limite temporal de execução alcançado;

- classe com mesma marcação já haver sido gerada anteriormente, como em um grafo de alcançabilidade clássico;
- sequência de disparos alcançado;
- ou, ainda, estruturais como número de níveis ou de classes do grafo.

4 Verificação por Tempo Global

Estima-se que 40% a 60% do esforço de desenvolvimento de sistemas embarcados são empregados em atividades de verificação e validação. Sistemas embarcados possuem fortes restrições de qualidade e confiabilidade. Seus projetos de desenvolvimento sofrem constantemente com a pressão do tempo-para-mercado e, portanto, a indústria busca a redução de prazos e custos, bem como o aumento da qualidade e da confiabilidade no processo e no produto de software. Entretanto, garantir alta qualidade enquanto mantém a produtividade é um grande desafio. Percebe-se maior necessidade de técnicas formais e automáticas incorporadas à verificação, em adição ao teste de software padrão, mesmo porque características específicas de software embarcado, como as análises de tempo real e de concorrência, requerem modelagens e simulações específicas (BAILEY, 2007) e (EBERT; JONES, 2009).

As atividades de verificação de software devem ser conduzidas complementarmente por atividades dinâmicas que exigem a execução do código e atividades estáticas que são baseadas na inspeção física de artefatos do software que não são necessariamente o código. A verificação dinâmica apenas pode ser realizada quando existir código e ela normalmente compreende as atividades de teste de software.

Este capítulo apresenta um método para a análise de restrições temporais baseada em modelos de redes de Petri e tempo global. Inicialmente apresentamos uma contextualização deste método em relação às técnicas de verificação estática de software embarcado que se utilizam de redes de Petri. Então apresentamos o método e sua estrutura, que, resumidamente, é um conjunto de processos que utilizam técnicas e ferramentas desenvolvidas na pesquisa descrita neste trabalho, ou mesmo adaptada de outros autores, com o objetivo de verificar as restrições temporais do software sendo construído utilizando a técnica de tempo global. Ao final do capítulo tecemos considerações a respeito deste método e sua aplicabilidade.

4.1 Verificação Estática de Software Embarcado

Atividades de verificação estática são baseadas no uso de métricas, simulação, análise de programas e de modelos e verificação formal (PRESSMAN, 2006). A verificação formal se baseia na conferência da corretude de especificações formais, por exemplo verificação de modelos, usando técnicas como SAT e OBDD, e inferência lógica, usando técnicas como provadores de teoremas.

A verificação de modelos (*Model Checking*) é uma técnica automática para a verificação da corretude de propriedades de sistemas reativos críticos ou sistemas críticos especificados por suas propriedades temporais. Uma propriedade temporal é um conjunto de comportamentos desejados no tempo. O sistema satisfaz a propriedade se cada execução do sistema pertence a este conjunto. Esta técnica também pode produzir um contraexemplo, i.e., um comportamento que viola os requisitos.

Para executar a verificação de modelos são necessários: uma linguagem de modelagem na qual o sistema pode ser descrito, uma linguagem de especificação para a formulação de propriedades e um algoritmo ou cálculo dedutivo para o processo de verificação. Normalmente o sistema é modelado como um grafo de transição de estados (finitos) e as propriedades são formuladas em uma lógica proposicional temporal adequada. Um procedimento eficiente de busca é então usado para determinar se o grafo de transição satisfaz ou não as fórmulas temporais (CLARKE; SCHLINGLOFF, 2001) e (CIMATTI *et al.*, 2002).

O maior impedimento para a aplicação totalmente automatizada de *Model Checking* foi a explosão de estados resultante. Entretanto, atualmente, com aproximadamente três décadas de pesquisas na área, o número de estados que podem ser explicitamente representados por estruturas de dados como listas ou tabelas *hash* é aproximadamente 10^6 representados por diagramas de decisão binária (BDDs) (MCMILLAN, 1993) é mais do que 10^{100}, o que é suficiente para representar grandes sistemas industriais. Estes resultados fazem com que a explosão de estados não seja atualmente um problema na verificação de modelos (CLARKE; SCHLINGLOFF, 2001).

RdP-T são usadas em diferentes aplicações na verificação de modelos para escalonamento e síntese de sistemas embarcados. Cortes et al. (CORTES; ELES; PENG, 2000) definem um método de modelagem e verificação de software embarcado usando PRES+, um tipo de RdP-T. Depois da modelagem, os autores propõem a realização de um conjunto de análises sobre as RdP como comportamentais, de alcançabilidade e temporais. Para a análise temporal, os autores propõem traduzir as redes PRES+ em autômatos híbridos e usar verificação simbólica de modelos

(CLARKE; GRUNBERG; PELED, 1999) para provar a corretude do sistema.

Lime and Roux (LIME; ROUX, 2003) propõem a modelagem usando uma rede de Petri temporal que possui uma dimensão de escalonamento, a SETPN. A proposta desses autores é modelar sistemas de tempo real, especialmente sistemas embarcados, para a análise de escalonabilidade. Estes autores provêm um método usando uma representação poliédrica e DBM (DILL, 1990). Em (LIME; ROUX, 2009), Lime e Roux usam os padrões de projeto do trabalho (LIME; ROUX, 2003) para modelar tarefas e implementar as políticas de escalonamento Prioridade Fixa e *Earliest Deadline First*.

Nos trabalhos de (CORTES; ELES; PENG, 2000) e (LIME; ROUX, 2009) as redes são traduzidas em grafos de estados que são automâtos lineares híbridos e propriedades temporais são verificadas usando a ferramenta de verificação de modelos Hytech (HENZINGER; HO; WONG-TOI, 1997).

A verificação de restrições temporais pode ainda ser realizada através da análise de programas ou da análise de modelos. No primeiro tipo, o objeto de observação é o programa e são realizadas atividades de análise, inspeção e revisão sobre o código. No segundo tipo, um modelo do software é construído e analisado com o objetivo de verificar se os requisitos temporais do software foram entendidos e se serão satisfeitos.

A análise por redes de Petri é um tipo de análise por modelos. Redes de Petri, por serem ao mesmo tempo um modelo gráfico e matemático, permitem modelar características do software de maneira a torná-las passíveis de serem avaliadas de uma maneira mais formal. A avaliação da representação do software embarcado por redes de Petri são feitas por análises estruturais, comportamentais e temporais. Na análise temporal por RdP é possível alcançar valores temporais de execução do sistema.

A análise temporal quantitativa, ou o cálculo de valores numéricos sobre o comportamento temporal do sistema, possui algumas vantagens. Entre elas está a possibilidade de se conhecer até que ponto o sistema realmente está fora do limite desejado. Isto é útil no entendimento do porquê uma restrição de sincronização quantitativa falha. Com a análise quantitativa, é possível não apenas avaliar as restrições temporais de um sistema, mas também entender seu comportamento e, em muitos casos, identificar otimizações de projeto. Mesmo informações mais detalhadas podem ser obtidas usando esta análise, que permite a observação de sequências de execuções específicas no modelo, ao invés da verificação do comportamento do total do sistema. Estes métodos podem determinar o tempo mínimo e máximo entre a ocorrência de eventos e o número de ocorrências de um evento em todos os caminhos entre dois outros eventos. A análise quantitativa temporal não

procura por verificação de propriedades específicas, ao invés disso, a análise provê informação que permite ao desenvolvedor a análise de diversas propriedades (CLARKE *et al.*, 2000).

Berthomieu *et al.* (BERTHOMIEU; MENASCHE, 1982), (BERTHOMIEU; MENASCHE, 1983), (BERTHOMIEU; VERNADAT, 2004) e (BERTHOMIEU; VERNADAT, 2006) apresentam uma análise quantitativa temporal, usando RdP e análise de tempo relativo baseado em grafo de classes. Esta é principal técnica de análise de RdP-T encontrada na literatura e é um método enumerativo para gerar o espaço de estado alcançável sendo baseado no comportamento de uma RdP-T.

Neste método, cada classe é um nó do grafo e agrupa o conjunto de estados com a mesma marcação. Cada intervalo de tempo agrupa instantes de disparo para cada transição habilitada na classe. Cada transição habilitada pode disparar uma vez na classe gerando outra classe em um nível posterior e este possível disparo é um arco do grafo conectando estas duas classes. Esta técnica também encontra o intervalo de tempo durante o qual o sistema permanece em cada classe, sendo portanto eficiente para verificar propriedades da rede e analisar restrições temporais relativas a uma determinada classe. No entanto, este método não apresenta informações temporais globais do comportamento da rede, ou seja, informações temporais explícitas referentes ao encadeamento de eventos representados por sequências de disparos de transições desde a marcação inicial da rede. Essa restrição impede a análise temporal de sistemas de tempo real que impõem um limite de tempo na execução de tarefas que consistem de sequências de eventos, tais como múltiplos fluxos de execução (*threads*) em software embarcado.

Para obter resultados mais precisos, Dill (DILL, 1990) descreve as estruturas de dados DBM (*Difference Bounds Matrix* - Matrizes de Limites de Diferença) e propõe um método para a verificação do grafo de estados de autômatos finitos temporais de sistemas concorrentes. Existem aplicações destas estruturas em associação com redes de Petri e verificação de modelos (CORTES; ELES; PENG, 2000), (LIME; ROUX, 2003) e (LIME; ROUX, 2009).

O método de Berthomieu *et al.* (BERTHOMIEU; MENASCHE, 1982) pode ser classificado como interativo e muitas vezes é baseado na simulação. Embora a simulação seja um método não exaustivo (pois é impossível prever todas as configurações possíveis) é possível ao menos testar o comportamento do sistema modelado para as configurações previstas (CARDOSO; VALETTE, 1997). Em alguns momentos a simulação pode ser mais difícil de utilizar, entretanto pode ser mais adaptável a mudanças que ocorram no sistema a ser produzido (CLARKE *et al.*, 2000).

Nesta seção apresentamos o contexto em que está inserido o método de análise por tempo global: no da verificação estática por análise quantitativa baseada em modelos.

4.2 O Método de Verificação por Tempo Global

O objetivo deste método é suportar a análise temporal, usando tempo global, do software embarcado construído com um núcleo que executa uma política de escalonamento dinâmico de prioridade fixa ou dinâmica. Sua aplicabilidade considera a classe dos sistemas embarcados que podem estar construídos em linguagens C ou C++. Os programas podem suportar múltiplos fluxos de execução (ou *threads*) em uma arquitetura monoprocessada.

A essência deste método está em permitir a análise das propriedades estruturais e comportamentais dos modelos em RdP, a análise temporal dos modelos usando tempo global, a análise de escalonabilidade e, consequentemente, a verificação temporal. Além disso, as diversas possibilidades de modelagem usando redes de Petri também estão observadas neste método, bem como o uso de padrões de projeto de redes de Petri (NAEDELE; JANNECK, 1998) para software embarcado, como proposto por (LIME; ROUX, 2003).

Uma visão geral das etapas propostas e suas interações para o método de verificação é apresentada na figura 4. Este método toma como entradas a especificação das tarefas, suas restrições temporais e sua arquitetura de comunicação e gera como saídas os resultados das análises estrutural, comportamental e temporal dos requisitos e ainda um modelo em RdP da interação entre as tarefas.

Como **primeira etapa**, tem-se a modelagem em RdP do sistema embarcado. Para a modelagem de sistemas por RdP são considerados dois elementos básicos: condições, modeladas por lugares, e eventos, modelados por transições. Transições podem representar diretamente atividades, código de programa ou chamadas a funções ou métodos permitindo que detalhes de especificação sejam granularmente maiores ou menores se necessário.

A **segunda etapa** envolve uma análise de algumas propriedades das RdP's que representam o software embarcado. Propõe-se a análise de bloqueio e alcançabilidade para a verificação da estrutura da RdP.

Após as análises de propriedades atemporais das RdP's, na **terceira etapa** propõe-se a associação de tempo às RdP's para futuras análises temporais da RdP. O princípio básico desta etapa está na associação dos tempos mínimo e máximo que representam o melhor caso e o pior caso de execução de um conjunto de atividades, se as RdP's representam partições de um projeto de software, e o melhor e pior caso de execução de um trecho de código, se as RdP's representam o código do software embarcado.

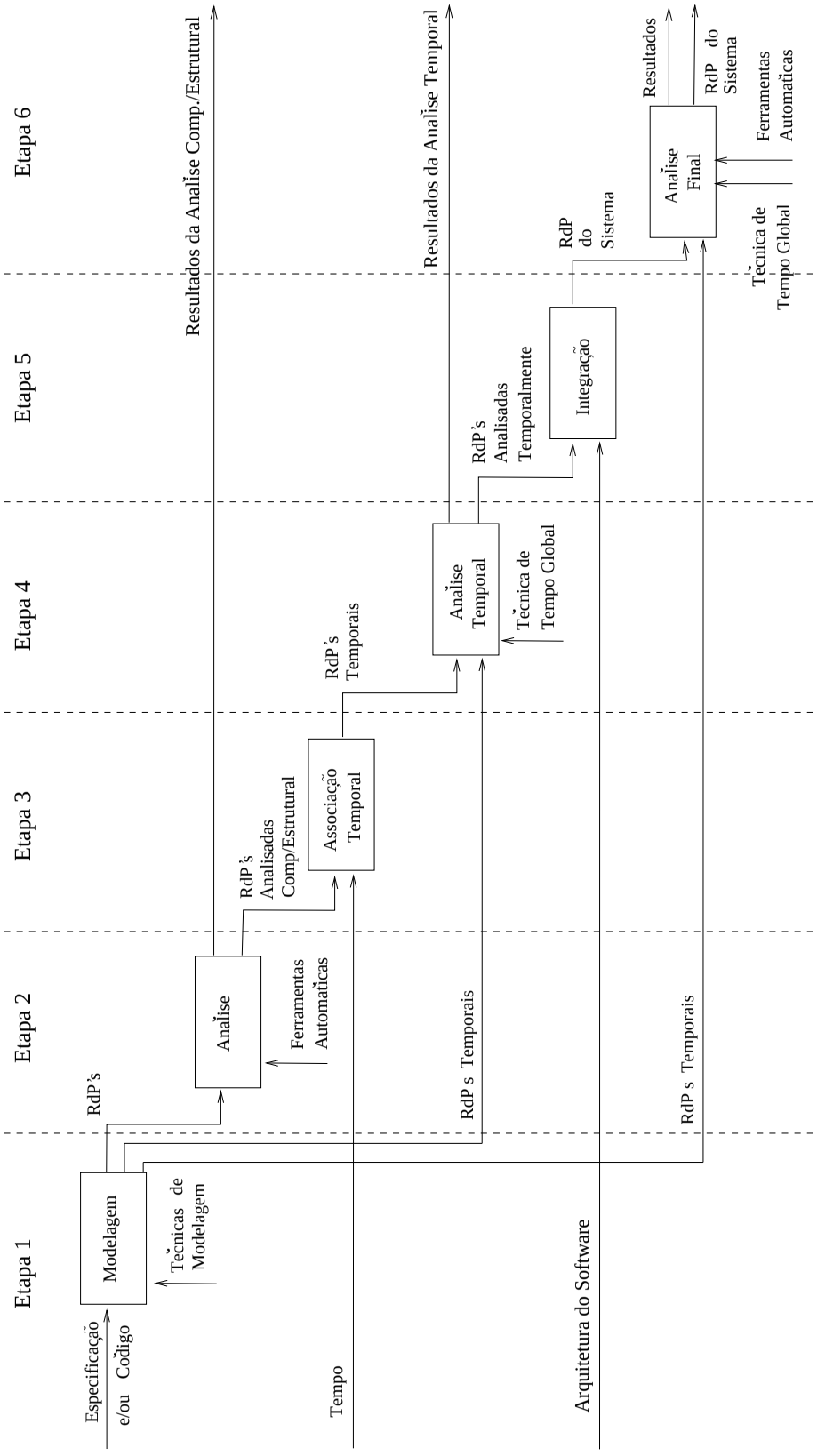

Figura 4: Visão geral do método

A **quarta etapa** compreende a análise temporal das RdP's geradas a partir das etapas anteriores. Nesta etapa se realizará a análise de cenários e roteiros com classes equivalentes para redes seguras, especialmente usando o método de análise de tempo global de Lima (LIMA, 2007). As RdP's analisadas nesta etapa representam módulos do sistema, funções ou métodos do código. Para realizar uma análise temporal sobre uma RdP que represente o sistema inteiramente é necessária uma integração destas redes.

Na **quinta etapa** se propõe esta integração. Na **sexta etapa** propõe-se uma nova análise temporal baseada na análise de tempo global de Lima (LIMA, 2007) sobre a única RdP que representa todo o sistema.

Cada uma dessas etapas é descrita em mais detalhes a seguir.

Etapa 1: Modelagem das RdP's

O modelo em RdP-T é naturalmente capaz de modelar diversas características do projeto do software embarcado monoprocessado, de tarefas que são:

- dependentes ou independentes;
- periódicas ou esporádicas;
- com prioridade fixa ou dinâmica;
- preemptivas ou não-preemptivas;
- com ou sem relações de concorrência, precedência, exclusão mútua e sincronizações.

Para uma redução do escopo de aplicação da técnica de tempo global, as redes de Petri geradas nesta etapa devem ser seguras, ou 1-limitadas.

Uma observação importante é que nosso método propõe a modelagem de redes de Petri que representam módulos do sistema de maneira separada, para que, nas etapas posteriores, as análises sejam realizadas sobre redes menores e depois integradas. Após a integração, ou hierarquização, as análises são retomadas considerando-se a integração dos módulos.

Para a modelagem de sistemas por RdP são considerados dois elementos básicos: condições, modeladas por lugares, e eventos, modelados por transições. Transições podem representar diretamente macroatividades, chamadas a funções ou métodos, ou mesmo trechos de código de programa, permitindo que detalhes de especificação sejam granularmente maiores ou menores

se necessário. Considerando tal visão, nesta etapa, informações de controle são modeladas por lugares e expressões de comandos, e processos ou macroatividades são modelados por transições, indo de acordo com os trabalhos (CORTES, 2001) e (DAHR *et al.*, 1994).

As especificações dadas como entrada para esta etapa podem estar descritas em modelos de projeto, em código fonte em alguma linguagem de programação, como C ou C++, ou ainda não estarem modeladas inicialmente. A seguir são apresentadas linhas gerais sobre a modelagem em RdP-T a partir destas especificações.

Modelagem Baseada em Especificações do Projeto Barreto *et al.* em (BARRETO; MACIEL; CAVALCANTE, 2003) e (BARRETO, 2005) apresentam uma metodologia para traduzir a especificação em uma RdP-T. Esta metodologia usa blocos de construção com o propósito de facilitar a composição das redes. Os blocos principais sugeridos por estes autores são: chegada de tarefas, com o cuidado de considerar que no início todas as tarefas chegam em seus *instantes críticos*, verificação de *deadline* e o bloco de estrutura da tarefa. A RdP-T gerada possui as propriedades necessárias para a utilização em nosso método. Mais detalhes podem ser encontrados nos trabalhos citados.

Em (LIME; ROUX, 2003), os autores apresentam um conjunto de RdP-T seguras que modelam alguns exemplos de serviços de núcleos (*executives*) em tempo real definidos em (OSEK/VDX, 2001). Foram modelados nestas redes, esquemas básicos de ativação para tarefas para protocolos de acesso a recursos complexos. Em (LIME; ROUX, 2009) os autores sugerem que essas redes podem ser usadas como padrões de projeto, mesmo que, obviamente, as possibilidades de modelagem não devam estar restritas às apresentadas no trabalho de (LIME; ROUX, 2003). No capítulo 5 usaremos as definições dessas redes e alguns exemplos para apresentar a aplicabilidade desta etapa de modelagem.

Modelagem Baseada em Modelos de Projeto Pesquisas em engenharia de software têm sido feitas com o objetivo de derivar redes de Petri a partir de modelos já existentes. Isto ocorre devido à grande utilidade em ter essas redes no desenvolvimento baseado em modelos (ou MDD, em inglês *Model-Driven Development*) (SELIC, 2003) ou mesmo na engenharia reversa (PRESSMAN, 2006).

O trabalho (SOARES; VRANCKEN, 2008) propõe a criação de RdP-T com arcos inibidores a partir de diagrama de sequência da UML 2.0 (STEVENS; WHITTLE; BOOCH, 2003), inclusive com os novos operadores desse diagrama. Outros trabalhos também propõem a criação

de redes de Petri a partir de diagramas de sequência, mas para redes coloridas ou orientadas a objetos, que não são utilizadas atualmente por nosso método.

No diagrama de atividades da UML, as atividades são baseadas na semântica de redes de Petri. Isto permite traduzir intuitivamente diagramas de atividades em notações de redes de Petri (TRICKOVIE, 2000), (GEHRKE; GOLTZ; WEHRHEIM, 1998) e (STAINES, 2008). Trabalhos propõem a tradução desses diagramas também para redes coloridas (STAINES, 2008) e (FAROOQ; LAM; LI, 2007), estocásticas (LóPEZ-GRAO; MERSEGUER; CAMPOS, 2004), objetos (SALDHANA; SHATZ, 2000), entre outras. O trabalho (DELATOUR; PALUDETTO, 1998) propõe a utilização deste diagrama adaptado para objetos da RdP sendo possível então construir uma RdP-T segura a partir de diagramas de atividades da UML.

Outros modelos podem ser usados como base para a tradução para redes de Petri. Cortadella *et. al.*, em (CORTADELLA *et al.*, 1998), propõem um método para derivar redes de Petri a partir de qualquer modelo de especificação que possa ser mapeado em uma representação baseada em estados com arcos rotulados com símbolos de um alfabeto de eventos, ou sistema de transição.

Modelagem Baseada no Código Fonte Para a tradução do código C em redes de Petri no contexto deste método, as atenções são dadas especialmente aos aspectos de fluxo de controle do software embarcado. Propriedades de computação, sincronização, comunicação e concorrência do sistema podem ser representadas por redes de Petri de uma maneira mais natural. Nesta etapa, o objetivo é gerar uma rede de Petri segura, ou mesmo FCPN (*Free-Choice Petri Nets*) ou CCPN(*Complex-Choice Petri Nets*) (SGROI *et al.*, 1999), que reflita o aspecto concorrente de um software embarcado, a partir de um determinado programa. A modelagem de redes de Petri a partir do código é baseada no controle de fluxo dos programas. A modelagem de fluxo de dados considera apenas as variáveis que impactam no fluxo de controle do programa. As variáveis que não afetam este fluxo não são modeladas no método proposto neste trabalho.

No contexto deste trabalho, foi estudada e proposta uma tradução de código nas linguagens C e C++ para redes de Petri seguras. Isto porque esta é uma linguagem de programação de propósito geral que oferece economia sintática, controle de fluxo, estruturas simples e um bom conjunto de operadores. É uma linguagem de alto nível apropriada para software embarcado tanto quanto por ser simples e ao mesmo tempo potente, como por ter um campo de aplicação ilimitado e com aprendizado rápido. Ainda, é uma linguagem estruturada e altamente portável, já que apresenta grande quantidade de bibliotecas e sistemas operacionais desenvolvidas para e sobre ela.

