Pós-Graduação em Ciência da Computação

"Modelagem e Análise de Especificações de Sistemas Embarcados de Tempo-Real Críticos com Restrições de Energia"

Por

Ermeson Carneiro de Andrade

Dissertação de Mestrado

Universidade Federal de Pernambuco
posgraduacao@cin.ufpe.br
www.cin.ufpe.br/~posgraduacao

RECIFE, Março/2009

Universidade Federal de Pernambuco
CENTRO DE INFORMÁTICA
PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Ermeson Carneiro de Andrade

"Modelagem e Análise de Especificações de Sistemas Embarcados de Tempo-Real Críticos com Restrições de Energia"

Este trabalho foi apresentado à Pós-Graduação em Ciência da Computação do Centro de Informática da Universidade Federal de Pernambuco como requisito parcial para obtenção do grau de Doutor em Ciência da Computação.

ORIENTADOR: ***Prof. Dr. Paulo Romero Martins Maciel***

RECIFE, MARÇO/2009

À minha família e amigos.

AGRADECIMENTOS

- A Deus por ter conseguido vencer mais esta etapa na minha vida.
- Ao meu orientador Prof. Paulo Maciel, pela amizade, paciência e por ter acreditado acima de tudo em mim e no meu trabalho. Agradeço também ao Prof. Ricardo Salgueiro e à Prof. Cremilda Matos por todo apoio e ajuda.
- A todos os meu amigos do CIN. Dentre eles: Bruno Nogueira, Carlos Romero, Erica Sousa, Eduardo Tavares, Gabriel Alves, Gustavo Callou, Ivaldir Júnior, Julian Menezes, Kalil Bispo, Rafael Antonello e Patricia Takako. Meu agradecimento em especial a Julian e a Gustavo, pelos momentos de descontração e por todo apoio e ajuda.
- Aos meus pais Eraldo e Nágila, bem como aos meus irmãos Eraldo Júnior, Eduardo, Ênio e Edimo, pois mesmo estando distantes, como sempre, me deram todo apoio e suporte necessários.
- Ao SOFTEX Recife, em especial a todos que fazem parte do NEXT.

Nada vem de graça.

—DESCONHECIDO

RESUMO

Análise de requisitos é uma tarefa crítica em qualquer projeto de Sistema Embarcado de Tempo-Real (ERTS). Normalmente, esses sistemas possuem restrições severas de tempo que precisam ser satisfeitas para o correto funcionamento, visto que violações podem ser catastróficas, tais como perdas de vidas ou grande quantias de recursos financeiros. Além disso, existem sistemas onde energia é outra restrição que também precisa ser satisfeita. Assim, a descoberta precoce dos potenciais problemas podem reduzir os riscos da propagação dos erros da especificação para o código final. É importante ressaltar que o custo da detecção de um erro após a entrega do sistema é, no mínimo, 10-100 vezes maior do que ele tivesse sido detectado em tempo de especificação do sistema.

Atualmente, a forma mais amplamente usada para modelar requisitos de sistemas críticos é através das linguagens semiformais, tais como SysML (*System Modelling Language*) ou UML (*Unified Modeling Language*), devido principalmente a sua notação amigável e intuitiva. No entanto, os modelos semiformais gerados por essas linguagens, por si só não fornecem suporte para avaliação de desempenho das especificações dos sistemas, assim, faz-se necessário o mapeamento destes modelos semiformais para modelos formais. Pois, modelos formais são apoiados por fundamentos matemáticos sólidos, que suportam sua semântica precisa, estimulam a avaliação de desempenho e fornecem suporte para verificações das propriedades qualitativas e análises. Esses modelos formais, no entanto, não são intuitivos e requerem um considerável esforço por parte dos projetistas para entenderem a notação usada. Assim, é sensato adotar o uso colaborativo dos modelos semiformais e formais.

Portanto, a fim de obter a integração dos modelos formais e semiformais, este trabalho propõe o mapeamento dos diagramas comportamentais da SysML em uma Rede de Petri Temporizada. As restrições de tempo e anotações energia são representadas pelo novo *profile* da UML MARTE (*Modeling and Analysis of Real-time and Embedded systems*). Além disso, uma metodologia de avaliação de desempenho das especificações de sistemas críticos é proposta, com o intuito de auxiliar o processo de modelagem e avaliação. Por fim, estudos de caso são apresentados mostrando a aplicabilidade deste trabalho.

Palavras-chave: Avaliação de Desempenho, Análise de Requisito, Consumo de Energia, Modelagem, Rede de Petri Temporizada, Sistemas Embarcados, SysML.

ABSTRACT

Requirements analysis is a critical task in any Embedded Real-Time System (ERTS) project. Normally, these systems have stringent timing constraints that must be satisfied for the correct functioning, since violation might be catastrophic, such as loss of human lives or large amount of financial resources. In addition, there are systems where energy is another constraint that must also be satisfied. Hence, early detection of potential problems may reduce risks of fault propagations from early specification to the final code. It is important to highlight that the cost of detection of an error after the product's releasing would cost 10-100 times more to fix than if it were found in the design phase of the system.

Nowadays, the most widely used way to specify critical system requirements is through semiformal language, such as SysML (*System Modelling Language*) or UML (*Unified Modeling Language*), chiefly due to their friendly and intuitive notations. However, the semiformal models generated by these languages, in themselves do not provide support for performance evaluation of the system specifications, thus, it is necessary the mapping of these semiformal models into formal models. Because formal models are supported by sound mathematic foundations that uphold their precise semantics, foster performance evaluation and provide support to qualitative property verification and analysis. These models, however, quite often are not intuitive and may require considerable designer's effort to figure out the respective representation. Hence, it is wise adopting a collaborative use of both semi-formal and formal models.