Tradução de Linha de Código

A seguir apresenta-se um conjunto de restrições associadas ao método proposto de tradução do código para redes de Petri.

1. Definições de variáveis e cabeçalhos de métodos e classes não são modelados na RdP, não gerando uma transição ou lugar que os represente;
2. o uso de classes na definição de objetos a serem utilizados pela *Thread* (no momento de definição de variáveis, por exemplo em *threadTime*) provocará a criação de uma RdP para cada método desta classe/objeto;
3. uma transição da RdP representa uma linha de comando;
4. lugares da RdP representarão o fluxo controle existente entre uma linha de comando e outra; e
5. trechos de código indisponíveis para análise devem ser representados como uma transição e devem ter todo o seu tempo tomado e associado a esta transição.

Mais detalhes da identificação de elementos do código em C ou C++ e associação a lugares ou transições para gerar uma RdP-T segura são apresentados no apêndice A.

Etapa 2: Análise Estrutural e Comportamental das RdP's

Nesta etapa propõe-se efetuar a análise não temporal das RdP's geradas na etapa anterior. Esta análise permite ao desenvolvedor verificar se a estrutura e comportamento, não temporal, dos modelos estão de acordo com as especificações.

Diversos tipos de análises de propriedades estruturais e comportamentais podem ser realizadas nesta etapa, como: alcançabilidade, limitação e segurança, vivacidade, reversibilidade, cobertura, persistência, limitação estrutural, conservação, repetitividade e consistência (MURATA, 1989).

As propriedades estruturais se baseiam na estrutura da rede e são facilmente implementáveis. As propriedades comportamentais baseiam-se na enumeração do espaço de estados, o que as inviabiliza na maior parte dos modelos de sistemas mundo real.

Dentre as propriedades comportamentais, sobretudo no caso de sistemas embarcados, adquirem importância a de alcançabilidade e as análises de bloqueio.

A análise de alcançabilidade verifica se um estado esperado do software é alcançável a partir de outro. Devem-se realizar verificações a este respeito porque, caso seja percebido que o software sendo analisado não possui o comportamento desejado, deve-se proceder a correções no seu desenvolvimento.

No caso da verificação da existência de bloqueios, em um sistema *multithread*, por exemplo, é necessário que os fluxos de execução liberem a CPU e não causem bloqueio no sistema. Para análise de bloqueios, é necessária a criação de um grafo de alcançabilidade. Entretanto, a criação deste grafo é exponencial, i.e. à medida que o tamanho da RdP aumenta, aumenta exponencialmente o tamanho do espaço de estados deste grafo de alcançabilidade. A comunidade acadêmica da área de RdP vem resolvendo este problema através da técnica de desdobramento (*unfolding*) (MCMILLAN, 1995) (ESPARZA; ROMER; VOGLER, 1996) (ESPARZA; HELJANKO, 2008). O trabalho (CORBETT, 1996) avalia alguns métodos para análise de bloqueio e o trabalho (MELZER; ROMER, 1997) propõe análise de bloqueio utilizando desdobramento.

Etapa 3: Associação do Tempo às RdP's

No caso de modelagem baseada em modelos de projeto, o tempo que será associado às transições das RdP's correspondem às restrições temporais constantes nos requisitos do projeto. Estas restrições temporais podem estar presentes no diagrama de sequências, de atividades ou mesmo nas descrições de casos de uso da UML (BARRETO, 2005).

No caso de uma modelagem baseada em código, é possível realizar o levantamento de tempos mínimo e máximo para cada trecho de código associado a uma transição. Estes tempos serão referentes ao melhor e pior caso de execução respectivamente. Esse levantamento de tempo será baseado em restrições, estimativas e medições do tempo de execução de trechos de código. Algumas observações são necessárias quanto a esta tomada de tempo:

- a instrumentação do código para medição de tempo será individual para cada expressão de comando; haverá a inclusão de um comando de medição imediatamente anterior à expressão e outro imediatamente posterior à mesma expressão;
- para RdP's hierárquicas que representem trechos de código binário que não tenham disponibilidade de código-fonte, o tempo de execução de todo este código indisponível corresponderá a uma transição específica na RdP de alto nível, com a inclusão de um comando de medição imediatamente anterior à expressão representando a chamada de método/função e outro imediatamente posterior à mesma expressão;

- mesmo com a definição da localização do ponto de medição, algumas questões de instrumentação ainda devem ser consideradas, como o impacto da intrusão da sondagem ou observação, o efeito-sonda e outras considerações presentes do trabalho de doutoramento de Cadamuro (CADAMURO, 2007).

Etapa 4: Análise Temporal das RdP's

Nesta etapa é realizada a aplicação da técnica de análise de tempo global de Lima (LIMA, 2007), apresentada no capítulo 3. Após a modelagem da RdP segura, temporal ou não, realizamos sua análise comportamental e estrutural por ferramentas já existentes e, se necessário, uma possível associação de tempo às transições. Agora é o momento de realizarmos sua análise temporal.

A partir de modelos de RdP-T que representem módulos do sistema, geramos um grafo de classes de alcançabilidade com tempos globais e relativos, de acordo com o método apresentado no capítulo 3. A partir desse grafo é construído um conjunto de sequências de disparos, de acordo com a escolha do desenvolvedor: alguma política de escalonamento, prioridade fixa ou primeiro *deadline*, ou todas as sequências possíveis possibilitando estudos do pior (*wcet*) e melhor caso. A definição da construção dessas sequências de disparos são detalhadas no capítulo 5.

A aplicação dessa técnica permite que se encontre uma resposta concreta sobre o valor temporal da duração de um cenário, permitindo estudar o tempo no melhor e pior caso e permitindo a análise de cenários e roteiros com classes equivalentes para redes seguras.

Utilizamos uma ferramenta que foi implementada para este trabalho e gera o grafo de classes, um conjunto de informações temporais a respeito dos estados do sistema e um conjunto de sequências de disparos de transições, que também será detalhada no capítulo 5.

Etapa 5: Construção da RdP Integrada/Hierárquica

A verificação completa do sistema impõe que as RdP's que representam módulos específicos sejam integradas em uma única rede que represente o software como um todo.

Existem duas maneiras de realizar tal tarefa: uma delas é integrando as redes por uma operação de adição, como no trabalho de Barreto (BARRETO, 2005), onde lugares que representam pontos de interface são substituídos pelas redes que realizam atividades para esta interface. Outra maneira consiste em criar uma rede hierárquica que corresponde a um nível de abstração

superior, como no trabalho de Cortes *et al.* (CORTES; ELES; PENG, 2001).

Na construção da RdP hierárquica, transições representam RdP relativas a atividades específicas e lugares representam as interfaces entre tais redes. É possível usar os diagramas de sequência e colaboração como base para a identificação da interação entre as classes e, consequentemente, entre as RdP's (GEHRKE; GOLTZ; WEHRHEIM, 1998). As restrições temporais da rede hierárquica são calculadas por operações de álgebra intervalar a partir dos intervalos temporais obtidos das redes de mais baixo nível.

Etapa 6: Análises Finais da RdP Integrada/Hierárquica

Após a integração ou hierarquização das RdP's, gera-se uma única rede que representa todo o sistema e sobre a qual devem ser reconduzidas as análises realizadas na Etapa 2. Isto porque não existe a garantia de que esta nova rede mantenha as propriedades diagnosticadas na etapa anterior. Portanto, como primeiro passo nesta etapa tem-se a recondução das análises realizadas na Etapa 2, agora sobre a RdP integrada/hierárquica.

No caso de uma rede integrada, i.e. uma RdP que é a união das redes menores que representam macroatividades, funções ou métodos, esta rede é grande e complexa mesmo para um sistema de tamanho médio. Este grande tamanho da RdP pode inviabilizar as análises baseadas em grafo de alcançabilidade que foram menos trabalhosas na Etapa 2, devido aos tamanhos menores das redes. Neste caso, durante a etapa anterior de modelagem (Etapa 1) deve-se modelar o sistema observando questões de granularidade, como as apresentadas no Apêndice B, para a dimuição do tamanho das RdP's e, consequentemente, do espaço de estados do grafo de alcançabilidade.

Empregando-se uma RdP hierárquica, minimiza-se a complexidade de análise pois seu tamanho é menor se comparado ao de uma rede integrada. Para este caso, os estudos de ausência de bloqueio devem ser reconduzidos sobre um novo grafo de alcançabilidade, bem como os de análise temporal utilizando a técnica de tempo global de Lima (LIMA, 2007).

4.3 Considerações do Capítulo

RdP têm sido usadas na construção de software embarcado como modelos que são evoluídos até as próximas fases de desenvolvimento. Modelos de RdP podem ser constantemente melhorados desde a fase inicial de levantamento de requisitos, podendo auxiliar na identificação, entendimento, validação e verificação de requisitos e da solução do software, chegando até a síntese, ou

geração automática de código. Neste trabalho nos mantivemos no escopo da verificação temporal do software embarcado.

Modelos de padrões de projeto auxiliam no aumento da produtividade e qualidade por permitirem o reuso de soluções em diferentes projetos que possuam as mesmas estruturas de problema. Redes de Petri que modelam estruturas complexas software de software embarcado foram propostas por (LIME; ROUX, 2003) como padrões de projeto. É possível utilizá-las em nosso trabalho por possuírem as mesmas propriedades das redes sobre as quais a técnica de tempo global são aplicadas.

No desenvolvimento baseado em modelos, RdP podem ser originadas de outros modelos e evoluídas para a fases seguintes do desenvolvimento do software. O esforço dos pesquisadores na área de tradução de modelos tem sido transformar a linguagem UML em métodos formais, incluindo RdP. Nosso método pode se beneficiar de mais técnicas que sejam desenvolvidas para essa tradução.

Na engenharia reversa, a tradução do código fonte para redes de Petri, por exemplo as FCPN (*Free-Choice Petri Nets*) e CCPN (*Complex-Choice Petri Nets*) (SGROI *et al.*, 1999), permitem que o código seja modelado e validado temporalmente, caso existam mecanismos de obtenção dos intervalos temporais que devem ser associados às transições que representam a execução de comandos ou conjuntos de comandos.

Entretanto, a análise de tempo global também sofre do problema de explosão de estados. Mas, como na aplicação da análise de bloqueios, citada na Etapa 2, também é possível diminuir este problema aplicando a técnica de desdobramento sobre o grafo de alcançabilidade gerado pela técnica de tempo global. Entretanto é possível que uma das vantagens do tempo global, que é sua precisão, seja em parte perdida. Estudos específicos sobre este tema devem ser conduzidos para permitir qualquer conclusão a respeito.

No caso do software ser grande o suficiente para que seja subdividido em módulos de implementação, como métodos, funções e procedimentos, as análises comportamentais, estruturais e de tempo global são realizadas primeiramente sobre estes módulos. O objetivo de aplicar a análise de tempo global individualmente nos módulos é diminuir a complexidade e tamanho da rede de Petri, e consequentemente do grafo de classes. Desta maneira, é possível encontrar o tempo em que o sistema permanece no estado referente à execução de cada módulo.

Outra técnica que propomos para diminuir o tamanho do grafo de classes de tempo global é a escolha da granularidade de modelagem dos comandos, tarefas ou subsistemas em transições.

À medida que subimos o nível de abstração, ou granularidade, diminuímos a complexidade das redes e o tamanho dos grafos de classes, entretanto perdemos em capacidade de representação e consequentemente de análise.

Como nos testes comportamentais e estruturais, que são complementares, consideramos que o uso deste método quantitativo seja complementar ao uso de verificações de modelos. O método proposto não é robusto o suficiente para garantir todos os cenários de comportamento do software, como a verificação de modelos bem feita o é, enquanto a verificação de modelos ainda possui problemas em retornar valores de execução e na falta de agilidade e usabilidade para o desenvolvedor, mesmo aqueles acostumados com redes de Petri.

O principal objetivo deste método é aplicar a análise de tempo global desenvolvida por Lima (LIMA, 2007) na verificação de software embarcado. O método geral proposto neste capítulo parte da modelagem de RdP's a partir de código-fonte ou modelos de especificação, passando por análises comportamentais e estruturais, chegando até a geração da RdP que represente todo o sistema, seja por hierarquização ou integração das redes já analisadas. Propõe-se, finalmente, utilizar novamente técnicas de análise, agora sobre a rede que representa o sistema, de maneira a finalizar a aplicação da análise de tempo global e verificar as restrições temporais do software embarcado.

5 *Aplicação do Método de Verificação por Tempo Global*

Este capítulo apresenta um modelo de aplicação da técnica de tempo global (MTG) no contexto do método de verificação de software embarcado usando tempo global. Considerando o método proposto no capítulo 4, na seção 4.2, este modelo se insere na etapa 1, de modelagem, e nas etapas 4 e 6, de análise temporal.

A partir do grafo de classes gerado pela técnica TG, é possível extrair informações para a análise de restrições temporais do sistema modelado pelas RdP-T. Nossa aplicação da técnica TG é baseada no mapeamento entre tarefas e elementos da RdP do trabalho de Lime e Roux de 2009 (LIME; ROUX, 2009) e nos padrões de projeto do trabalho de Lime e Roux de 2003 (LIME; ROUX, 2003). Esta atividade está inserida na etapa 1 do método proposto no capítulo 4. Após a modelagem das RdP-T seguindo a seção 5.1 e a geração do grafo de classes de acordo com a técnica TG, propomos a geração de caminhos que se referem às sequências de disparo, de acordo com a definição 12. Estes caminhos devem ser gerados de acordo com algum critério. Propomos que estes critérios satisfaçam as políticas de escalonamento *Earliest Deadline First* (EDF) e Prioridade Fixa (PF). A geração do grafo de classes equivalentes e a geração das sequências, ou caminhos, de disparo podem ser contextualizadas nas etapas 4 e 6, de análise temporal.

A seção a seguir apresenta definições do mapeamento de tarefas em RdP. Na seção 5.1 são introduzidos alguns exemplos dos padrões de projeto do trabalho (LIME; ROUX, 2003). Na seção 5.2 definimos como as políticas de escalonamento são aplicadas como critérios de geração sequências de disparos. Os padrões de projeto apresentados como exemplos são utilizados em experimentos de aplicação do método que, por sua vez, são explicados na seção 5.3.

Assumimos que o projeto do software já foi realizado, ou seja, o problema inicial da alocação de processos funcionais a recursos da arquitetura foi resolvido. O objetivo deste método de aplicação é definir e resolver o problema de escalonamento para uma porção da especificação funcional alocada para um único processador.

5.1 Mapeamento de Tarefas em RdP-T

Seja, de acordo com (LIME; ROUX, 2009):

- $\tau \in$ *Tasks*, sendo *Tasks* o conjunto de tarefas do sistema, onde não existe migração de tarefas entre processadores.
- *Sched:Procs* $\mapsto \{PF,EDF\}$ a função que mapeia um processador para uma política de escalonamento, sendo PF "*prioridade fixa*" e EDF "*Earliest Deadline First*" ;
- Π*: Tasks* $\mapsto$ *Procs* a função de que mapeia uma tarefa para seu processador;
- ϖ*: Tasks* $\mapsto \mathbb{N}$, para $Sched(\Pi(\tau)) = PF$, a função que dá a prioridade da tarefa no processador;
- δ*: Tasks* $\mapsto (Q^+ \times (Q^+ \cup \{\infty\}))$, para $Sched(\Pi(\tau)) = EDF$, a função que dá o intervalo limite (*deadline*) da tarefa em relação a seu tempo de ativação.

Para mapear cada lugar da RdP-T para uma tarefa, usamos:

- $\gamma : P \mapsto$ Tasks $\cup\{\phi\}$ a função que associa cada lugar $p \in P$, $P \in RdPT$ a uma tarefa, onde ϕ denota que o lugar não é mapeado para qualquer tarefa real.

Como em (LIME; ROUX, 2009), assumimos que, para cada transição, existe no máximo um lugar p, $p \in Pre(t)$ e $\gamma(p) \neq \phi$, i.e., cada lugar que alcança uma transição $t \in T$, $T \in RdPT$ é associado a uma tarefa distinta. Se $\forall p \in Pre(t)$, $\gamma(p) = \phi$, então t não é associado a qualquer tarefa real e dizemos que ele *é parte de* ϕ, denotado por $\gamma(t) = \phi$.

Assim, para cada transição t, dizemos que t *é parte da* tarefa τ e denotamos $t \in \tau$ se um de seus lugares de entrada é mapeado para $\tau : t \in \tau \Leftrightarrow \exists p \in Pre(t)$, tal que $\gamma(p) = \tau$. Então, denotamos que $\gamma(t)$ é a tarefa tal que $t \in \tau$.

Cada tarefa τ é modelada por uma subrede da RdP-T composta de lugares p mapeados para τ por γ e de transições t (com seu tempo estático $e(t)$) que são partes de τ.

Ainda como em (LIME; ROUX, 2009), assumimos que no máximo uma instância de cada tarefa é ativa em um dado instante, o que é expresso pela restrição: no máximo um lugar mapeado para τ por γ é marcado em um dado instante. Seja $B(\tau)$ o conjunto de transições que *iniciam* a tarefa τ e similarmente, seja $E(\tau)$ o conjunto de transições que encerram τ. Estes dois conjuntos são definidos pelo usuário como parte da fase de modelagem.

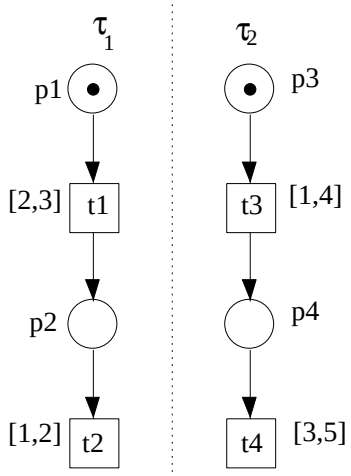

Figura 5: RdP-T de duas tarefas sobre um processador (LIME;ROUX,2003).

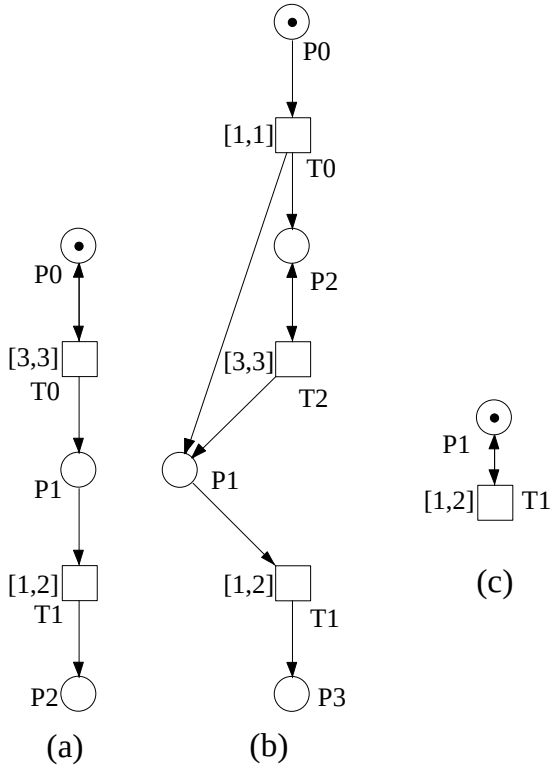

Figura 6: RdP-T de esquemas básicos de ativação: (a) periódica, (b) periódica atrasada e (c) cíclica (LIME;ROUX,2003).

Depois que a RdP-T é modelada, propomos a geração do grafo de classes de estados de acordo com a técnica de TG.

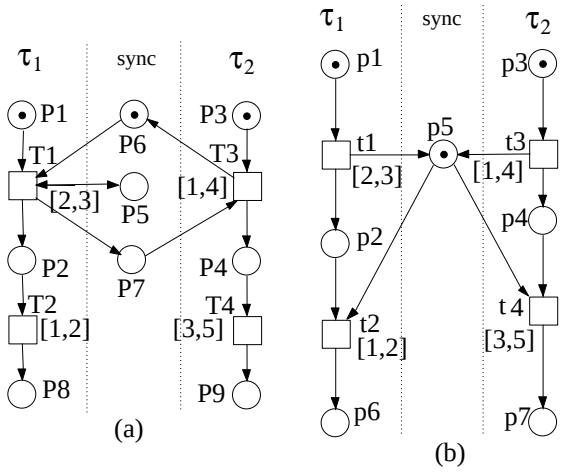

Figura 7: RdP-T de sincronização: (a) mostra um modelo para eventos memorizados e (b) para recursos compartilhados usando um semáforo (LIME;ROUX,2003).

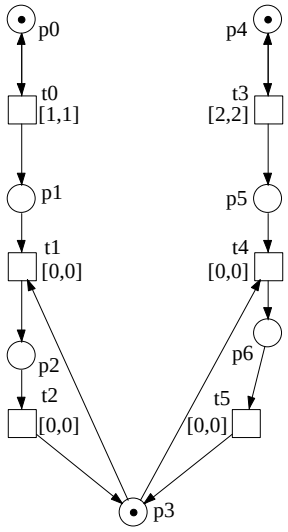

Figura 8: Protocolo de teto de prioridade com dois grupos de tarefas (LIME;ROUX,2003).

Padrões de Projeto

Padrão de projeto pode ser entendido como a descrição de um problema ou de um tipo de problema recorrente e uma solução geral para este problema (NAEDELE; JANNECK, 1998).

Nas figuras 5, 6 e 7 apresentamos exemplos de padrões de projeto retirados do trabalho de (LIME; ROUX, 2003) para exemplificar como são modeladas as redes de Petri a partir das definições do mapeamento entre tarefas e RdP-T da seção 5.1. Estas figuras apresentam alguns modelos de serviços de sincronização, e são apresentadas como estruturas gerais de RdP-T que representam tarefas de software embarcado. Entretanto, as possibilidades de modelagem não devem ser restritas às apresentadas nesta seção e nem mesmo ao trabalho de (LIME; ROUX,

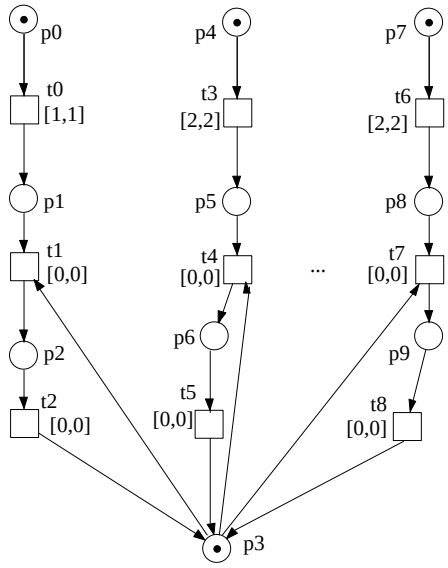

Figura 9: RdP-T de acesso ao barramento CAN, com três grupos de tarefas (LIME;ROUX,2003).

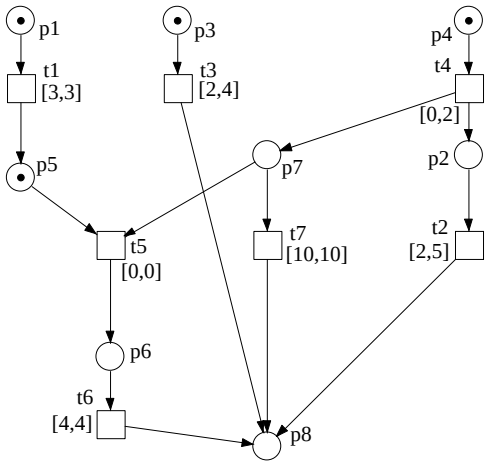

Figura 10: RdP-T de duas tarefas sobre um processador (LIME;ROUX,2003).

2003).

5.2 Análise de Escalonabilidade sobre a RdP-T

Considerando que as tarefas que compõem o sistema estão modeladas (etapa 1) em redes de Petri temporais e que estas representam uma solução de projeto para o software, propomos verificar *a priori* se um dado conjunto de tarefas cumpre ou não seus requisitos temporais (etapas 4 e 6).

Para isso, a análise de escalonabilidade propõe-se a responder à seguinte pergunta: ”Cada tarefa satisfaz o seu *deadline*?”. A técnica de tempo global apresentada no capítulo 3 deve então

ser aplicada sobre a RdP-T modelada de acordo com a seção 5.1 para a geração do grafo de classes de tempo global. Após a geração deste grafo, uma sequência de disparo é gerada de acordo com uma das políticas de escalonamento propostas a seguir, para a observação dos tempos de modo a responder à pergunta do início deste parágrafo.

Algumas análises específicas são possíveis de serem realizadas sobre os dados gerados pela técnica de tempo global e associados às sequências de disparos. A seguir exemplificamos algumas.

Análise de *deadline* e Análise de janela. Os requisitos temporais das tarefas podem ser de *deadline*, de sincronismo e de distância. Interessam-nos as duas primeiras: a análise de *deadline* é realizada tomando-se como base o valor do *deadline*, ou seja, da limitação ao tempo máximo para o término da tarefa, enquanto a análise de *janela*, é realizada tomando-se como base a delimitação máxima e mínima, ou seja um intervalo, do instante de término.

Análise de Hiperperíodo (*hyper period*) ou de Ciclo Maior (*major cycle*). "(...) Quando tarefas periódicas são executadas indefinidamente, a observação do comportamento teria também que ser permanente. Entretanto, como o comportamento do conjunto de tarefas é periódico, é suficiente analisar somente um período, ou pseudoperíodo, (...) chamado período de escalonamento, tamanho do escalonamento ou hiperperíodo (...)". O hiperperíodo é delimitado por instantes críticos. " O período de escalonamento de um conjunto de tarefas inicia-se na data de ativação mais cedo, i.e., na data $t_i = Min\{r_{i,0}\}$, sendo $r_{i,0}$, a data de primeira ativação da tarefa i, sendo i pertencente ao conjunto de tarefas periódicas, considerando todas as tarefas desse conjunto" (COTTET *et al.*, 2002). O período conclui-se em $t_f = Max\{r_{i,0}, (r_{j,0} + D_j)\} + 2 \cdot MMC(T_i)$. Esta é uma função, principalmente, do mínimo múltiplo comum (*MMC*) dos períodos (T_i), sendo $r_{i,0}$ e $r_{j,0}$, as datas das primeiras ativações das tarefas i e j que variam no conjunto de índices de tarefas periódicas e aperiódicas, respectivamente, e sendo D_j o *deadline* da tarefa aperiódica j (COTTET *et al.*, 2002).

Análise de pior caso (wcet). Os sistemas de tempo real somente podem garantir que os *deadlines* são satisfeitos se os tempos de execução do pior caso (*worst case execution time - wcet*) de todas as tarefas da aplicação são conhecidos *a priori*. O *wcet* de uma tarefa é um limite superior para o intervalo de ativação da tarefa e seu término. Deve ser válido para todos os cenários de execução da tarefa (HATLEY; PIRBHAI, 1987).

Mapeamento de Políticas de Escalonamento

Após a modelagem das RdP-T a partir do proposto na seção 5.1, ou seja, utilizando os padrões de projeto e a geração do grafo de classes de acordo com a técnica TG, propomos a geração de caminhos que se referem às sequências de disparo, de acordo com a definição 12. Estes caminhos devem ser gerados de acordo com algum critério pré-definido. Nas seções a seguir estabelecemos os critérios para as políticas de escalonamento *Earliest Deadline First* e *Fixed Priority*. Para o caso de uma política de escalonamento Executiva Cíclica (CE), para *Sched:Procs* $\mapsto \{CE\}$, onde CE é Executiva Cíclica, o modelo RdP-T representa somente uma tarefa que é tipicamente realizada como um laço infinito na função principal *main()*, como na figura 6. Devido ao fato de *CE* não possuir um critério específico a ser satisfeito na enumeração dos caminhos, esta política é alcançada somente pela modelagem da RdP-T e não é necessário formalizar esta função *Sched*.

Fixed Priority
($Sched(\Pi(\gamma(t))) = PF$**)**

A função ϖ*: Tasks* $\mapsto \mathbb{N}$ guia a enumeração da sequência de disparo composta pelas transições disparáveis t_f. Ou seja, as prioridades de execução das tarefas são associadas às transições t_i da RdP-T e, para a função ϖ, $t_f = t_i$ para a t_i com a maior prioridade. No caso das transições com a mesma prioridade em uma mesma classe, um dos seguintes critérios de desempate podem guiar a enumeração: uma escolha FIFO (primeiro-a-entrar-primeiro-a-sair); uma escolha "*Earliest Deadline First*" como apresentaremos na próxima seção; ou uma escolha aleatória.

Ao final desta enumeração temos uma sequência completa de disparos, gerada de acordo com a política de escalonamento PF, e um intervalo que corresponde a seu tempo global, i.e. um intervalo referente ao acúmulo do tempo desde o início até o final da execução da sequência de disparos, de acordo com a técnica TG e apresentado em 3.3.3.

Earliest Deadline First
($Sched(\Pi(\gamma(t))) = EDF$**)**

A função δ*: Tasks* $\mapsto (Q^+ \times (Q^+ \cup \{\infty\}))$ guia a enumeração sobre o grafo de classes usando a política de escalonamento *Earliest Deadline First*. Esta política permite escolher a transição t_i que possui o intervalo com menor o limite superior (*deadline* máximo) dado por $\delta(\tau)$ como a seguir:

Seja uma transição t_i com $r_k(t_i) = [a,b]$ calculado na classe c_k, $c_k[t_f > c_{k+1}$, e o $\delta(\tau)$ de uma

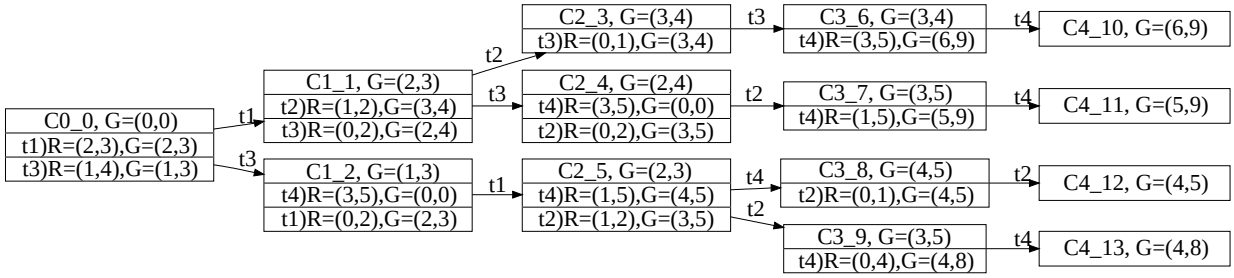

Figura 11: Grafo de classes TG de 5 níveis, das RdP-T das figuras 5 e 7(b).

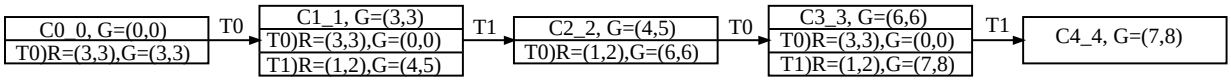

Figura 12: Grafo de classes TG de 5 níveis, da RdP-T da figura 6(a).

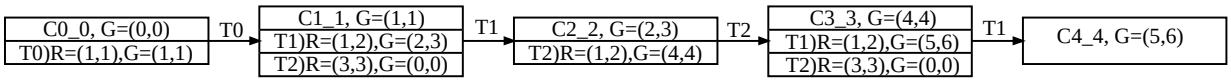

Figura 13: Grafo de classes TG de 5 níveis, da RdP-T da figura 6(b).

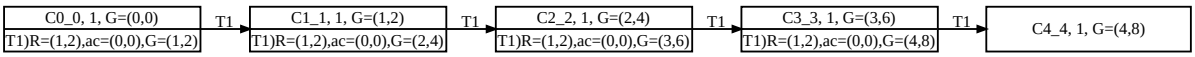

Figura 14: Grafo de classes TG de 5 níveis, da RdP-T da figura 6(c).

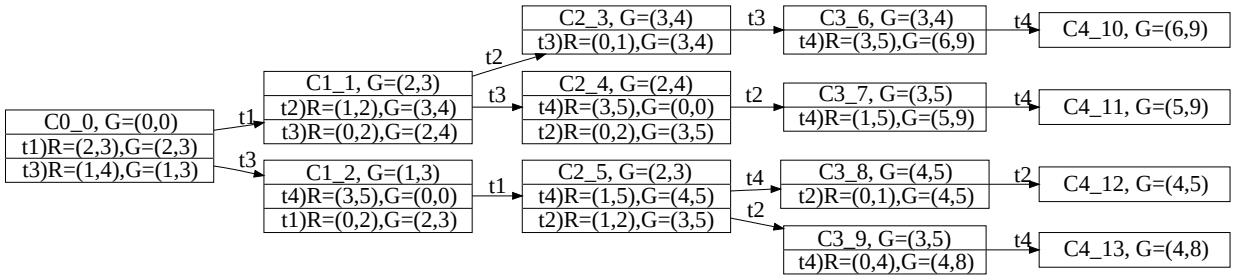

Figura 15: Grafo de classes TG de 5 níveis, da RdP-T da figura 7(b).

transição t_i definido como: $\delta(\tau) = b$, $t_f = t_i$ para t_i o menor b.

Como na política de escalonamento PF, ao final da enumeração temos a sequência de disparos e seu tempo global correspondente, como apresentado em 3.3.3.

5.3 Exemplos de Análise de Escalonabilidade

Apresentamos a seguir alguns exemplos de aplicação da técnica usando alguns dos exemplos de RdP-T apresentadas na seção 5.1.

Considerando a RdP-T da figura 5, a tarefa τ_1 possui prioridade $\varpi = 1$ e um ponto de preempção. A preempção controlada por um núcleo operacional não é modelada, apenas a possibilidade de preempção entre as tarefas. A tarefa τ_2 também possui um ponto de preempção, mas a prioridade $\varpi = 2$. Então, $\varpi(t_1) = 1$, $\varpi(t_2) = 1$, $\varpi(t_3) = 2$ e $\varpi(t_4) = 2$. O grafo de classe de acordo com a técnica TG é apresentado na figura 11. Para $Sched(\Pi(\tau)) = PF$, a sequência de disparo é: t_3, t_1, t_4, t_2. É interessante notar que t_1 somente é disparável na classe $C2\_5$, mesmo t_4 sendo habilitado e t_3 estando na mesma tarefa que t_4; t_4 executa depois de t_1 porque t_4 possui mais alta prioridade. O tempo global desta sequência é $g_{4\_13} = [4,5]$. Para $Sched(\Pi(\tau)) = EDF$, a sequência de disparo é: t_1, t_2, t_3, t_4, com tempo global $g_{4\_10} = [6,9]$, ou t_1, t_3, t_2, t_4, com $g_{4\_11} = [5,9]$.

Para a RdP-T da figura 6(a) que representa uma única tarefa τ_1 com prioridade $\varpi = 3$ sendo $\varpi(T_0) = 3$ e $\varpi(T_1) = 3$. Seu grafo TG é apresentado na figura 12. Existe apenas uma sequência de disparo possível, tanto para $Sched(\Pi(\tau)) = PF$ quanto para $Sched(\Pi(\tau)) = EDF$, pois nas classes $C1_1$ e $C3_3$ quando T_0 e T_1 estão habilitadas, apenas T_1 é disparável.

As mesmas considerações do exemplo anterior podem ser feitas para a RdP-T da figura 6(b) que também representa uma única tarefa τ_1 com prioridade $\varpi = 3$ sendo $\varpi(T_0) = 3$, $\varpi(T_1) = 3$ e $\varpi(T_1) = 3$. Seu grafo TG é apresentado na figura 13 e existe apenas uma sequência de disparo possível, tanto para $Sched(\Pi(\tau)) = PF$ quanto para $Sched(\Pi(\tau)) = EDF$. Também ocorre que nas classes $C1_1$ e $C3_3$ quando T_1 e T_2 estão habilitadas, apenas T_1 é disparável.

Para a RdP-T da figura 7(b): a tarefa τ_1 possui prioridade $\varpi = 1$ e um ponto de preempção controlado pelo semáforo (lugar p_5). A tarefa τ_2 também possui um ponto de preempção controlado pelo mesmo semáforo, mas com prioridade $\varpi = 2$. Então, $\varpi(t_1) = 1$, $\varpi(t_2) = 1$, $\varpi(t_3) = 2$ e $\varpi(t_4) = 2$. Coincidentemente, como os tempos deste exemplo são iguais aos tempos do exemplo da figura 5 o grafo TG para a figura 7(b) foi gerado igual ao da figura 5, por este motivo apresentamos apenas o grafo da figura 11. Ainda, as sequências de disparo são as mesmas: para $Sched(\Pi(\tau)) = PF$, a sequência de disparo é: t_3, t_1, t_4, t_2 com tempo global $g_{4\_12} = [4,5]$ e para $Sched(\Pi(\tau)) = EDF$, a sequência de disparo é: t_1, t_2, t_3, t_4, com tempo global $g_{4\_10} = [6,9]$, ou t_1, t_3, t_2, t_4, com $g_{4\_11} = [5,9]$.

As mesmas análises podem ser feitas para as redes de Petri apresentadas nas figuras 8 e 9 e seus grafos de classes apresentados respectivamente nas figuras 25 e 26.

5.4 Implementação

Foi realizada uma implementação de software da técnica de tempo global apresentada no capítulo 3, o que propicia uma ferramenta para a aplicação do método proposto no capítulo 4. Uma breve visão geral a respeito desta aplicação será feita a seguir: primeiramente são apresentadas suas funcionalidades principais e posteriormente suas entradas e saídas e sua estrutura de objetos.

Funcionalidades A ferramenta possui as seguintes funcionalidades:

- Geração do grafo de classes com os seguintes valores em cada classe, ou nó: tempo global da classe, coeficiente de ajuste de persistência, tempo global e tempo relativo de cada transição disparável, marcação da rede; e identificação da transição disparada para cada arco;
- Geração facultativa de sequência de disparo, aleatória ou de acordo com alguma política de escalonamento, estando atualmente disponível PF e EDF;

Caso a RdP-T seja acíclica, é possível gerar o grafo completo ou, ainda, limitado estruturalmente pelo número de níveis ou pelo número de nós. Caso a RdP-T seja cíclica, sugere-se a utilização destes limites estruturais para não haver a explosão de estados. Optou-se por realizar uma limitação estrutural para permitir estudos mais detalhados sobre o grafo de classes não limitado. Assim, é possível identificar elementos úteis para a análise temporal como, por exemplo, os hiperperíodos, ou ciclos maiores, refletido no grafo de classes.

Entradas e Saídas Existem atualmente dois tipos de arquivos de entrada possíveis para a ferramenta: **.net** e **.in**. Para cada execução, exige-se um arquivo de entrada que possui a descrição da rede de Petri temporal, no formato **.net** da ferramenta Tina (LAAS, 2010).

Para a geração de sequências de disparo baseadas na política de PF ($Sched(\Pi(\tau)) = PF$), deve-se fornecer para a ferramenta um arquivo de entrada **.in**. Este arquivo possui uma definição de valor de prioridade ϖ para cada transição da RdP-T por linha de arquivo.

A ferramenta gera como saída um arquivo com o grafo de classes e seus valores nos formatos **.txt** e **.dot** (GRAPHVIZ, 2010), para a visualização do grafo.

Estrutura de Objetos A ferramenta foi construída utilizando-se o paradigma de orientação a objetos e a linguagem C++.

As classes de objetos mais importantes são as de operações intervalares (*intervalo.cpp),* as de leitura e *parser* do arquivo de entrada (*leitura.cpp* e *parser.cpp*), as da RdP-T (*rede.cpp*), as do grafo de classes (*grafo.cpp*), as de classe de equivalência (*classe.cpp*) e as referentes à classe de relacionamento entre classe e transição (*transicaoClasse.cpp*).

Inicia-se a execução com a leitura do arquivo .net de entrada, através dos objetos leitura e parser. Após o objeto rede ter sido criado com todos os valores do arquivo de entrada, inicia-se a construção do grafo de classes, criando-se os objetos classe à medida em que são necessários. Para cada transição disparável na classe de equivalência analisada cria-se um objeto "transicaoClasse" que contém todos os valores calculados utilizando-se as operações intervalares. O objeto transicaoClasse possui um atributo que direciona para a próxima classe do grafo, criando semanticamente o arco para a próxima classe de equivalência.

5.5 Estudo de Caso

Realizamos um estudo de caso com o objetivo de avaliar a aplicabilidade do método de verificação por tempo global proposto nesta tese. Conduzimos dois tipos de estudos de caso baseado em: tarefas e análise de escalonabilidade, como proposto nas seções 5.1 e 5.2; e mapeamento de requisitos sobre uma RdP-T.

5.5.1 Estudo de Caso 1

Este estudo de caso foi baseado nos seguintes passos, com suas respectivas etapas do modelo proposto no capítulo 4:

1. modelagem das RdP, equivalente à etapa 1;
2. análise comportamental, equivalente às etapas 2 e 6;
3. análise temporal usando a ferramenta de análise de tempo global construída, equivalente às etapas 4 e 6;

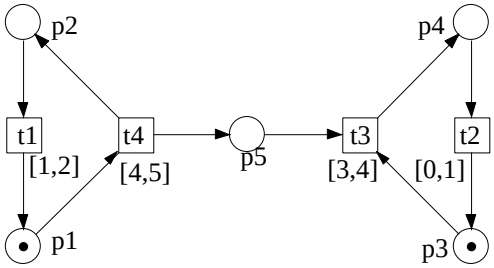

Figura 16: RdP-T de duas tarefas cíclicas sincronizadas por um semáforo (LIME;ROUX,2003).

Etapa 1 - Modelagem das RdP

Este estudo de caso é baseado em uma rede de Petri modelada em (LIME; ROUX, 2003). Esta rede é usada aqui por ser uma rede cíclica e representar uma estrutura muito comum no contexto de software embarcado em tempo real. A RdP-T é apresentada na figura 16.

Etapa 2 - Análise Estrutural e Comportamental

Esta rede de Petri foi analisada pela ferramenta Tina (LAAS, 2010), que retornou os seguintes resultados para a nossa análise:

- Quanto à análise de invariantes.
  - 2 invariantes de lugar: $M(p_1)+M(p_2)=1$ e $M(p_3)+M(p_4)=1$;
  - 2 invariantes de transição: $t_1;t_3$ e $t_2;t_4$;
- Quanto à cobertura. O grafo de cobertura resultante é apresentado na figura 17.

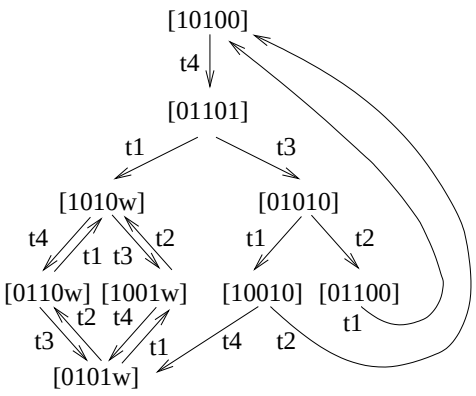

Figura 17: Grafo de cobertura da RdP-T da figura 16.

Essa rede é um exemplo simples de duas tarefas com características semelhantes, independentes, cíclicas, sendo que os ciclos também são independentes. Os invariantes comprovam essas observações e o grafo de cobertura mostra uma rede cíclica não-limitada.

Etapa 4 - Análise Temporal

Como a RdP modelada na etapa 1 já possui tempos associados às transições, não realizamos os processos associados à etapa 3 do método geral apresentado no capítulo 4 e passamos diretamente à análise temporal da etapa 4. Este fluxo alternativo é permitido pelo método, como visualizado na figura 4 do capítulo mencionado.

O grafo de classes segundo a técnica de tempo global é apresentado na figura 27. Apresentamos este grafo como uma árvore, para aumentar a capacidade de avaliação sobre este grafo. Ainda, por ser uma RdP-T cíclica, criamos um árvore com 15 níveis de classes.

A sequência de transições dos 14 disparos, segundo a política EDF, é s=$\{$t4, t1, t3, t2, t4, t1, t3, t2, t4, t1, t3, t2, t4, t1$\}$ e seu tempo global é $g = [20,28]$ ut, encontrado na classe C14_44. A partir dessa sequência, e tomando por base o grafo de classes, é possível identificar dois hiperperíodos, $h_1 = [t1,t3,t2,t4]$ e $h_2 = [t1,t3,t4,t2]$, sobre os quais se pode realizar a análise de hiperperíodos. O tempo global, desde o primeiro disparo de $t4$, do primeiro hiperperíodo será $g(t4+h_1) = [9,10]$ e do segundo hiperperíodo será $g(t4+h_2) = [7,10]$.

É interessante notar que, na análise temporal, obtém-se um grafo de classes onde a rede é 1-limitada, ou segura, e o espaço de estados do grafo de cobertura é maior que o espaço de estados do grafo de classes. É possível observar esta última afirmação no grafo de classes realizando uma análise de marcação. A figura 18 representa esta análise: a parte mais clara do grafo refere-se ao espaço de estados do grafo de cobertura, enquanto a parte mais escura refere-se ao espaço de estados do grafo de cobertura e ao espaço de estados do grafo de classes.

5.5.2 Estudo de Caso 2

Este estudo de caso foi baseado nos seguintes passos, com suas respectivas etapas do modelo proposto no capítulo 4:

1. modelagem das RdP, equivalente à etapa 1;
2. análise temporal com tempo global, equivalente à etapa 4;

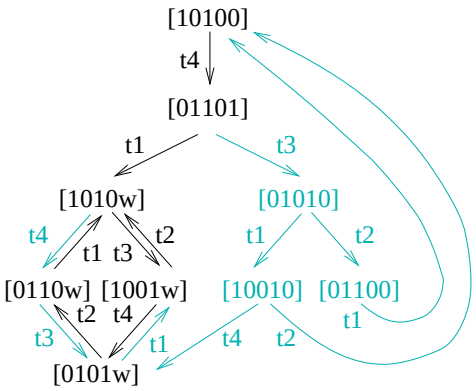

Figura 18: Grafo de cobertura da RdP-T da figura 16.

3. construção da RdP hierárquica, equivalente à etapa 5;

4. análise temporal com tempo global, equivalente à etapa 6;

Etapa 1 - Modelagem das RdP

Este estudo de caso é baseado em uma rede de Petri modelada em (WANG; DENG; XU, 2000). Esta rede, apresentada na figura 19 foi escolhida como estudo de caso por possuir várias estruturas de representação de software e sistemas embarcados, como a representação de componentes de sensores com ativação periódica (subsistemas B e E da figura), atuadores (subsistemas D e G), componentes de controle (subsistema A) e componentes de conexão e auxílio ao controle (subsistemas C e F). Os subsistemas de cada tipo possuem a mesma topologia e propriedadades temporais. Neste estudo é irrelevante o significado da nomenclatura dos lugares e transições, que servem apenas para identificação destes elementos da RdP-T.

Etapa 4 - Análise Temporal

Considerando que a RdP-T possui tamanho médio, optamos por realizar o cálculo de tempo global inicialmente para os requisitos temporais definidos para suas subredes:

- Requisito 1: o tempo de reação do sistema não deve ser superior a 45 unidades de tempo (ut). O tempo de reação do sistema é o intervalo de tempo que transcorre desde que um evento foi capturado pelos sensores e passado para os componentes de controle e conexão (subsistemas A e C ou F) até sua execução pelo atuador correspondente. Este requisito corresponde à submarcação $M(SYSR1) = 1, M(P302) = 1, M(SC1R1) = 1$.

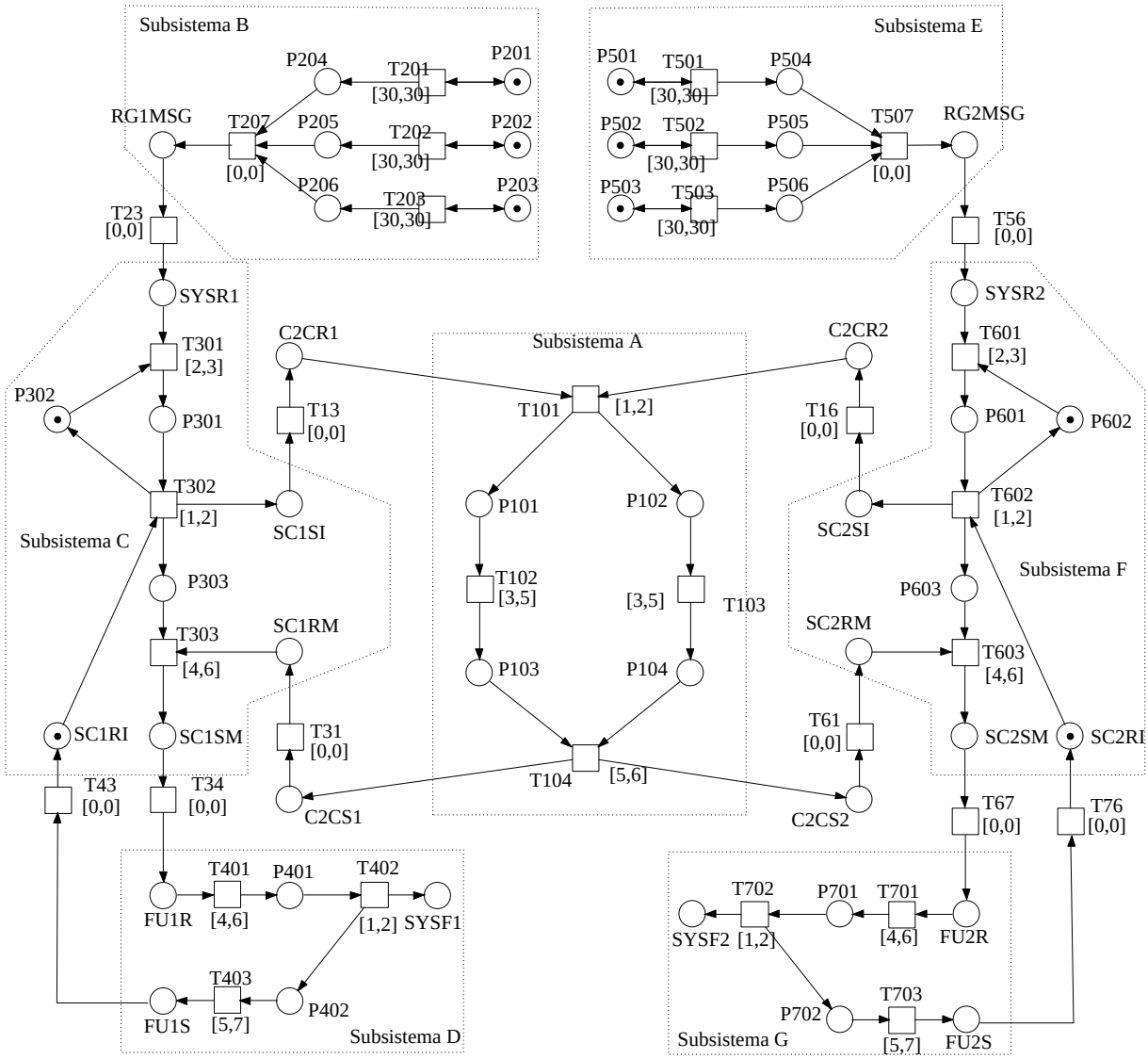

Figura 19: RdP-T de um sistema embarcado.

- Requisito 2: o tempo de atuação e retorno da atuação não deve ser superior a 20 ut. Este tempo é referente à execução e um retorno de evento do componente atuador (subsistemas D ou G). Este requisito corresponde à submarcação $M(SC1SM) = 1$.

- Requisito 3: o tempo de controle não deve ser superior a 22 ut. Este tempo se refere à execução do subsistema A. Este requisito corresponde à submarcação $M(C2CR1) = 1, M(C2CR2) = 1$.

- Requisito 4: cada subsistema de sensores deve enviar informação periodicamente em, no máximo, 40 ut. Este requisito corresponde à submarcação $M(P201) = 1, M(P202) = 1, M(P203) = 1$.

Aplicando a análise por tempo global, encontramos os valores apresentados no próximo parágrafo. O primeiro valor refere-se ao tempo durante o qual o sistema permanece em uma determinada marcação, encontrado usando a definição 20 do capítulo 3. A sequência de transições foi gerada de acordo com a política EDF e satisfaz o respectivo requisito funcional, através da execução da rede, e temporal. O tempo global da sequência de disparo segundo a política EDF também é apresentada a seguir.

- O requisito 1 será satisfeito em [21,39] ut, portanto atende o requisito temporal de não ser superior a 45 ut. Este intervalo envolve todas as possíveis sequências de disparo de transições. A sequência de transições baseada na política EDF é s={T301, T601, T302, T13, T602, T16, T101, T102, T103, T104, T31, T61, T303, T34, T603, T67, T401, T402, T701, T702} e seu tempo global é $g = [22, 32]$ ut, encontrado na classe C20_6364.

- O requisito 2 será satisfeito em [10,15] ut e atende o requisito temporal de não ser superior a 20 ut. A sequência de transições é s={T34, T401, T402, T403, T43} e seu tempo global é $g = [10, 15]$ ut, encontrado na classe C4_4.

- O requisito 3 será satisfeito em [9,13] ut e atende o requisito temporal de não ser superior a 22 ut. A sequência de transições é s={T101, T102, T103, T104} e seu tempo global é $g = [9, 13]$ ut, encontrado na classe C4_6.

- O requisito 4 será satisfeito em [33,36] ut e atende o requisito temporal de não ser superior a 40 ut. A sequência de transições é s={T201, T202, T203, T204, T205} e seu tempo global é $g = [33, 36]$ ut, encontrado na classe C5_22.

Foi realizada uma análise de escalonabilidade baseada em deadline de janela para os requisitos estabelecidos e mapeados na rede de Petri avaliada. A análise de escalonabilidade baseada em prioridade fixa também pode ser realizada nesta etapa de análise temporal.

Etapa 5

De acordo com a descrição da etapa 5 no capítulo 4, é possível simplificar a análise temporal de uma rede de Petri através da hierarquização, ou seja, alocando o tempo global referente a um submodelo da rede a uma transição que represente este submodelo.

Neste estudo de caso realizamos, de acordo com as condições apresentadas nesta etapa, a hierarquização dos submodelos representando o subsistema A e os subsistemas B ou E. Consideramos ser possível tal procedimento devido ao fato dos submodelos B e E, C e F, D e G

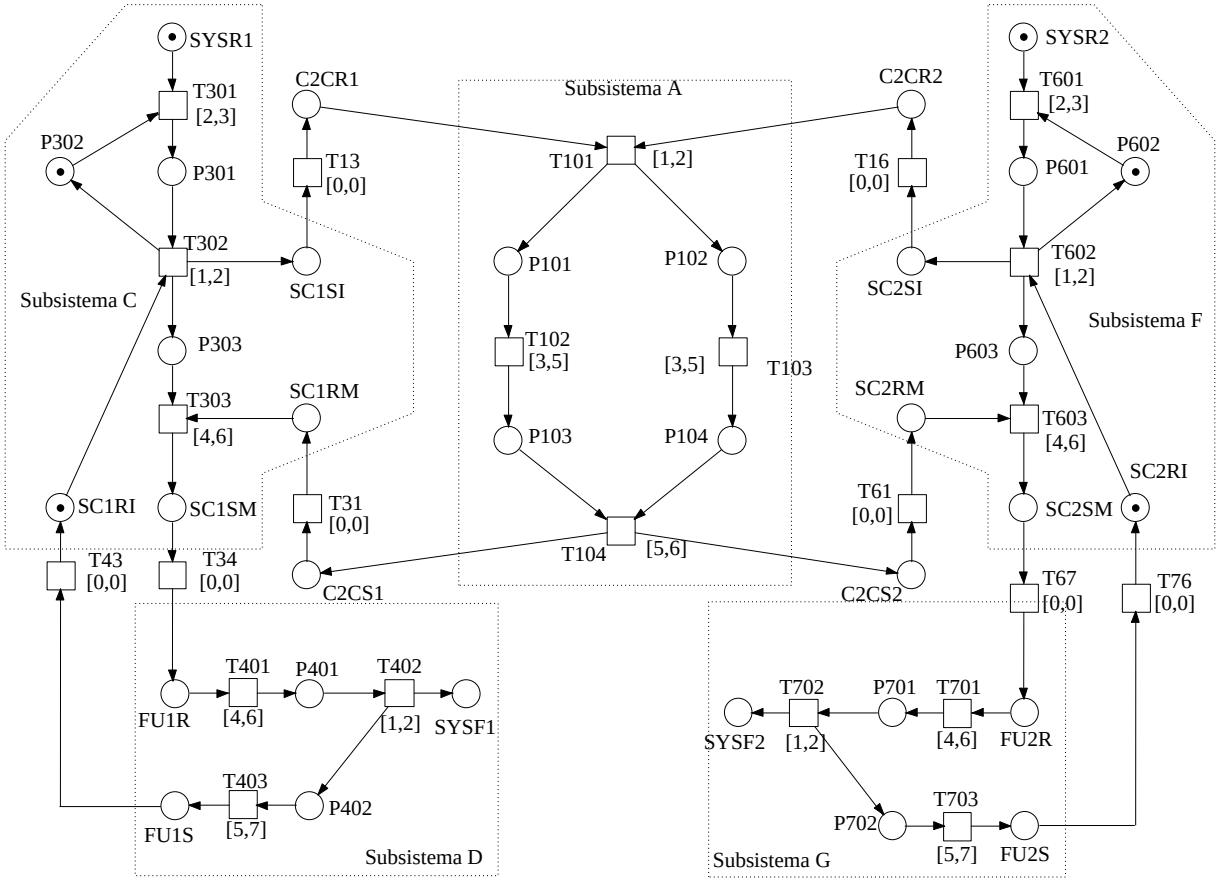

Figura 20: Submodelo do requisito 1 da RdP-T da figura 19.

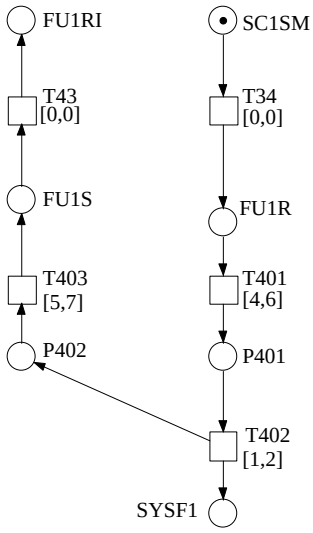

Figura 21: Submodelo do requisito 2 da RdP-T da figura 19.

serem simétricos em relação ao submodelo do subsistema A e possuírem a mesma topologia e propriedadades temporais entre si. Os subsistemas C e F e os subsistemas D e G não puderam ser hierarquizados por não satisfazerem as condições definidas nesta etapa.

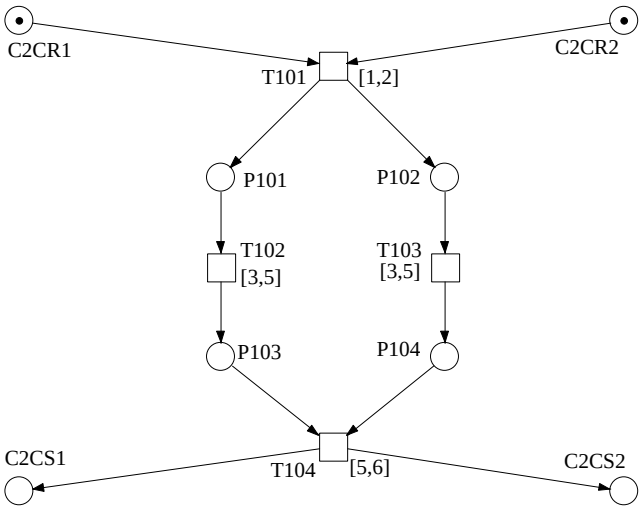

Figura 22: Submodelo do requisito 3 da RdP-T da figura 19.

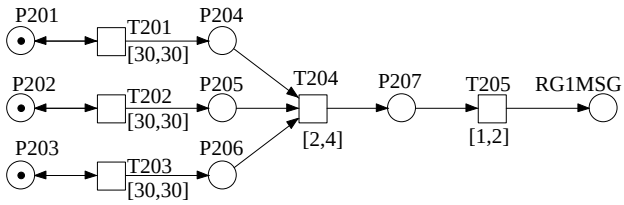

Figura 23: Submodelo do requisito 4 da RdP-T da figura 19.

A RdP-T gerada é mostrada na figura 24.

Etapa 6

Executando a ferramenta de cálculo de tempo global, realizamos a análise temporal da rede hierárquica.

A única sequência possível de disparo de transições é s={Tsensores, T23, T301, T302, T13, Tcentral, T31, T303, T34, T401, T402, T403, T43} e seu tempo global é $g = [59, 75]$ ut, encontrado na classe C13_13.

Como retiramos os ciclos do subsistema de sensores, trocando-os por uma transição, provocamos uma única execução de toda a rede equivalente a um hiperperíodo. Portanto, a análise hiperperíodo realizada para esta rede encontra um tempo global g=[59,75].

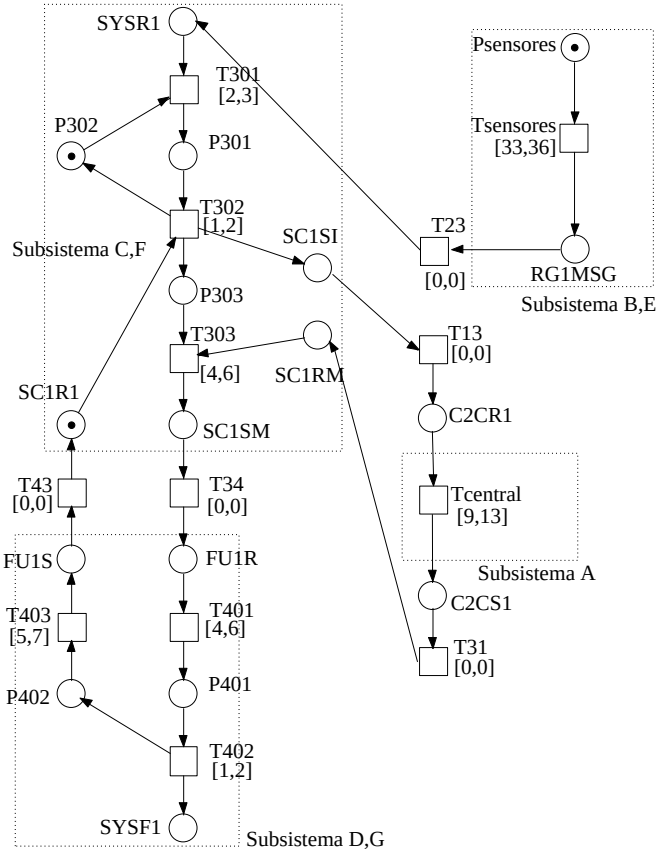

Figura 24: RdP-T hierárquica de um sistema embarcado.

5.6 Considerações do Capítulo

O trabalho (LIME; ROUX, 2009) define uma RdP-T especial, *Scheduling-TPN*, com uma camada de escalonamento que, entre outras coisas, permite mapear cada lugar da rede a uma tarefa. Este trabalho se utiliza de estruturas e técnicas de verificação de modelos, como autômatos híbridos lineares, ferramenta HyTech e estruturas DBM.

Nós propomos usar partes da camada de escalonamento desta rede para modelar uma RdP-T clássica, como na definição 4, de acordo com padrões de projeto do trabalho de Lime e Roux de 2003 (LIME; ROUX, 2003), associando tarefas de sistemas a transições e lugares na rede. A partir das tarefas modeladas em RdP, propomos a geração do grafo de classe de estados de TG e a análise das sequências de disparos que satisfaçam as políticas de escalonamento prioridade fixa (PF) e *earliest deadline first* (EDF).

O trabalho implementado gera uma sequência de disparos que representam um cenário, ou roteiro de comportamento, e seus intervalos de tempo global e relativo. O intervalo de tempo relativo se refere ao tempo durante o qual o sistema permanece em um determinado estado, ou

classe da rede. O intervalo de tempo global se refere ao acúmulo de tempo desde o início da execução do sistema, ou marcação inicial da RdP, até o estado, ou classe, desejado.

As sequências de disparo geradas com o método de tempo global permitem verificar importantes propriedades (BARRETO, 2005):

- Verificação de que as alocações de tarefas do processador são mutuamente exclusivas. Esta propriedade situa que não mais que uma tarefa pode ser alocada ao processador. Isto é possível através da modelagem de RdP com a estrutura de exclusão mútua (*mutexes*).
- Verificação de que a tarefa seguinte somente pode iniciar sua execução após o final da tarefa anterior. Isto é possível analisando o tempo de permanência em uma classe. O tempo de permanência em uma classe não pode ser inferior ao tempo de execução da tarefa.

A utilização do método de tempo global torna mais natural a verificação de restrições temporais quando oferece resultados quantitativos desde o início da execução do sistema. Para agilizar a tarefa do desenvolvedor, propõe-se a utilização de modelos de redes de Petri que já são padrões de projeto, como os propostos por (LIME; ROUX, 2003) e que já possuem suas propriedades estruturais analisadas.

Caso o desenvolvedor opte por modelar sua rede de Petri desde o início, existem na literatura diversos trabalhos que propõem a construção de RdP a partir da junção de elementos bem conhecidos como de processos paralelos, computação de fluxo de dados, exclusão mútua (*mutexes*), *pipeline*, produtor/consumidor, protocolos de comunicação, entre outros (BARRETO, 2005).

A identificação das restrições temporais das tarefas torna-se então uma das atividades mais complexas dentro deste contexto, e que demandam mais experiência do desenvolvedor. A ferramenta desenvolvida pode ser usada no auxílio da verificação destas restrições.

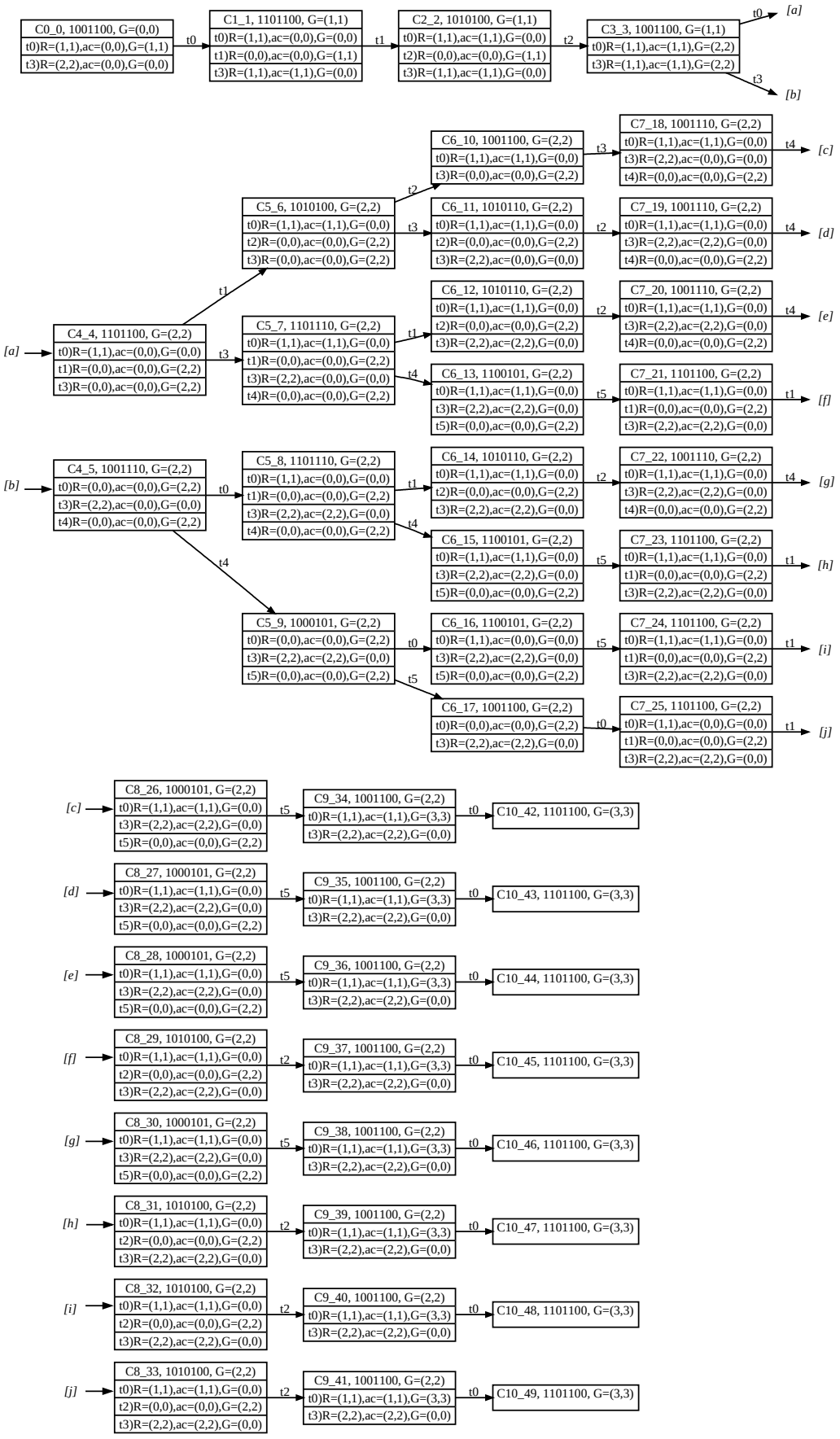

Figura 25: Grafo de classes TG de 5 níveis, da RdP-T da figura 8.

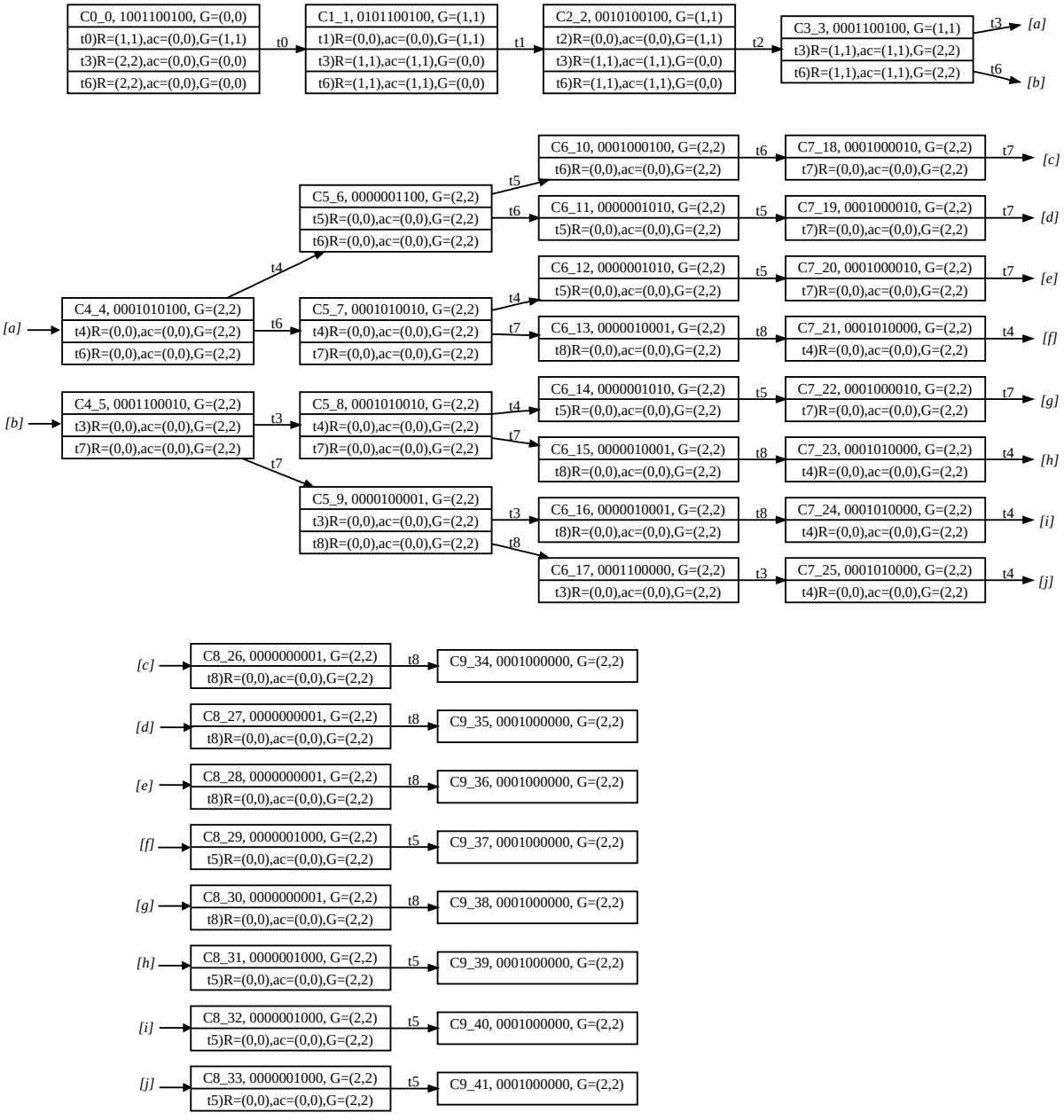

Figura 26: Grafo de classes TG de 5 níveis, da RdP-T da figura 9.

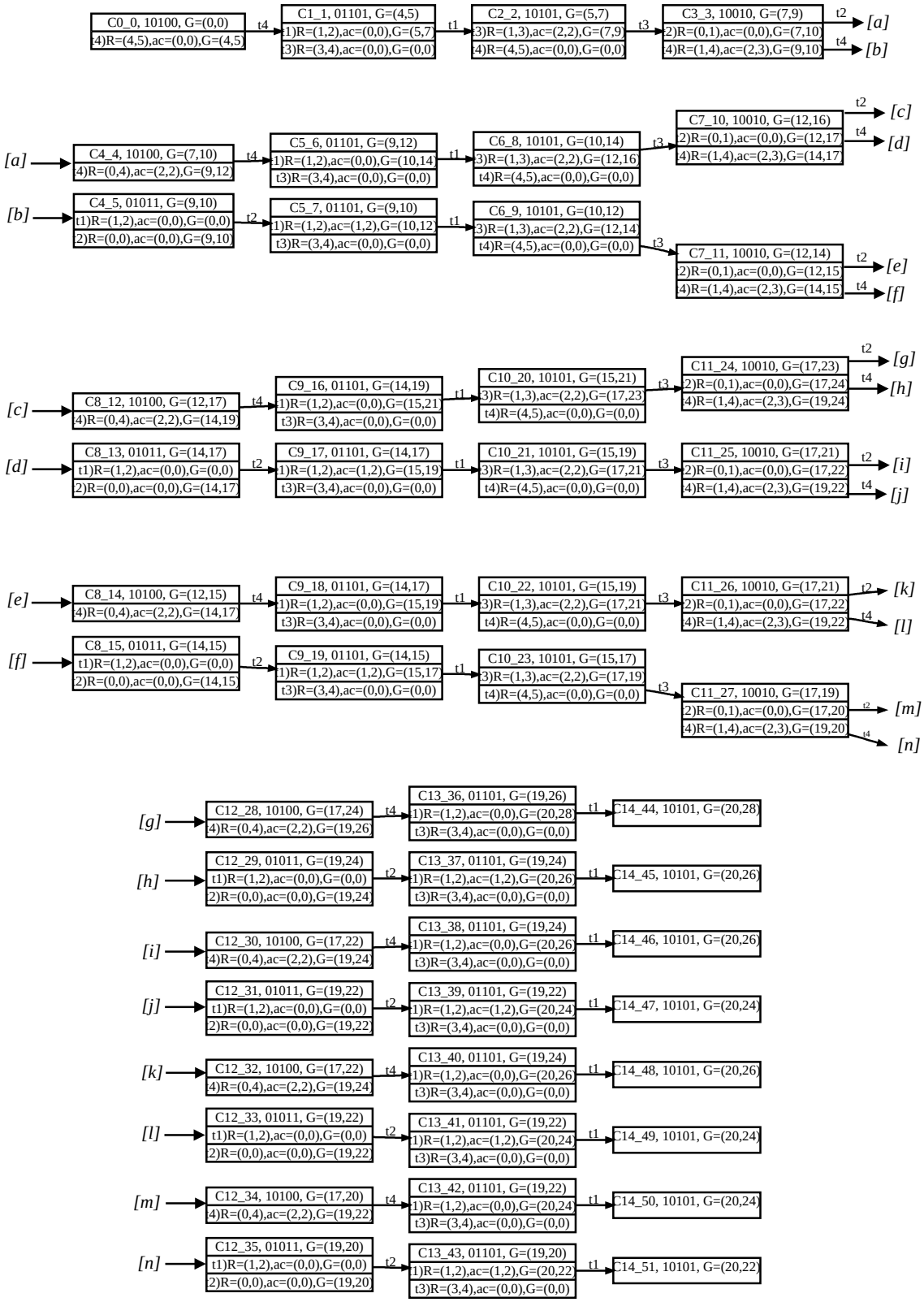

Figura 27: 1a. Parte do grafo de classes de tempo global para a RdP-T da figura 16.

6 Conclusões

O problema tratado nesta tese foi o de verificação de software embarcado, por meio de análise de redes de Petri que modelam código ou especificações de projeto de software embarcado. Escolheu-se redes de Petri porque propriedades de sequencialização, sincronização, comunicação e concorrência do sistema podem ser representadas e verificadas formalmente neste modelo. O objetivo deste método é suportar a análise temporal do software embarcado. Para esta análise temporal foi necessário estabelecer técnicas de tratamento das redes de Petri durante a aplicação do método. O Laboratório de Inteligência Artificial e Métodos Formais do PPGINF da UFPR, laboratório de pesquisa no qual este trabalho foi desenvolvido, tem buscado aprimorar métodos e técnicas de análise temporal baseadas em redes de Petri.

No que se refere ao contexto de aplicação, as redes de Petri têm sido usadas na construção de software embarcado como modelos que são aprimorados sucessivamente, em diferentes fases do desenvolvimento de software. Modelos de RdP podem ser construídos desde a fase inicial de levantamento de requisitos, podendo auxiliar na identificação, entendimento, validação e verificação de requisitos e da solução do software, chegando até a síntese, ou geração automática de código.

Em sistemas de tempo real é extremamente necessária a garantia de que as restrições temporais inerentes a estes sistemas serão satisfeitas. Esta é a motivação necessária para a aplicação de técnicas de análise temporal de redes de Petri no contexto de sistemas embarcados em tempo real.

Consideramos que nosso trabalho está inserido no contexto da verificação estática do software embarcado (D'SILVA; KROENING; WEISSENBACHER, 2008), mais especificamente na análise estática temporal. A análise estática abrange uma família de técnicas para computar automaticamente informação sobre o comportamento de um sistema sem executá-lo. Propomos a análise estática observando o comportamento temporal através da adoção da análise de tempo global de redes de Petri temporais que modelam as tarefas do software embarcado.

O uso de redes de Petri na fase de projeto permite a modelagem de tarefas e suas interações. As análises estrutural e comportamental dessas redes, nessa fase, permitem observar e identificar alguns tipos de comportamentos e problemas na interação entre essas tarefas. A análise temporal de redes de Petri permite a verificação da ordem de execução dessas tarefas e a satisfação de seus requisitos temporais. A possibilidade de adoção de padrões de projeto de redes de Petri já definidos permitem maior agilidade da modelagem dessas redes e, muitas vezes, com propriedades estruturais e comportamentais já verificadas.

A análise de tempo global também sofre do problema de explosão de estados. Nosso trabalho propõe diminuir o tamanho da rede, e consequentemente do grafo de classes, através da modelagem baseada em granularidade, composicionalidade hierárquica e padrões de projeto. Pesquisas têm sido realizadas pelo grupo de Inteligência Computacional e Métodos Formais do Departamento de Informática, UFPR, no contexto de desdobramento de redes de Petri e consequente redução desse espaço de estados.

Como nos testes comportamentais e estruturais, que são complementares, consideramos que o uso deste método quantitativo seja complementar ao uso de verificações de modelos. O método proposto não é robusto o suficiente para garantir todos os cenários de comportamento do software, como a verificação de modelos bem feita o é, enquanto a verificação de modelos ainda possui problemas em quantificar a execução e na falta de agilidade e usabilidade para o desenvolvedor, mesmo aqueles acostumados com redes de Petri.

Quanto à análise de escalonabilidade, propomos modelar uma RdP-T clássica, ou mesmo usar padrões de projeto, associando tarefas de sistemas a transições e lugares na rede. A partir das tarefas modeladas em RdP, propomos a geração do grafo de classe de estados de tempo global e a análise das sequências de disparos que satisfaçam as políticas de escalonamento PF e *earliest deadline first* (EDF).

6.1 Contribuições

Nosso trabalho é inédito ao propor um método de verificação temporal de software embarcado, baseado na técnica de tempo global de Lima *et al.* (LIMA, 2007) e (LIMA; LÜDERS; KÜNZLE, 2008) e análise de escalonabilidade de sequências de disparos. As sequências de disparos são geradas de acordo com as políticas EDF e PF sobre o grafo de classes de redes de Petri.

Nossas contribuições são:

- A proposição de um método de aplicação que, com pouco esforço, pode ser um apoio ao desenvolvimento de software embarcado mais confiável. O método proposto é baseado na separação e organização de atividades de modelagem e análise do software embarcado em redes de Petri temporais. Para a realização dessas atividades o método sugere a utilização de técnicas e ferramentas já existentes e aplicados na comunidade de engenharia de software, redes de Petri e sistemas embarcados.

- A proposição e implementação de um algoritmo de geração do grafo de classes de tempo global. A técnica de tempo global baseia-se na construção de um grafo de classes que, posteriormente, é percorrido para a geração de sequências de disparos de acordo com alguma política. Neste trabalho foi definido, implementado e validado o algoritmo que cria este grafo de classes com todas as informações temporais necessárias para a geração das sequências e sua análise de tempo global.

- Uma ampliação da técnica de tempo global a partir da definição e da utilização do método de verificação analítico. A técnica de tempo global foi definida por Lima em (LIMA, 2007) e foi refinada em outros trabalhos (LIMA; LUDERS; KUNZLE, 2006a), (LIMA; LUDERS; KUNZLE, 2006b), (MATTAR JUNIOR *et al.*, 2007), (LIMA; LÜDERS; KÜNZLE, 2008). Nesta tese definimos o tempo em uma marcação como um elemento útil para a identificação de estados do sistema e seus tempos de permanência (definição 20) e realizamos ajustes nas definições e identificação de persistência.

- A identificação e proposição de um modelo para a representação de tarefas por redes de Petri temporais. Este modelo foi desenvolvido considerando-se seu ajuste às restrições da técnica de tempo global e sua possibilidade de utilização na geração de sequências de disparo.

- A aplicação e experimentação prática do método de tempo global usando análise de escalonabilidade para as políticas EDF e PF. No contexto do método de verificação temporal proposto, nas etapas de análise temporal, propomos a análise de escalonabilidade como um tipo de análise temporal, com o objetivo de verificar o tempo de execução das tarefas e, possivelmente, de seus requisitos.

- Um estudo sobre níveis de abstração, ou de granularidade, de tarefas associadas às transições para gerar RdPs tratáveis. Durante o desenvolvimento deste trabalho de doutoramento foi

realizado um estudo sobre o nível de abstração que permitiu identificar possibilidades de modelagens de redes de Petri que permitam um tratamento temporal com maior ou menor precisão de acordo com o nível de granularidade, ou abstração, entretanto mais tratável quanto à explosão de estados.

6.2 Linhas de Trabalhos Futuros

Linhas Diretas

- Implementar uma ferramenta de visualização, simulação e análise do grafo de classes com tempos associados. Esta ferramenta permitiria a visualização de partes do grafo de acordo com os tempos de análise, inicial e final, passados pelo usuário. Ainda, permitiria visualizar em cores distintas as sequências de disparo, de acordo com a política de escalonamento a ser analisada.

- Trabalhar com técnicas de geração de dados de teste baseados no grafo de classes de tempo global. Como as redes de Petri temporais usadas com a técnica de tempo global refletem o fluxo de controle do sistema embarcado, é possível gerar dados de teste que executem elementos desse grafo, e consequentemente, dessa rede.

- Implementar a identificação do intervalo temporal de uma transição, dada uma rede de Petri temporal com tempo limite de execução. A descoberta de um intervalo temporal pode ser útil dentro de um contexto de prototipação de software, em que os requisitos temporais das tarefas estão incompletamente especificados.

- Estender a ferramenta realizando o corte (*prunning*) da árvore de busca de acordo com a política de escalonamento. A ferramenta implementada permite uma visão geral do comportamento temporal ao longo do grafo de classes. Para aumentar sua eficiência, é possível realizar cortes desse grafo de classes durante sua geração.

- Estender o modelo para representar recursos. O conceito de memória é facilmente implementado sendo associado à marcação, a partir de onde seu tamanho ser calculado quando se percorre o grafo de classes de tempo global. Pode-se tratar memória global, memória local, *buffer* e limpeza (*garbagge collection*) (BARRETO, 2005). Ou ainda, pode-se estender as redes de Petri temporais para representar recursos associados a lugares, sem que, com isso se altere a representação do fluxo de controle usada pela técnica de tempo global. Esses lugares poderiam então modelar recursos como memória e energia.

- Adoção de novas técnicas de diminuição do espaço de estados. Estudos sobre desdobramento (MCMILLAN, 1995) (ESPARZA; ROMER; VOGLER, 1996) têm sido conduzidos pelo grupo de pesquisa LIAMF-DINF-UFPR, que podem ter sua aplicação no contexto do método de verificação temporal proposto neste trabalho. Entretanto é possível que uma das vantagens do tempo global, que é sua precisão, seja em parte perdida com o desdobramento. Estudos sobre este tema devem ser conduzidos para permitir qualquer conclusão a respeito.

Linhas Indiretas

- Estudos sobre verificação de modelos das RdP geradas de acordo com o método proposto. Nosso método, apesar de ter sido construído explicitamente para a análise temporal quantitativa, não exclui a possibilidade de inclusão de atividades referentes à aplicação de métodos qualitativos, como de verificação de modelos usando redes de Petri. Tais métodos poderiam ser aplicados nas etapas 4 e 6, posteriormente à análise de tempo global.

- Estudos no contexto de sistemas embarcados com restrições de energia e de redes de sensores sem fio, na análise dos pontos de acesso. Os programas no contexto de redes sensores sem fio são pequenos, com boa aplicabilidade do método proposto neste trabalho.

- Estudar aplicações da técnica de tempo global nas áreas de: Banco de Dados, na análise de geração de consultas, Sistemas de Informação, na análise de *workflow* e Sistemas Distribuídos, para a análise de escalonamento e escalonabilidade.

APÊNDICE A – Diretivas de Tradução do Código-Fonte em C/C++ para RdP

A.1 Uma RdP Específica: PRES+

A modelagem apresentada a seguir foi definida tendo-se como base o método PRES+ (CORTES, 2001), com as adaptações necessárias para a realização da análise temporal de Tempo Global.

A.1.1 Definições da rede de Petri e suas equivalências no código C/C++

Uma expressão é uma expressão da gramática das linguagens C e C++ [1].

Sugere-se a modelagem da RdP *hierárquica* (CORTES; ELES; PENG, 2001) a partir de um código em C/C++, como apresentado a seguir:

- variáveis globais: declaração e uso.
  - Proposta 1: identificar como argumento de entrada;
  - Proposta 2: deixar implícito na rede e apresentar apenas sua definição e seu uso através de argumentos e valores de marcas;
- declaração de variáveis: para o caso do código estar em linguagem C é possível considerar uma super-transição para a declaração das variáveis. Para o caso do código estar em linguagem C++ pode-se abstrair e também considerar esta super-transição para a declaração/criação de todos os objetos, já que se deve considerar que todos os objetos em sua criação chamam métodos de criação (não apenas as classes definidas pelo usuário, mas as classes definidas pela linguagem também) ou se pode não abstrair e considerar uma transição para cada declaração de variável;

[1] <expressão> → <var> + <var>; <expressão> → <var> − <var>; <expressão> → <var> .

- "valores"como marcas de lugares p

1. **Para o cabeçalho da função**:
   (a) definir **uma transição t** representando o cabeçalho;
   (b) definir **um lugar** p_{ant} **em paralelo** para cada argumento do cabeçalho e definir o **tipo da marca** de p_{ant}, baseado no tipo do respectivo argumento do cabeçalho; e
   (c) definir **um arco** para a ligação de cada p_{ant} para t
2. **Para a declaração de variáveis/objetos**:
   (a) De acordo com a abstração desejada para a transição:
      - criar uma **super-transição S** definindo **um único lugar** p_{ant} e o **tipo da marca** de $p_{ant} = \emptyset$, que represente a declaração de todas as variáveis;
        ou
      - criar uma **transição t** definindo **um lugar** p_{ant} **para cada declaração em sequência** e definir **tipo da marca** de cada $p_{ant} = \emptyset$;

   (b) definir **um arco** para a ligação de cada p_{ant} para t; e
   (c) definir **um arco** para a ligação de cada t_{ant} para p_{ant}.
3. **Para a parte funcional**, lê cada linha de comando (separadas por ";","}","}") e para cada caso:
   (a) chamada de função (independente se é do SO ou outra):
      - criar **uma super-transição S** e associar a função a S;
      - criar **um lugar** p_{ant} para cada argumento de S;
      - definir **tipo da marca** de p_{ant} de acordo com o tipo de argumento de S;
      - definir **um arco** para a ligação de cada p_{ant} para t;
      - rotular cada arco com o nome do valor da marca; e
      - definir **um arco** para a ligação de cada t_{ant} para p_{ant};

   (b) comando de repetição **while (proposição)**:
      - criar uma **transição t** e associar o comando a t;
      - criar **um lugar** p_{ant};
      - definir **tipo da marca** de p_{ant} de acordo com o tipo de argumento de t (criando um registro, somando argumentos se necessário);

- definir **um arco** para a ligação de p_{ant} para t;
- rotular o arco com o nome do valor da marca;
- definir **um arco** para a ligação de cada t_{ant} para p_{ant};
- armazena p_{ant} como "pai do laço";
- cria t^1_{post} para **guarda Verdadeira**; e
- cria t^2_{post} para **guarda Falsa**, se existir (avaliando a proposição do while);

(c) comando de condição **if (proposição)**:

- criar uma **transição t** e associar o comando a t;
- criar **um lugar** p_{ant};
- definir **tipo da marca** de p_{ant} de acordo com o tipo de argumento de t (criando um registro, somando argumentos se necessário);
- definir **um arco** para a ligação de p_{ant} para t;
- rotular o arco com o nome do valor da marca;
- definir **um arco** para a ligação de cada t_{ant} para p_{ant};
- cria t^1_{post} para **guarda Verdadeira**;
- cria t^2_{post} para **guarda Falsa**, se existir (avaliando a proposição do if);

(d) **fechamento** } de comando de repetição *while*:

- define **um arco** de t (sendo t a última transição referente à última linha do bloco de comando while) que se liga ao lugar p "pai do laço", anterior à transição referente ao while;

(e) **fechamento** } de comando de condição *if*:

- define **um arco** de t, sendo t a última transição referente à última linha do bloco de comando if, que se liga ao lugar p;

(f) **comando simples**:

- criar uma **transição t** e associar o comando a t;
- criar **um lugar** p_{ant};
- definir **tipo da marca** de p_{ant} de acordo com o tipo de argumento de t, criando um registro e somando argumentos se necessário, i.e., de acordo com os argumentos ou variáveis existentes do lado direito das expressões de atribuições;
- definir **um arco** para a ligação de p_{ant} para t;
- rotular o arco com o nome do valor da marca; e

- definir **um arco** para a ligação de cada t_{ant} para p_{ant}.

Ao final, pode-se proceder a uma simplificação da RdP, o que provocará que comandos que não possuem interferência do fluxo de controle ficarão em um bloco e serão representados por uma transição apenas.

Diferenças entre as Modelagens em C e C++

- Declarações de objetos e de variáveis.
- Interfaces.

APÊNDICE B – Estudos sobre Granularidade na Modelagem de RdP a Partir de Código de Software Embarcado

B.1 Introdução

São poucos os trabalhos existentes na literatura a respeito da tradução do código para RdP. O que se constata é um número cada vez maior de estudos de representação de projetos de software embarcado utilizando RdP, algumas vezes com RdP's sendo geradas diretamente a partir de requisitos funcionais, como é o trabalho proposto por Barreto em (BARRETO, 2005) e outras a partir de modelos já construídos de projeto, como são os trabalhos (MACHADO; FERN; SANTOS, 2001) e (GOMES; COSTA, 2006).

Fuhs e Cannady apresentam em (FUHS; CANNADY, 2004) três níveis de abstração para a representação em RdP a partir de códigos em Java: no primeiro nível, uma rede Petri representa apenas as chamadas, a partir do método principal, o *main*, a métodos de objetos; no segundo nível representa-se a mesma informação de chamadas a métodos de objetos, mas mostram-se as estruturas de controle e execução de caminhos; e no terceiro nível todas as variáveis, instâncias de variáveis, estruturas de controle em adição às chamadas de objetos, são representadas. O nível correto de abstração é um tema muito subjetivo e para uma abstração ser útil, ela deve prover ao usuário entendimento o suficiente sem tornar-se uma duplicata exata do sistema original, considerando ainda que se o sistema fosse uma duplicata exata como em uma "tradução parte-a-parte", a utilidade geral da abstração poderia ser reduzida.

Cortes propõe em (CORTES, 2001) a implementação de sistemas embarcados a partir de projetos modelados com RdP. Neste trabalho o autor definiu um novo modelo de RdP, o modelo PRES+, onde as transições representam funções de transição que descrevem a funcionalidade do sistema a ser implementado. Essas funções permitem que o sistema seja modelado em diferentes níveis de granularidade, com transições representando desde simples operações aritméticas

até algoritmos complexos e arcos de entrada destas transições representando argumentos destas funções de transição.

Estes trabalhos são alguns dos motivadores para se estudar as diferentes possibilidades de representação de instruções por transições em redes de Petri.

B.2 Desenvolvimento

Pode-se considerar que o estado do sistema se altera a cada execução de uma instrução. Isto porque uma instrução pode representar a atribuição de um valor a uma variável, a execução de uma operação, ou a alteração da própria execução do processo, ou seja, a alteração do contador de programa, a troca de contexto ou a alteração do estado de quaisquer dos recursos citados no capítulo anterior.

O uso de transições de RdP para modelar o comportamento de sistemas implica na divisão do espaço de estados de instruções, ou atividades, em conjuntos disjuntos, de tal maneira que uma transição passa a representar um conjunto de instruções que permite a mudança de um estado do sistema para outro. Consequentemente, é o tamanho deste conjunto de atividades que dará a granularidade das transições e consequentemente do modelo.

Para dar prosseguimento à apresentação do estudo, devem-se definir que as transições t_i do conjunto T de transições de uma RdP N, serão os elementos que determinarão a granularidade, ou tamanho, do conjunto de instruções sendo executado, sendo que uma transição corresponderá a um conjunto de instruções de código em execução. O conjunto de instruções para cada estudo é definido em cada seção deste capítulo.

Ainda, para cada estudo de granularidade realizado percebeu-se a necessidade de estudar a modelagem dos pontos de sincronismo como transições. Um ponto de sincronismo pode ser considerado como um ponto de preempção ou, mais especificamente, uma chamada a um procedimento ou método do sistema operacional Kernel X que lhe repassa o controle sobre a execução das tarefas permitindo-lhe reescalonar tarefas de acordo com suas políticas. Tem-se então, que para cada item de granularidade são apresentados dois estudos: um observando pontos de sincronismo e outro sem esta observação.

Os exemplos citados nas próximas seções são baseados no código do programa que implementa uma variação do jogo *Codebreaker*, ou *Mastermind*, (WIKIPEDIA, 2008) para o módulo *eAT55* usando o sistema operacional *Kernel X* (ESYSTECH, 2008a, 2008b). Este código está

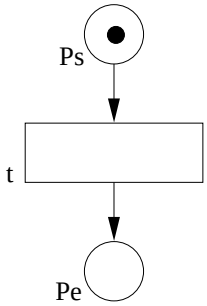

Figura 28: Um exemplo de programa ou processo como transição de RdP.

disponível no Apêndice C.

Programa ou processo como transição

Um programa inteiro pode ser considerado o conjunto de instruções passível de ser modelado por uma transição, ou seja, o conjunto de instruções é do tamanho do programa.

Para a modelagem sem os pontos de sincronismo, uma RdP N possuirá dois lugares e dois arcos: um lugar inicial p_s que possui um arco apontando para esta única transição t e um lugar final p_e que recebe um arco de t. A marcação inicial se dá com uma marca m em p_s.

Um exemplo é apresentado na Figura 28, que representa o sistema com o código exemplo apresentado no Apêndice C.

A modelagem de um programa onde os pontos de sincronismo são incluídos, sem a representação das chamadas a procedimentos ou métodos do programa, compromete a dinâmica da RdP que não conseguirá representar o comportamento de chamadas de procedimentos ou métodos. Portanto é sem sentido este tipo de representação.

Procedimento ou método como transição

O conjunto de instruções a ser representado por uma transição t é o conjunto de instruções que compõe um procedimento ou método. Para o caso de uma RdP N com x procedimentos ou métodos:

- existirão x transições t_i, cada uma representando um procedimento ou método i;
- existirá pelo menos um lugar p_i que antecede uma transição t_i;

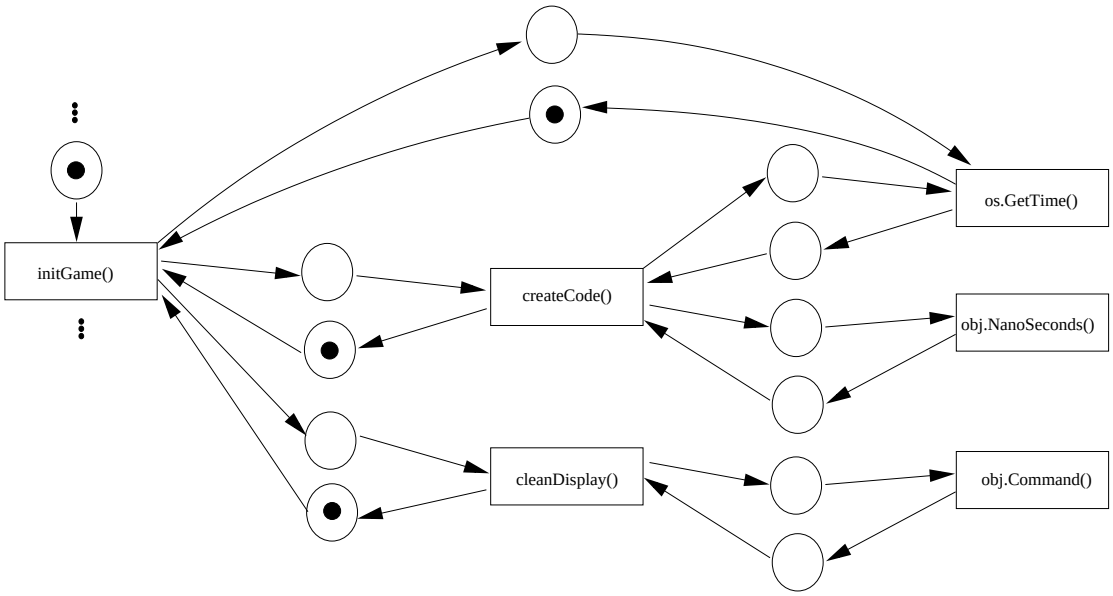

Figura 29: Exemplo de um trecho de código com procedimentos e métodos como transições.

- os arcos que conectam lugares a transições possibilitam a representação do fluxo de controle;
- uma marcação inicial possível é aquela que permite que a transição t_i representando a função *main*, para $i = 1$, seja executada, i.e., existirá uma marca m em cada lugar p_1.

Seguindo essas regras, uma sub-rede representando o trecho de código referente às linhas 54 a 86 do programa dado como exemplo é apresentada na Figura 29.

É possível perceber por este exemplo, que a integração das transições em uma sub-rede provoca uma paralelização da execução dos procedimentos que elas representam, e que não corresponde à realidade de sua execução. Isto porque esta paralelização não é desejada e não está presente no código, que exige chamadas sequenciais a estes procedimentos. Destaca-se a paralelização inserida entre os procedimentos *createCode()*, *cleanDisplay()* e *os.GetTime()*.

Para corrigir este grave problema, uma solução seria incluir novas transições que representem a sequencialidade desejada e consequentemente a ideia deste nível de granularidade seria desfeita. Ainda, o desenvolvimento desta nova solução exige um novo conceito para gerar RdP's que sejam consistentes com o fluxo de controle dos processos que é o conceito de *guarda de condição* (TRICKOVIE, 2000), (KNOKE; HOMMEL, 2005), existente na modelagem PRES+ de (CORTES, 2001). Estudos mais detalhados devem ser realizados para a modelagem neste nível de granularidade utilizando tal conceito. Na seção B.2 é apresentado como seria possível a realização de uma modelagem de expressão de comando como transição com utilização da

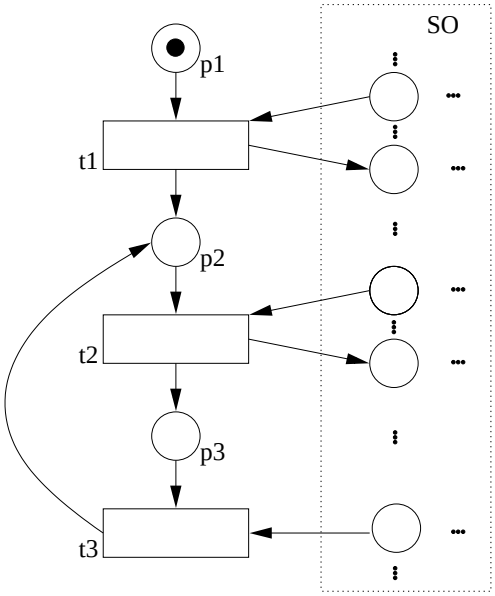

Figura 30: Um procedimento representado com seus pontos de sincronismo.

condição guarda em um nível de granularidade mais baixo.

Por outro lado, a introdução de pontos de sincronismo na representação por transições de procedimentos e métodos provoca que cada procedimento ou método que era representado apenas por uma transição deve ser representado agora pelo número de transições x referentes ao número de pontos de sincronismo *po* mais um, ou seja, $x = |po| + 1$.

Um exemplo é apresentado na Figura 30, que representa o método *threadFault()* encontrado nas linhas 166 a 182 do programa dado como exemplo. É importante notar que é necessária a modelagem de alguns comandos de fluxo de controle presentes no código, como no exemplo o retorno ao comando *while* da linha 173 através de um arco que sai da transição t3 e chega ao lugar p2.

Entretando, o problema de paralelização encontrado no exemplo anterior também continua existindo quando ocorre a integração das sub-redes que representam procedimentos ou métodos, necessitando de estudos específicos para a inclusão de condição guarda. A sugestão de correção deste problema é a mesma do primeiro exemplo desta seção.

Linha de execução de processo como transição

O interesse deste trabalho é pela modelagem de código concorrente para um único processador, portanto as referências a linhas de execução se darão sempre neste contexto.

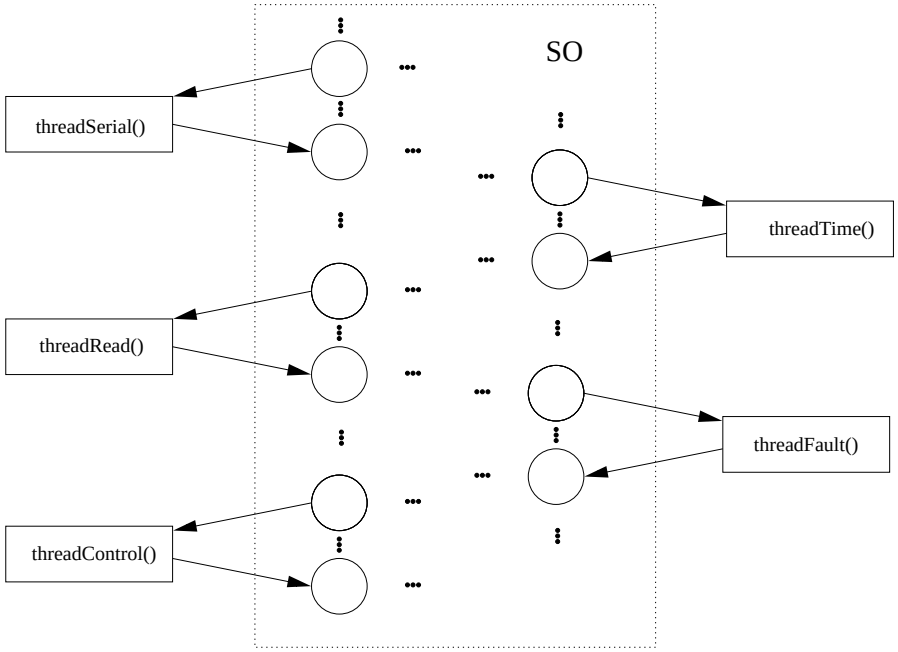

Figura 31: Exemplo de linha de execução como transição.

O conjunto de instruções a ser representado por uma transição é o conjunto de instruções que compõe uma linha de execução de processo, de acordo com a definição dada no Capítulo 2 do texto de qualificação de doutoramento. Em uma RdP representando um processo com *x* linhas de execução:

- existirão *x* transições t_i, uma para cada linha de execução fe_i;
- existirá ao menos um lugar p_i que antecede uma transição t_i;
- os arcos que conectam lugares a transições possibilitam a representação do fluxo de controle de chamadas a métodos do sistema operacional Kernel X;
- a marcação inicial depende inteiramente do escalonamento de tarefas do sistema operacional.

No exemplo utilizado neste trabalho, as linhas de execução referem-se às sequências de comandos executadas a partir das chamadas aos seguintes procedimentos, realizadas internamente pelo sistema operacional: *threadSerial(), threadRead(), threadControl(), threadTime(), threadFault()*.

A Figura 31 mostra um exemplo deste tipo de granularidade, para as linhas de execução presentes no programa dado como exemplo.

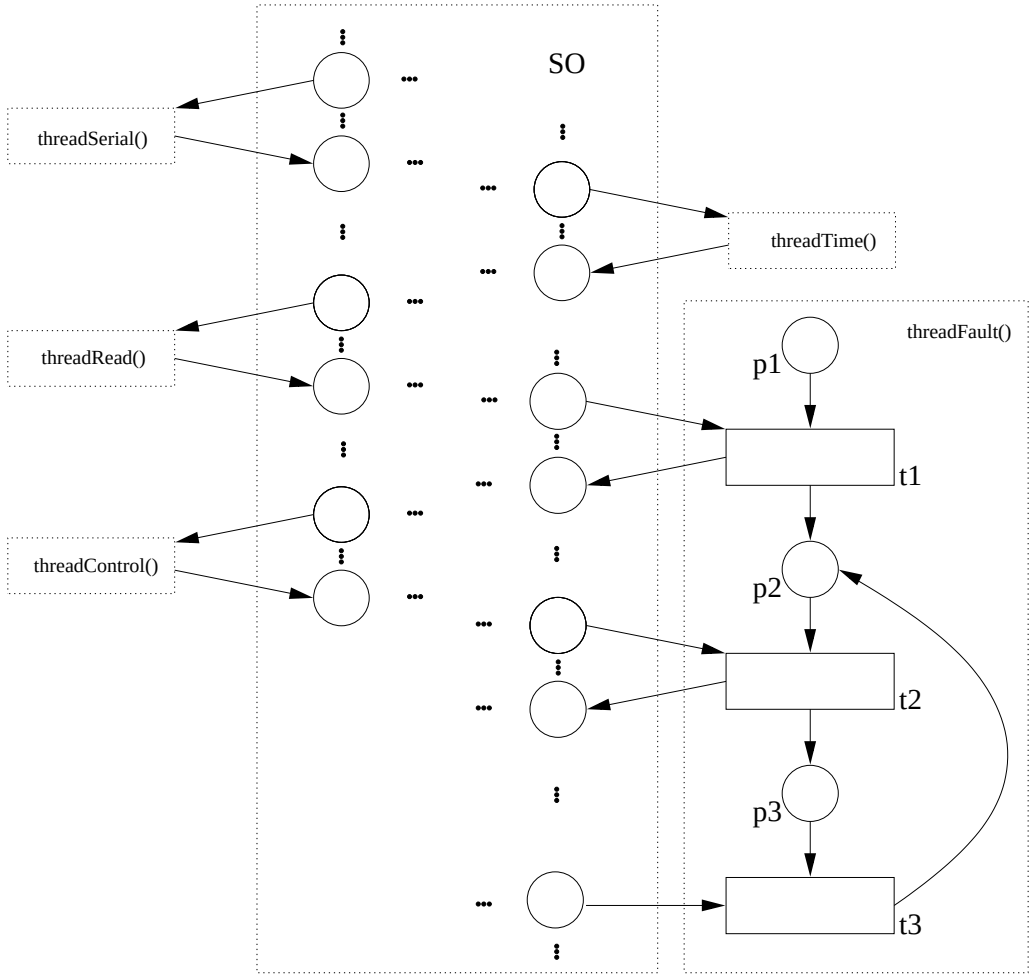

Figura 32: Exemplo de linha de execução com ponto de sincronismo como transição.

Para modelar os pontos de sincronismo, divide-se a transição que comporia este conjunto de instruções em duas ou mais transições, de acordo com o número de pontos de sincronismo. Agora, a linha de execução que era representada por uma transição deve ser representada pelo número de transições x referentes ao número de pontos de sincronismo *po* mais um, ou seja, $x = |po| + 1$.

A Figura 32 mostra um exemplo deste tipo de granularidade para a linha de execução iniciada no procedimento *threadFault()* do programa apresentado no Apêndice C.

Bloco de comandos como transição

Foram estudadas para este trabalho duas propostas para a caracterização de um bloco de comando e estas serão apresentadas a seguir.

Comandos de fluxo de controle Verificou-se ser impraticável a granularização do programa via caracterização de um bloco de comando como transição. Isto porque uma transição representaria apenas o bloco de comando de controle superior, tendo que, para se representar todo o fluxo de controle do código, desenvolver diversas redes separadas para cada nível do fluxo de controle. Por exemplo se dentro de um laço existir outro laço e dentro deste segundo laço existir uma condição: uma transição em uma rede representaria apenas o primeiro laço, sendo necessário para representar o segundo laço outra RdP e para representar a condição uma terceira RdP, que deveria ser aninhada à segunda rede e posteriormente esta rede gerada deste aninhamento deveria ser aninhada novamente na primeira rede gerada para o primeiro causando uma RdP.

Bloco de rede de Petri Barreto em sua tese de doutorado (BARRETO, 2005) apresenta um método de modelagem do projeto de sistema de tempo real crítico usando RdP, propondo a geração automática de código a partir de RdP's modeladas na fase de projeto. Este autor propõe blocos de RdP que são equivalentes a funções de concorrência, sincronização e controle temporal, e que compõem uma tarefa concorrente. A partir dos requisitos do usuário, RdP's são geradas durante o projeto e são a base para a geração de código em linguagem C++.

A seguir apresenta-se um resumo deste trabalho, sem a intenção de esgotar o tema mas com o objetivo de ilustrar qual seria a base de um futuro estudo de tradução inversa do proposto por Barreto.

Estes blocos são ilustrados e exemplificados em (BARRETO, 2005):

- bloco de chegada de tarefas periódicas: modela a invocação periódica para todas as instâncias da tarefa no período escalonado (P_s);
- bloco de estrutura de tarefas, que modela:
  - tempo de liberação, porque o sistema pode estar ocioso até alcançar todas as restrições temporais;
  - ganho e liberação de processador. A modelagem representa a utilização deste recurso por exclusão mútua, sendo utilizadas duas transições: uma específica para representar o ganho e outra para representar a liberação do processador; e
  - computação, que é modelada tanto com a política preemptiva quanto com a não preemptiva.

- bloco de conferência de tempo limite de execução (*deadline*): são estruturas elementares de redes de petri incluídas com o objetivo de detecção do tempo limite;
- bloco de envio de mensagem interprocessadores: como o autor fez sua proposta para sistemas multiprocessados, a especificação destes sistemas considera que toda comunicação interprocessadores se dá através de uma nova tarefa de comunicação. Este bloco modela esta tarefa;
- bloco de recursos, tais como processadores e barramentos: processadores e barramentos são modelados explicitamente como lugares individuais na RdP. As marcas sobre um lugar representando um processador indicam o número de processadores que estão disponíveis, todos acessando uma única memória que não é representada explicitamente, e é possível ter apenas uma marca sobre um lugar representando um barramento;
- bloco de ramificação (*fork*), que representa a criação de *n* processos concorrentes iniciando de um único processo-pai em um período de escalonamento;
- bloco de junção (*join*), que modela todas as tarefas no sistema tendo suas execuções concluídas no período de escalonamento.

Para a composição ou simplificação das redes foi proposto um conjunto de operadores sobre RdP: junção de lugar (*place merging*), refinamento serial de lugar (*serial place refinement*), adição de lugar (*place addition*), adição de arco (*arc addition*), remoção de arco (*arc removing*) e união de rede (*net union*).

Para a composição de uma tarefa, a geração é baseada nos blocos de construção:

- chegada de tarefa periódica;
- estrutura de tarefa;
- conferência de limite de tempo;
- lugares são adicionados de acordo com certas suposições; e
- aplica-se o operador *junção de lugar* para a união destas sub-redes.

Estes estudos de granularidade permitiram identificar uma possibilidade de, a partir do código concorrente, gerar RdP's utilizando os blocos propostos em (BARRETO, 2005). A ideia seria identificar nos trabalhos de Barreto quais comandos na linguagem C++ estariam associados aos blocos de RdP apresentados a seguir em termos funcionais.

Expressão de comandos como transição

Uma expressão é uma expressão da gramática das linguagens C e C++ [1].

Sugere-se a modelagem da RdP *hierárquica* (CORTES; ELES; PENG, 2001) a partir de um código em C/C++, como apresentado a seguir:

- variáveis globais: declaração e uso.
  - Proposta 1: identificar como argumento de entrada;
  - Proposta 2: deixar implícito na rede e apresentar apenas sua definição e seu uso através de argumentos e valores de marcas;
- declaração de variáveis: para o caso do código estar em linguagem C é possível considerar uma supertransição para a declaração das variáveis. Para o caso do código estar em linguagem C++ pode-se abstrair e também considerar esta supertransição para a declaração/criação de todos os objetos, já que se deve considerar que todos os objetos em sua criação chamam métodos de criação (não apenas as classes definidas pelo usuário, mas as classes definidas pela linguagem também) ou se pode não abstrair e considerar uma transição para cada declaração de variável;
- "valores"como marcas de lugares *p*

Mais detalhes desta tradução de expressões em linguagem C/C++ para RdP podem ser encontrados no Apêndice A.

B.3 Conclusões

Este texto apresenta os resultados dos estudos sobre granularidade de representação de código concorrente de software embarcado utilizando o sistema operacional Kernel X.

Quanto à modelagem apresentada na Seção B.2 a utilidade percebida é quando da necessidade de representação de uma aplicação em um contexto maior, possivelmente em um contexto do ambiente. Entretanto a representação dos pontos de sincronismo neste nível se mostraram errôneos, já que não é possível representar os fluxos de controle da aplicação.

[1] <expressão> → <var> + <var>; <expressão> → <var> − <var>; <expressão> → <var> .

Outra representação que pode levar a um entendimento errado do processo analisado é quando cada procedimento ou método existente no código é representado exclusivamente por uma transição. Este problema e sua solução foram discutidos na Seção B.2 deste texto.

A representação de linhas de execução do processo apresentada na Seção B.2 mostra-se promissora apesar do fluxo de controle interno, por comandos de condição e repetição, não ser representado. A identificação dos pontos de sincronismo, que provoca a geração de mais transições na RdP permite aproximar a modelagem ao verdadeiro comportamento do sistema.

Os estudos sobre os blocos de RdP's identificados na Seção B.2 sugerem como resultado um estudo de tradução inversa, i.e. a partir do código para a RdP, para uma etapa posterior dos trabalhos de doutoramento.

Na Seção B.2 foi apresentada uma técnica de modelagem desenvolvida para estes estudos especiais de doutorado baseadas nas redes PRES+ propostas por (CORTES; ELES; PENG, 2001). PRES+ são RdP's coloridas, hierárquicas e com condição guarda. O objetivo da tradução proposta neste trabalho é estudar a viabilidade da implementação de uma tradução comando-a-comando. É possível perceber que essa tradução é detalhada e, caso seja realizada manualmente e com grande quantidade de código, é lenta, podendo ser uma atividade tediosa e com grande possibilidade de inclusão de erros. Sugere-se a adoção de modelagem de mais alta granularidade, como as soluções apresentadas nas seções B.2 e B.2 destacando-se a modelagem de condições guarda para representar os fluxo de controle que não seja possível representar com a teoria de redes de Petri clássicas. Caso sejam percebidas necessidades de representação mais detalhada, as traduções sugeridas na Seção B.2 devem tender à automatização.

APÊNDICE C – Código em C do Jogo Codebreaker

```
#include "stdio.h"
#include "DDLX\DDLX_atmel_devices.h"
#include "DDLX\DDLX_atmel_rtc.h"
#include "DDLX\DDLX_LCD.h"
#include "eSysTech\DD_keyb.h"
#include "eSysTech\DD_serial_232.h"
#include "X\xl.h"
#include "X\x_shell.h"

X os(15, 5, 0);
CX_CommPort dbg_comm(8*1024, 8, 32, 300);

#ifdef X_TRACE
CX_Trace x_trace(11, TR_END);
#endif

enum EMessageType {KEY_UP = 1, KEY_DOWN, KEY_LEFT, KEY_RIGHT, KEY_ENTER,
IS_BEGIN, VERIFY, IS_FAULT};

Byte   g_led = 0;
Byte   g_buffer[3072];
Int16  g_ind_buffer = 0;
Byte   g_code[4];
Byte   g_move[4];
Byte   g_result[4];
Card8  g_temptsN;
Card8  g_position;
Card8  g_prev_position;
AbsoluteTime g_initTime;

const Card8 CODESIZE = 4;
const Card8 TEMPTS_LIMIT = 8;
const Card16 NUMCOLORS = 9; // quantidade de algarismos possiveis para uma
posicao do codigo

// Declaracoes das funcoes
void initDisplay();
void clearDisplay();
void initMove(Card8 temptsN, Byte* move);
void showResult(Byte* move, Byte* result, Card8 temptsN);
void showTime(char* gameTime);
void updateCursor(Card8 position, Card8 prevPosition, Card8 temptsN);
void updateContent(Byte moveContent, Card8 position, Card8 temptsN);
void showFault();
void createCode();
void initGame();
void won();
void Oracle(Byte* result, Byte* move);

// Definicoes das funcoes
void initGame(){
  createCode();              // cria o codigo
  clearDisplay();            // limpa o display
  g_temptsN=1;               // inicia contagem de tentativas
```

```
55    g_initTime = os.GetTime(); // zera o tempo
56    initMove(g_temptsN,g_move); // inicia o jogo e suas posicoes no display
57  };
58
59  void createCode(){
60    Card32 nanosec;
61    Card8 ok, i, j;
62    AbsoluteTime seed;
63    ok = 0;
64    while(!ok) {
65      seed = os.GetTime();
66      nanosec = (Card32) seed.NanoSeconds();
67      for(i = 0; i < CODESIZE; i++) {
68        g_code[i] = nanosec % NUMCOLORS;
69        nanosec = nanosec / NUMCOLORS;
70      }
71      ok = 1;
72      for(i = 0; i < CODESIZE - 1; i++)
73        for(j = i + 1; j < CODESIZE; j++)
74          if(g_code[i] == g_code[j])
75            ok = 0;
76    }
77  };
78
79  void initDisplay() {
80    g_lcd.Command(LCD_DISP_ON);
81    g_lcd.Command(LCD_CLEAR);
82  };
83
84  void clearDisplay() {
85    g_lcd.Command(LCD_CLEAR);
86  };
87
88  void won(){
89    Card64 i;
90    Card8  j;
91    char turn[3];
92    g_lcd.GoToLineCol(2,18);
93    for(i=0; i < 500; i++){
94    }
95    initGame();
96  }
97
98  void lost(){
99    g_lcd.GoToLineCol(2,18);
100   g_lcd.WriteString("L");
101 }
102
103 void initMove(Card8 temptsN, Byte* move){
104   Card8 i;
105   Card8 moveCol  = 2 * (temptsN - 1); // coluna do conteudo da jogada
106   char num_tempt[3];
107   char moveContent[2];
108   g_position = 0;
109   g_prev_position = g_position;
110   sprintf(num_tempt, "%02d", temptsN%100);
111   for (i=0; i < CODESIZE; i++){
112     move[i] = 0;
113     sprintf(moveContent, "%d", move[i]);
114     g_lcd.GoToLineCol(i,moveCol);  // escreve zeros
115     g_lcd.WriteString(moveContent);
116   };
117   g_lcd.GoToLineCol(0,moveCol+1);  // volta o cursor para o inicio
118   g_lcd.WriteString("*"); // apresenta o cursor
119   g_lcd.GoToLineCol(1,18);
120   g_lcd.WriteString(num_tempt);
121 };
122
123 void showResult(Byte* move, Byte* result, Card8 temptsN){
124   char num_tempt[3];
125   char turn[3];
```

```
126     Card8 i;
127     Card8 resultCol  = (2 * temptsN) - 1; // coluna do resultado - 1 (repete move)
128     sprintf(num_tempt,"%02d",temptsN);
129     for (i=0; i < CODESIZE; i++){
130       g_lcd.GoToLineCol(i,resultCol);
131       sprintf(turn,"%c",result[i]);
132       g_lcd.WriteString(turn);
133     };
134     g_lcd.GoToLineCol(1,18);
135     g_lcd.WriteString(num_tempt);
136   }
137
138   void showTime(char* gameTime){
139     g_lcd.GoToLineCol(0,17);
140     g_lcd.WriteString(gameTime);
141   }
142
143   void updateCursor(Card8 position, Card8 prev_position, Card8 temptsN){
144     Card8 cursorCol  = (2 * temptsN) - 1; // coluna do cursor
145     g_lcd.GoToLineCol(prev_position, cursorCol);
146     g_lcd.WriteString(" ");
147     g_lcd.GoToLineCol(position, cursorCol);
148     g_lcd.WriteString("*");
149   }
150
151   void updateContent(Byte moveContent, Card8 position, Card8 temptsN){
152     Card8 moveCol  = 2 * (temptsN - 1); // coluna do conteudo da jogada
153     char content[2];
154     sprintf(content,"%d",moveContent);
155     g_lcd.GoToLineCol(position, moveCol);
156     g_lcd.WriteString(content);
157   }
158
159   void showFault(){
160     g_lcd.GoToLineCol(0,17);
161     g_lcd.WriteString("999");
162     g_lcd.GoToLineCol(1,18);
163     g_lcd.WriteString("99");
164   };
165
166   // Thread Fault - Mostra erro do dispositivo externo no display
167   // Esporadica, executa somente se chegar uma mensagem
168   void threadFault(Word arg1, Word arg2) {
169     Card8 *led_ptr = (Card8 *) 0x7300000;
170     Card32 command;
171     PutMsg msg;
172     os.PeriodicMsg(10000*MSEC);
173     while (1) {
174       os.Receive(&msg, sizeof(PutMsg));
175       command = msg.command;
176       g_led ^= 4;
177       *led_ptr = g_led;
178       if (command == IS_FAULT) {
179         showFault();
180       }
181     }
182   }
183
184   // Thread Time - Mostra a hora no display
185   // Periodica
186   void threadTime(Word arg1, Word arg2) {
187     Card8 *led_ptr = (Card8 *) 0x7300000;
188     PutMsg msg;
189     char strTime[4];
190     AbsoluteTime nowTime, initTime;
191     os.PeriodicMsg(1000*MSEC);
192     while (1) {
193       os.Receive(&msg, sizeof(PutMsg));
194       g_led ^= 8;
195       *led_ptr = g_led;
196       nowTime = os.GetTime();
```

```
197       sprintf(strTime, "%03d", (nowTime.Seconds() - g_initTime.Seconds())%1000);
198       showTime(strTime);
199     }
200   }
201
202   // threadSerial -  le dados da porta serial e coloca no buffer da aplicacao
203   // Periodica, trabalha diretamente com o Hardware
204   void threadSerial(Word arg1, Word arg2) {
205     Card8 *led_ptr = (Card8 *) 0x7300000;
206     Int16 aval, num_read;
207     PutMsg msg;
208     os.PeriodicMsg(50*MSEC);
209     num_read = 0;
210     while (1) {
211       os.Receive(&msg, sizeof(PutMsg));
212       g_led ^= 1;
213       *led_ptr = g_led;
214       aval = g_serial_232.StartRx();
215       num_read = g_serial_232.ReadBytes(&(g_buffer[g_ind_buffer]), aval);
216       g_ind_buffer = g_ind_buffer + num_read;
217     }
218   }
219
220   //Thread Read - le dados do vetor e identifica se falha
221   void threadRead(Word arg1, Word arg2) {
222     Card8 *led_ptr = (Card8 *) 0x7300000;
223     Int16 i, word_length;
224     PutMsg msg;
225     os.PeriodicMsg(10*MSEC);
226     while (1) {
227       os.Receive(&msg, sizeof(PutMsg));
228       g_led ^= 2;
229       *led_ptr = g_led;
230       if ((g_ind_buffer != 0) && (g_buffer[0] <= g_ind_buffer)) {
231         word_length = g_buffer[0];
232         if ((char) g_buffer[1] == 'X') {
233           msg.command = IS_FAULT;
234           os.Put(os.GetTId("Fault"), (PutMsg *) &msg);
235         }
236         else
237         {
238           msg.command =  g_buffer[2] - 48;
239           os.Put(os.GetTId("Control"), (PutMsg *) &msg);
240         }
241         for (i=word_length; i<g_ind_buffer; i++) {
242           g_buffer[i-word_length] = g_buffer[i];
243         }
244         g_ind_buffer -= word_length;
245       }
246     }
247   }
248
249   // Thread Control - identifica teclas digitadas
250   // Periodica
251   void threadControl(Word arg1, Word arg2) {
252     Card8 *led_ptr = (Card8 *) 0x7300000;
253     PutMsg msg;
254     os.PeriodicMsg(10*MSEC);
255     while (1) {
256       os.Receive(&msg, sizeof(PutMsg));
257       g_led ^= 16;
258       *led_ptr = g_led;
259       switch(msg.command){
260       case IS_BEGIN:
261         break;
262       case KEY_UP:
263         if (g_position == 0) {  // se topo, encerra jogada
264           g_prev_position = g_position;
265           g_position=3;
266         }
267         else {
```

```
268           g_prev_position = g_position;
269           g_position--;
270         }
271         updateCursor(g_position,g_prev_position,g_temptsN);
272         break;
273       case KEY_DOWN:
274         if (g_position == 3){   // se raso, encerra jogada
275           g_prev_position = g_position;
276           g_position=0;
277         }
278         else {
279           g_prev_position = g_position;
280           g_position++;
281         }
282         updateCursor(g_position,g_prev_position,g_temptsN);
283         break;
284       case KEY_RIGHT:
285         if (g_move[g_position] == NUMCOLORS) // se lim. sup., zera conteudo
286           g_move[g_position]=0;
287         else
288           g_move[g_position]++;
289         updateContent(g_move[g_position], g_position, g_temptsN);
290         break;
291       case KEY_LEFT:
292         if (g_move[g_position] == 0) // se lim. inf., zera conteudo
293           g_move[g_position]=9;
294         else
295           g_move[g_position]--;
296         updateContent(g_move[g_position], g_position, g_temptsN);
297         break;
298       case KEY_ENTER:
299         Oracle(g_result, g_move);
300         showResult(g_move, g_result,g_temptsN);
301         if    ((g_result[0] == '+') && (g_result[1] == '+')
302             && (g_result[2] == '+') && (g_result[3] == '+')){
303            won();
304         }
305         else if (g_temptsN == TEMPTS_LIMIT) {
306            lost();
307         }
308         else {
309           g_temptsN++;
310           initMove(g_temptsN, g_move);
311         }
312         break;
313       }
314     }
315   }
316
317   // Verifica jogada
318   void Oracle(Byte* result, Byte* move){
319     Card8 i,position;
320     Card8 *led_ptr = (Card8 *) 0x7300000;
321     g_led ^= 32;
322     *led_ptr = g_led;
323     for(i=0; i< CODESIZE; i++){
324       result[i]=' '; // em branco
325     };
326     for(position = 0; position < CODESIZE; position++){
327       if (move[position] == g_code[position]) {
328         result[position] =  '+';
329       }
330       else {
331         for(i=0; i< CODESIZE; i++){
332           if (move[position] == g_code[i]) {
333             result[position] =  '-';
334           }
335           else if ((result[position] != 'x')&&(result[position] != '-')){
336             result[position] =  'x';
337           }
338         }
```

```
339       }
340     }
341   }
342
343   int main() {
344     *(Byte*) 0x7300000 = 0x0;
345     os.Init();
346     g_lcd.Init();
347     g_serial_232.Init(38400);
348     initGame();
349     os.CreateThread(threadSerial, 0, 0, "Serial", 1024,
350                 ARM_CODE | FIQ_ENABLE | IRQ_ENABLE, 7);
351     os.CreateThread(threadRead, 0, 0, "Read", 1024,
352                 ARM_CODE | FIQ_ENABLE | IRQ_ENABLE, 7);
353     os.CreateThread(threadControl, 0, 0, "Control", 1024,
354                 ARM_CODE | FIQ_ENABLE | IRQ_ENABLE, 7);
355     os.CreateThread(threadTime, 0, 0, "Time", 1024,
356                 ARM_CODE | FIQ_ENABLE | IRQ_ENABLE, 7);
357     os.CreateThread(threadFault, 0, 0, "Fault", 1024,
358                 ARM_CODE | FIQ_ENABLE | IRQ_ENABLE, 7);
359     os.Start();
360   }
```

Referências Bibliográficas

AALST, W. M. P. van der. Petri net based scheduling. Eindhoven University of Technology, 1995. Disponível em: <http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.33.3601>.

ALEXANDER, C. *et al*. *A Pattern Language*. New York, NY (USA): Oxford University Press, 1977.

ALLEN, J. F. Maintaining knowledge about temporal intervals. *Commun. ACM*, ACM, New York, NY, USA, v. 26, n. 11, p. 832–843, 1983. ISSN 0001-0782.

AMATRIAIN, X. *An Object-Oriented Metamodel for Digital Signal Processing with a focus on Audio and Music*. Tese (Doutorado) — Universitat Pompeu Fabra, Barcelona, Spain, 2004.

ARTEMIS EUROPEAN PLATFORM ON INTELLIGENT EMBEDDED SYSTEMS. *Advanced Research and Technology for Embedded Intelligent Systems - Embedded Systems Technology Platform*. Session proceedings compiled by Bob Malcolm.

ARTIST2. *ARTIST2 South-American School for Embedded Systems 2008*. Universidade Federal de Santa Catarina, Florianopolis, Brazil: [s.n.], August 25-29 2008. Notas de aula. Organised and funded by Artist. Disponível em: <http://www.artist-embedded.org/artist/Objectives,1365.html>.

BAILEY, B. *Dealing with SoC hardware/software design complexity with scalable verification methods*. 2007. [*Online*; acesso em 03/04/2010]. Disponível em: <`http://www.embedded.com/columns/technicalinsights/199202018?_requestid=787601`>.

BARRETO, R. *A Time Petri Net-Based Methodology for Embedded Hard Real-Time Software Synthesis*. Tese (Doutorado) — Centro de Informática - UFPE, April 2005.

BARRETO, R. S.; MACIEL, P. R. M.; CAVALCANTE, S. A modeling methodology and pre-run-time scheduling for embedded real-time software. In: *SBAC-PAD*. [S.l.]: IEEE Computer Society, 2003. p. 72–81. ISBN 0-7695-2046-4.

BARROS, J. P. M. P. R. e. *CpPNeTS: uma Classe de Redes de Petri de Alto-nível. Implementação de um sistema de suporte à sua aplicação e análise*. Dissertação (Mestrado) — UNIVERSIDADE NOVA DE LISBOA, 1996.

BERRY, G. The foundations of esterel. MIT Press, Cambridge, MA, USA, p. 425–454, 2000.

BERTHOMIEU, B.; DIAZ, M. Modeling and verification of time dependent systems using time Petri nets. *IEEE Trans. Softw. Eng.*, IEEE Press, Piscataway, NJ, USA, v. 17, n. 3, p. 259–273, 1991. ISSN 0098-5589.

BERTHOMIEU, B.; MENASCHE, M. A state enumeration approach for analyzing time petri nets. In: *3rd European Workshop on Applications and Theory of Petri Nets*. Varenna, Italy: [s.n.], 1982.

BERTHOMIEU, B.; MENASCHE, M. An enumerative approach for analyzing time petri nets. In: *IFIP Congress*. Paris: [s.n.], 1983.

BERTHOMIEU, B.; VERNADAT, F. Time petri nets analysis with tina. In: *Proceedings of 3rd Int. Conf. on The Quantitative Evaluation of Systems (QEST 2006)*. [S.l.]: IEEE Computer Society, 2006.

BERTHOMIEU, P.-O. R. B.; VERNADAT, F. The tool tina – construction of abstract state spaces for petri nets and time petri nets. *International Journal of Production Research*, v. 42, n. 14, July 2004.

BOOCH, G.; RUMBAUGH, J.; JACOBSON, I. *The Unified Modeling Language User Guide*. [S.l.]: Addison-Wesley, 1998. ISBN 0-201-57168-4.

BOOTH, T. L. *Sequential Machines and Automata Theory*. New York: J. Wiley & Sons, 1967.

BUSS, A. H. A tutorial on discrete-event modeling with simulation graphs. In: *WSC '95: Proceedings of the 27th conference on Winter simulation*. Washington, DC, USA: IEEE Computer Society, 1995. p. 74–81. ISBN 0-7803-3018-8.

BUTTAZZO, G. C. *Hard Real-time Computing Systems: Predictable Scheduling Algorithms And Applications (Real-Time Systems Series)*. Santa Clara, CA, USA: Springer-Verlag TELOS, 2004. ISBN 0387231374.

CADAMURO, J. *Dyretiva: Um Método para a Verificação das Restrições Temporais em Sistemas Embarcados*. Tese (Doutorado) — Curso de Pós Graduação Em Engenharia Elétrica e Industrial - Universidade Tecnológica Federal do Paraná, 2007.

CARDOSO, J.; VALETTE, R. *Redes de Petri*. [S.l.]: Editora da Universidade Federal de Santa Catarina, 1997.

CASPI, P. *et al.* Lustre: a declarative language for real-time programming. In: *POPL '87: Proceedings of the 14th ACM SIGACT-SIGPLAN symposium on Principles of programming languages*. New York, NY, USA: ACM, 1987. p. 178–188. ISBN 0-89791-215-2.

CHIODO, M. *et al.* Hardware-software codesign of embedded systems. *Micro, IEEE*, v. 14, n. 4, p. 26–36, 1994. Disponível em: <http://dx.doi.org/10.1109/40.296155>.

CIMATTI, A. *et al.* Integrating bdd-based and sat-based symbolic model checking. In: *FroCoS '02: Proceedings of the 4th International Workshop on Frontiers of Combining Systems*. London, UK: Springer-Verlag, 2002. p. 49–56. ISBN 3-540-43381-3.

CLARKE, E. *et al. Verification Tools for Embedded Systems*. 2000.

CLARKE, E.; SCHLINGLOFF, H. Handbook of automated reasoning. In: ____. [S.l.]: Elsevier Science, 2001. II, cap. 24. Model checking, p. pages 1635–1790.

CLARKE, E. M.; GRUNBERG, O.; PELED, D. *Model checking*. Cambridge, England: The MIT Press, 1999.

CORBETT, J. C. Evaluating deadlock detection methods for concurrent software. *IEEE Transactions on Software Engineering*, v. 22, p. 161–180, 1996.

CORTADELLA, J. *et al.* Deriving petri nets from finite transition systems. *IEEE Trans. Comput.*, IEEE Computer Society, Washington, DC, USA, v. 47, n. 8, p. 859–882, 1998. ISSN 0018-9340.

CORTADELLA, J. *et al.* Quasi-static scheduling of independent tasks for reactive systems. In: *Application and Theory of Petri Nets 2002*. [S.l.]: Springer-Verlag, 2002, (Lecture Notes in Computer Science, v. 2360). p. 80–99.

CORTES, L. A. *A Petri Net based Modeling and Verification Technique for Real-Time Embedded Systems*. Dissertação (Mestrado) — Dept. of Computer and Information Science, Linköping University, Sweden, December 2001. Licentiate Thesis No. 919.

CORTES, L. A. *Verification and Scheduling Techniques for Real-Time Embedded Systems*. Tese (Doutorado) — Dept. of Computer and Information Science, Linköping University, Sweden, 2005. Dissertation No. 920.

CORTES, L. A.; ELES, P.; PENG, Z. Verification of embedded systems using a Petri net based representation. In: *ISSS '00: Proceedings of the 13th international symposium on System synthesis*. Washington, DC, USA: IEEE Computer Society, 2000. p. 149–155. ISBN 1080-1082.

CORTES, L. A.; ELES, P.; PENG, Z. *Hierarchies for the Modeling and Verification of Embedded Systems*. Sweden, February 2001. SAVE Project Report.

COTTET, F. *et al. Scheduling in real-time systems* . [S.l.]: John Wiley & Sons, 2002. 282 p. ISBN 978-0-470-84766-4.

DAHR, M. *et al. NET CASE: Towards a Petri Net Based Technique for the Development of Expert/Database Systems*. [S.l.], 1994.

DELATOUR, J.; PALUDETTO, M. UML/pno: A way to merge UML and Petri net objects for the analysis of real-time systems. In: *ECOOP '98: Workshop ion on Object-Oriented Technology*. London, UK: Springer-Verlag, 1998. p. 511–514. ISBN 3-540-65460-7.

DILL, D. L. Timing assumptions and verification of finite-state concurrent systems. In: *Proceedings of the International Workshop on Automatic Verification Methods for Finite State Systems*. London, UK: Springer-Verlag, 1990. p. 197–212. ISBN 3-540-52148-8.

D'SILVA, V.; KROENING, D.; WEISSENBACHER, G. A survey of automated techniques for formal software verification. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, IEEE, v. 27, n. 7, p. 1165–1178, July 2008.

EBERT, C.; JONES, C. Embedded software: Facts, figures, and future. *Computer*, IEEE Computer Society, Los Alamitos, CA, USA, v. 42, p. 42–52, 2009. ISSN 0018-9162.

EDWARDS, S. *et al.* Design of embedded systems: Formal models, validation, and synthesis. In: *Proceedings of the IEEE*. [S.l.: s.n.], 1997. p. 366–390.

ESPARZA, J.; HELJANKO, K. *Unfoldings –A Partial-Order Approach to Model Checking*. Springer-Verlag, 2008. (EATCS Monographs in Theoretical Computer Science). Disponível em: <http://www7.in.tum.de/ esparza/bookunf.html>.

ESPARZA, J.; ROMER, S.; VOGLER, W. An improvement of McMillan's unfolding algorithm. In: *Tools and Algorithms for Construction and Analysis of Systems*. [s.n.], 1996. p. 87–106. Disponível em: <citeseer.ist.psu.edu/article/esparza96improvement.html>.

ESYSTECH. *Kit eAT55*. 2008. Http://www.esystech.com.br/produtos/hard/KiteAT55.php [*Online*; acesso em 09/12/2008]. Disponível em: <http://www.esystech.com.br>.

ESYSTECH. *X Real-Time Kernel*. 2008. Http://www.esystech.com.br/produtos/XKernel/XKernel.php [*Online*; acesso em 09/12/2008]. Disponível em: <http://www.esystech.com.br>.

FAROOQ, U.; LAM, C. P.; LI, H. Transformation methodology for uml 2.0 activity diagram into colored petri nets. In: *ACST'07: Proceedings of the third conference on IASTED International Conference*. Anaheim, CA, USA: ACTA Press, 2007. p. 128–133.

FUHS, J.; CANNADY, J. An automated approach in reverse engineering java applications using Petri nets. In: *SoutheastCon, 2004. Proceedings. IEEE*. [S.l.: s.n.], 2004. p. 469–487. Current Version Published: 2004-08-24.

FURFARO, A.; NIGRO, L. Modelling and schedulability analysis of real-time sequence patterns using time petri nets and uppaal. In: *Proceedings of the International Multiconference on Computer Science and Information Technology*. Wisła, Poland: [s.n.], 2007. p. 821–835. ISSN 1896-7094.

GAMBIER, A. Real-time control systems: A tutorial. In: *Control Conference, 2004. 5th Asian*. [S.l.: s.n.], 2004. p. 1024.

GANSSLE, J. G. *Art of Designing Embedded Systems*. Newton, MA, USA: Butterworth-Heinemann, 1999. ISBN 0750698691.

GARMAN, J. R. The "bug"heard 'round the world: discussion of the software problem which delayed the first shuttle orbital flight. *SIGSOFT Softw. Eng. Notes*, ACM, New York, NY, USA, v. 6, n. 5, p. 3–10, 1981. ISSN 0163-5948.

GEHRKE, T.; GOLTZ, U.; WEHRHEIM, H. *The Dynamic Models of UML: Towards a Semantics and its Application in the Development Process*. Germany, 1998.

GILL, A. *Introduction to the Theory of Finite-state Machines*. [S.l.]: McGraw-Hill, 1962.

GOMAA, H. *Designing Concurrent, Distributed, and Real-Time Applications with Uml*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2000. ISBN 0201657937.

GOMES, L.; COSTA, A. Petri nets as supporting formalism within embedded systems co-design. In: *International Symposium on Industrial Embedded Systems, 2006.* [S.l.: s.n.], 2006. p. 1–4.

GRAPHVIZ. 2010. *Online*; acesso em 27/06/2010. Disponível em: <http://www.graphviz.org/>.

HAREL, D. Statecharts: A visual formalism for complex systems. *Science of Computer Programming*, v. 8, n. 3, p. 231–274, June 1987. Disponível em: <http://citeseer.ist.psu.edu/547939.html>.

HATLEY, D. J.; PIRBHAI, I. *Strategies for Real-Time System Specification*. [S.l.]: Dorset House, 1987.

HENNESSY, J.; PATTERSON, D. *Computer Architecture: A Quantitative Approach*. [S.l.]: Morgan Kaufmann, 2006. (Series in Computer Architecture and Design). ISBN 0123704901.

HENZINGER, T.; SIFAKIS, J. The embedded systems design challenge. In: SPRINGER. *Proceedings of the 14th International Symposium on Formal Methods (FM), Lecture Notes in Computer Science*. 2006. Disponível em: <http://chess.eecs.berkeley.edu/pubs/264.html>.

HENZINGER, T. A.; HO, P.-H.; WONG-TOI, H. Hytech: A model checker for hybrid systems. *Software Tools for Technology Transfer*, v. 1, p. 460–463, 1997.

HOPCROFT, J. E.; MOTWANI, R.; ULLMAN, J. D. *Introduction to Automata Theory, Languages, and Computation*. 2nd. ed. Addison Wesley, 2000. Hardcover. ISBN 0201441241. Disponível em: <http://www.amazon.com/exec/obidos/redirect?tag=citeulike07-20&path=ASIN/020144 1241>.

HSIUNG, P.-A.; SU, F.-S. Synthesis of real-time embedded software by timed quasi-static scheduling. In: *VLSI Design*. [S.l.]: IEEE Computer Society, 2003. p. 579–584. ISBN 0-7695-1868-0.

HU, L.; GORTON, I. *Performance Evaluation for Parallel Systems: A Survey*. 1997. Technical Report UNSW-CSE-TR-9707, Department of Computer Systems, University of NSW, Sydney, Australia. Disponível em: <citeseer.ist.psu.edu/article/hu97performance.html>.

IEEE. *1364-2001 IEEE Standard Verilog Hardware Description Language*. [S.l.], 2001.

IEEE. *1012-2004 IEEE Standard for Software Verificiation and Validation*. [S.l.], 2004.

IEEE. *1076-2008 IEEE Standard VHDL Language Reference Manual*. [S.l.], 2008.

JACOBSON, I.; BOOCH, G.; RUMBAUGH, J. *The Unified Software Development Process*. [S.l.]: Addison-Wesley, 1999. ISBN 0201571692.

JANTSCH, A. *Models of computation in embedded system design (50 min)*. September 2001. [*arquivo PDF*; Acesso em 03/03/2009]. Disponível em: <`http://web.it.kth.se/~axel/preslist.html`>.

KAHN, G. The semantics of a simple language for parallel programming. In: ROSENFELD, J. L. (Ed.). *Information Processing '74: Proceedings of the IFIP Congress*. New York, NY: North-Holland, 1974. p. 471–475.

KAVI, K. M.; MOSHTAGHI, A.; CHEN, D.-J. Modeling multithreaded applications using Petri nets. *Int. J. Parallel Program.*, Kluwer Academic Publishers, Norwell, MA, USA, v. 30, n. 5, p. 353–371, 2002. ISSN 0885-7458.

KNOKE, M.; HOMMEL, G. Dealing with global guards in a distributed simulation of colored Petri nets. In: *DS-RT '05: Proceedings of the 9th IEEE International Symposium on Distributed Simulation and Real-Time Applications*. Washington, DC, USA: IEEE Computer Society, 2005. p. 51–60. ISBN 0-7695-2462-1.

LAAS. *TINA - Time Petri Net Analyzer*. 2010. *Online*; acesso em 27/06/2010. Disponível em: <http://www.laas.fr/tina>.

LAVAGNO, L. *et al.* Models of computation for embedded system design. In: *in System-Level Synthesis*. [S.l.]: Kluwer Academic Publishers, 1998. p. 45–102.

LILIUS, J. Efficient state space search for time petri nets. In: *Electronic Notes in Theoretical Computer Science*. [S.l.]: Springer-Verlag, 1999.

LIMA, E. A. *Análise e Aplicações de Redes de Petri Temporais: Uma Abordagem via Álgebra Intervalar*. Tese (Doutorado) — Pós Graduação Em Engenharia Elétrica e Industrial - Universidade Tecnológica Federal do Paraná, fevereiro 2007.

LIMA, E. A.; LUDERS, R.; KUNZLE, L. A. Análise de redes de petri temporais usando tempo global. In: *In: VII Simpósio Brasileiro de Automação Inteligente - SBAI*. [S.l.: s.n.], 2005.

LIMA, E. A.; LUDERS, R.; KUNZLE, L. A. Análise de redes de Petri temporais via Álgebra intervalar. In: *In: XVI Congresso Brasileiro de Automática*. Salvador - Bahia: [s.n.], 2006.

LIMA, E. A.; LUDERS, R.; KUNZLE, L. A. Interval analysis of time Petri nets. In: *In: 4th CESA Multiconference 4th CESA Multiconference on Computational Engineering in Systems Applications*. Beijing - China: [s.n.], 2006.

LIMA, E. A.; LÜDERS, R.; KÜNZLE, L. A. Uma abordagem intervalar para a caracterização de intervalos de disparo em redes de petri temporais. *SBA. Sociedade Brasileira de Automática*, v. 19, n. 4, p. 379, 2008. ISSN ISSN/ISBN: 01031759. In Portuguese, http://dx.doi.org/10.1590/S0103-17592008000400002.

LIME, D.; ROUX, O. H. Expressiveness and analysis of scheduling extended time Petrinets. In: *5th IFAC Int. Conf. on Fieldbus Systems and Applications,(FET'03)*. Aveiro, Portugal: Elsevier Science, 2003. p. 193–202.

LIME, D.; ROUX, O. H. Formal verification of real-time systems with preemptive scheduling. *Journal of Real-Time Systems*, Springer, v. 41, n. 2, p. 118–151, 2009.

LIU, C.; DONG, X. An improved quasi-static scheduling algorithm for mixed data-control embedded software. *Journal of Applied Sciences*, v. 6, n. 7, p. 1571–1575, 2006. ISSN 1812-5654.

LIU, C. L.; LAYLAND, J. W. Scheduling algorithms for multiprogramming in a hard-real-time environment. *J. ACM*, ACM, New York, NY, USA, v. 20, n. 1, p. 46–61, 1973. ISSN 0004-5411.

LIU, J. W. S. W. *Real-Time Systems*. Upper Saddle River, NJ, USA: Prentice Hall PTR, 2000. ISBN 0130996513.

LóPEZ-GRAO, J. P.; MERSEGUER, J.; CAMPOS, J. From uml activity diagrams to stochastic petri nets: application to software performance engineering. *SIGSOFT Softw. Eng. Notes*, ACM, New York, NY, USA, v. 29, n. 1, p. 25–36, 2004. ISSN 0163-5948.

MACHADO, R. J.; FERN, J. M.; SANTOS, H. D. A methodology for complex embedded systems design: Petri nets within a UML approach. In: *UML Approach?, Architecture and Design of Distributed Embedded Systems, B. Kleinjohann (editor), Kluwer A.P*. [S.l.]: Kluwer Academic Publishers, 2001. p. 1–10.

MARTIN. *UML Distilled: Applying the Standard Object Modeling Language*. [S.l.]: Addison Wesley, 1997. Paperback. ISBN 0201325632.

MARZO, G. *Stepwise Refinement of Formal Specifications Based on Logical Formulae: from CO-OPN/2 Specifications to Java Programs*. Tese (Doutorado) — Swiss Federal Institute of Technology in Lausanne, 1999.

MATTAR JUNIOR, N. *Redes de Petri temporais : método de análise baseado em tempo global*. Dissertação (Mestrado) — Universidade Federal do Paraná, Setor de Ciências Exatas, Programa de Pós-Graduação em Informática, 2008.

MATTAR JUNIOR, N. *et al.* Análise da duração de seqüências de disparos de transições em redes de Petri temporais. In: INTELIGENTE, V. S. B. de A. (Ed.). *Anais do SBAI 2007 - VIII Simpósio Brasileiro de Automação Inteligente*. Florianópolis: [s.n.], 2007. SBAI 2007.

MCMILLAN, K. L. *Symbolic Model Checking*. Norwell, MA, USA: Kluwer Academic Publishers, 1993. ISBN 0792393805.

MCMILLAN, K. L. A technique of state space search based on unfolding. *Formal Methods in System Design: An International Journal*, v. 6, n. 1, p. 45?65, January 1995.

MELZER, S.; ROMER, S. Deadlock checking using net unfoldings. In: *In Proceeding of 9th International Conference on Computer Aided Verification (CAV?97*. [S.l.]: Springer-Verlag, 1997. p. 352–363.

MERLIN, P. *A Study of Recoverability of Computer Systems*. Tese (Doutorado) — University of California IRVINE, 1974.

MERLIN, P.; FABER, D. Recoverability of communication protocol. COM-24, n. 9, September 1976.

MOHAMMADI, A.; AKL, S. G. *Technical Report No. 2005-499 Scheduling Algorithms for Real-Time Systems*. 2005.

MURATA, T. Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*, v. 77, n. 4, p. 541–580, 1989.

NAEDELE, M. An approach to modeling and evaluation of functional and timing specifications of real-time systems. *J. Syst. Softw.*, Elsevier Science Inc., New York, NY, USA, v. 57, n. 2, p. 155–174, 2001. ISSN 0164-1212.

NAEDELE, M.; JANNECK, J. W. Design patterns in petri net system modeling. In: *4th IEEE International Conference on Engineering Complex Computer Systems (ICECCS'98)*. Monterey, California: [s.n.], 1998.

NOERGAARD, T. *Embedded Systems Architecture: A Comprehensive Guide for Engineers and Programmers*. [S.l.]: Newnes, Elsevier Inc., 2005. ISBN 0750677929.

OSEK/VDX. *Specification*. 2001. *Online*; acesso em 24/06/2010. OSEK is a registered trademark of Continental Automotive GmbH (until 2007: Siemens AG). Disponível em: <http://www.osek-vdx.org/>.

PETERSON, J. L. Petri nets. *ACM Comput. Surv.*, ACM, New York, NY, USA, v. 9, n. 3, p. 223–252, 1977. ISSN 0360-0300.

PRESSMAN, R. S. *Engenharia de Software*. 6a. edição. ed. [S.l.]: McGraw-Hill, 2006.

RAMCHANDANI, C. *Analysis of asynchronous concurrent systems by timed Petri nets*. [S.l.], fev. 1974.

RIBEIRO, O.; FERNANDES, J.; PINTO, L. Model checking embedded systems with PROMELA. *IEEE International Conference on the Engineering of Computer-Based Systems*, IEEE Computer Society, Los Alamitos, CA, USA, v. 0, p. 378–385, 2005.

RUMBAUGH, J. *et al. Object-Oriented Modeling and Design*. Englewood Cliffs, NJ: Prentice–Hall, 1991.

SALDHANA, J. A.; SHATZ, S. M. UML diagrams to object Petri net models: An approach for modeling and analysis. In: *In International Conference on Software Engineering and Knowledge Engineering*. [S.l.: s.n.], 2000. p. 103–110.

SCHIEBE, M.; PFERRER, S. (Ed.). *Real-time systems engineering and applications*. Norwell, MA, USA: Kluwer Academic Publishers, 1992. ISBN 0-7923-9196-9.

SCHMIDT, Á.; VARRÓ, D. Checkvml: A tool for model checking visual modeling languages. In: STEVENS, P.; WHITTLE, J.; BOOCH, G. (Ed.). *UML*. [S.l.]: Springer, 2003. (Lecture Notes in Computer Science, v. 2863), p. 92–95. ISBN 3-540-20243-9.

SELIC, B. Model-driven development of real-time software using omg standards. *Object-Oriented Real-Time Distributed Computing, 2003. Sixth IEEE International Symposium on*, p. 4–6, May 2003.

SGROI, M. *et al.* Synthesis of embedded software using free-choice petri nets. In: *DAC '99: Proceedings of the 36th annual ACM/IEEE Design Automation Conference*. New York, NY, USA: ACM, 1999. p. 805–810. ISBN 1-58133-109-7.

SGROI, M. *et al.* Formal models for embedded system design. *IEEE Design and Test of Computers*, IEEE Computer Society, Los Alamitos, CA, USA, v. 17, n. 2, p. 14–27, 2000. ISSN 0740-7475.

SHA, L.; RAJKUMAR, R.; LEHOCZKY, J. Priority inheritance protocols: An approach to real-time synchronization. *IEEE Transactions on Computers*, IEEE Computer Society, Los Alamitos, CA, USA, v. 39, n. 9, p. 1175–1185, 1990. ISSN 0018-9340.

SOARES, M. d. S.; VRANCKEN, J. A metamodeling approach to transform uml 2.0 sequence diagrams to petri nets. In: *SE '08: Proceedings of the IASTED International Conference on Software Engineering*. Anaheim, CA, USA: ACTA Press, 2008. p. 159–164. ISBN 978-0-88986-716-1.

SOMMERVILLE, I. *Software engineering (5th ed.)*. Redwood City, CA, USA: Addison Wesley Longman Publishing Co., Inc., 1995. ISBN 0-201-42765-6.

STADZISZ, P.; RENAUX, D. Software embarcado. In: ____. *XIV Escola Regional de Informática SBC Paraná. Org. SBC*. 1. ed. Guarapuava: UNICENTRO (Universidade Estadual do Centro-Oeste), 2007. v. 1, p. 107–155.

STAINES, T. S. Intuitive mapping of UML 2 activity diagrams into fundamental modeling concept Petri net diagrams and colored Petri nets. In: *ECBS*. [S.l.]: IEEE Computer Society, 2008. p. 191–200.

STANKOVIC, J. A. Misconceptions about real-time computing: A serious problem for next-generation systems. *Computer*, IEEE Computer Society Press, Los Alamitos, CA, USA, v. 21, n. 10, p. 10–19, 1988. ISSN 0018-9162.

STANKOVIC, J. A. Strategic directions in real-time and embedded systems. *ACM Comput. Surv.*, ACM, New York, NY, USA, v. 28, n. 4, p. 751–763, 1996. ISSN 0360-0300.

STEVENS, P.; WHITTLE, J.; BOOCH, G. (Ed.). *UML 2003 - The Unified Modeling Language, Modeling Languages and Applications, 6th International Conference, San Francisco, CA, USA, October 20-24, 2003, Proceedings*, v. 2863 de *Lecture Notes in Computer Science*, (Lecture Notes in Computer Science, v. 2863). [S.l.]: Springer, 2003. ISBN 3-540-20243-9.

SU, F.-S.; HSIUNG, P.-A. Extended quasi-static scheduling for formal synthesis and code generation of embedded software. In: *CODES '02: Proceedings of the tenth international symposium on Hardware/software codesign*. New York, NY, USA: ACM, 2002. p. 211–216. ISBN 1-58113-542-4.

TELELOGIC. *Telelogic® Harmony for Embedded Software*. 2009. [*Online*; acesso em 20/02/2009]. Disponível em: <`http://www.telelogic.com/services/process_improvement/index.cfm`>.

TRICKOVIE, I. Formalizing activity diagram of UML by PETRI nets. *Novi Sad Journal of Mathematics*, v. 30, n. 3, p. 161–171, 2000. ISSN 1450-5444. Published by Institute of Mathematics, Faculty of Science, University of Novi Sad.

WANG, J.; DENG, Y.; XU, G. Reachability analysis of real-time systems using time petri nets. *IEEE Trans. on Systems, Man and Cybernetics-Part B : Cybernetics*, 2000.

WIKIPEDIA. *Mastermind (board game) — Wikipedia, The Free Encyclopedia*. 2008. [*Online*; acesso em 19/12/2008]. Disponível em: <`http://en.wikipedia.org/w/index.php?title=Mastermind_(board_game)&oldid=256830761`>.

WORDSWORTH, J. B. *Software Development with Z: A Practical Approach to Formal Methods in Software Engineering*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1992. ISBN 0201627574.

XU, D.; HE, X.; DENG, Y. Compositional schedulability analysis of real-time systems using time petri nets. *IEEE Transactions on Software Engineering*, IEEE Computer Society, Los Alamitos, CA, USA, v. 28, p. 984–996, 2002. ISSN 0098-5589.

XU, D. *et al.* Compositional schedulability analysis of real-time systems using time petri nets. *IEEE Trans. Soft. Engineering*, v. 28, p. 984–996, 2002.

YI, W.; PETTERSSON, P.; DANIELS, M. Automatic Verification of Real-Time Communicating Systems By Constraint-Solving. In: HOGREFE, D.; LEUE, S. (Ed.). *Proc. of the 7*th *Int. Conf. on Formal Description Techniques*. [S.l.]: North–Holland, 1994. p. 223–238.

YONEDA, T.; SCHLINGLOFF, B.-H. Efficient verification of parallel real–time systems. *Form. Methods Syst. Des.*, Kluwer Academic Publishers, Hingham, MA, USA, v. 11, n. 2, p. 187–215, 1997. ISSN 0925-9856.