Therefore, in order to obtain the integration of formal and semiformal models, this work proposes the mapping of SysML behavioral diagrams into a Time Petri Net. The time constraints and energy annotations are represented by the new profile of UML MARTE (*Modeling and Analysis of real-time and Embedded systems*). Furthermore, a performance evaluation methodology of the critical system specifications is proposed, with the goal of helping the process of modeling and evaluation. Lastly, case studies are presented showing the applicability of this work.

Keywords: Performance Evaluation, Energy Consumption, Modeling, Time Petri Net, Embedded Systems, Requirements Analysis, SysML.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS

ACET - *Average Case Execution Time.*

BCET - *Best Case Execution Time.*

CPN - *Coloured Petri Net.*

DA - Diagrama de Atividades.

DC - Diagrama de Colaboração.

DCS - Diagramas Comportamentais da SysML.

DE - Diagrama de Estados.

DS - Diagrama de Seqüência.

EFT - *Limite Inferior.*

ERTS - Sistemas Embarcados de Tempo-Real.

ETPN - Redes de Petri Temporizadas com restrições de Energia.

INA - *Integrated Net Analyser.*

LSC - *Live Sequence Chart.*

LFT - *Limite Superior.*

LGSPN - *Labeled Generalized Stochastic Petri Net.*

MARTE - *Profile for Modeling and Analysis of Real-time and Embedded systems.*

MEMBROS - *Methodology for embedded Critical Software Construction.*

MSC - *Message Sequence Chart.*

OMG - *Object Management Group.*

RdP - Redes de Petri.

SPN - *Stochastic Petri Nets.*

SPT - *Profile for Schedulability, Performance and Time.*

SysML - *System Modelling Language.*

TimeNET - *Timed Net Evalution.*

TPN - Redes de Petri Temporizadas.

UML - *Unified Model Language.*

WCET - *Worst Case Execution Time.*

CAPÍTULO 1

INTRODUÇÃO

Este capítulo provê uma breve introdução aos sistemas embarcados, destacando-se as principais restrições a serem avaliadas neste contexto, como também, a necessidade da integração dos modelos semiformais e formais. Em seguida, são apresentados a motivação, os trabalhos relacionados e a proposta deste trabalho bem como o seu escopo.

1.1 CONTEXTO

Atualmente, os sistemas embarcados estão presentes em praticamente todas as áreas do nosso cotidiano; nós os utilizamos diariamente de forma que muitas vezes sequer percebemos que eles estão lá. Caixas eletrônicos, telefones celulares, relógios, geladeiras, microondas, osciloscópios, roteador, filmadoras são alguns exemplos de tais dispositivos. De fato, com o aumento significativo dos sistemas dedicados de controle, a maioria dos dispositivos que usamos no dia-a-dia possuem um processador digital, responsável por realizar uma tarefa específica.

Sistemas Embarcados de Tempo-Real (ERTS) são aqueles que dependem não só da integridade dos resultados, mas também do tempo em que tais resultados são produzidos. Esses sistemas podem ser classificados em duas categorias: críticos (*hard*) e não críticos (*soft*). Em sistemas embarcados de tempo-real não críticos, caso as restrições temporais não sejam satisfeitas, poderá ocorrer uma degradação no desempenho do sistema que poderá ser tolerada (ex.: servidores web, telefone celular, voz sobre IP, TV digital, vídeo conferência, etc). No entanto, os sistemas embarcados de tempo-real críticos são aqueles cuja restrição de tempo deve ser respeitada a todo custo, visto que a violação pode ser catastrófica [TMSO08]. Exemplos de sistemas críticos podem ser encontrados em controle automobilístico, equipamentos médicos, aplicações militares, controle aéreo e espacial, centrais nucleares, entre outros. Assim, garantir as restrições temporais é uma questão fundamental nos ERTS [BL04, TMSO08, TMS$^+$07].

Além disso, avanços tecnológicos na área da microeletrônica têm possibilitado o desenvolvimento dos ERTS com funcionalidades cada vez mais complexas e sofisticadas, permitindo assim a criação de dispositivos móveis eficientes, precisos e seguros, tais como equipamentos militares (ex.: satélites e mísseis teleguiados) e médicos (ex.: termômetro e oxímetro de pulso). Esses dispositivos geralmente possuem uma fonte de energia restrita (ex.: bateria), de tal forma que se essa fonte chegar ao fim, o sistema pára de funcionar [TMS$^+$07]. Logo, estudos relativos à conservação/economia de energia tornaram-se extremamente relevantes nos projetos desses sistemas. Portanto, estimativas referentes ao

consumo de energia podem fornecer informações importantes aos projetistas tanto relativas ao tempo de vida da bateria como também de partes da aplicação que precisam ser otimizada [AMCNb, AMCNc, TMS+07].

Adicionalmente, com o crescimento da heterogeneidade e da complexidade dos ERTS, é requerida uma abordagem interdisciplinar no processo de desenvolvimento de tais sistemas, envolvendo as áreas de engenharia de *software*, mecânica, elétrica e eletrônica. Nesse sentido, foi especificada pela OMG (*Object Management Group*) [OMG89] uma linguagem de modelagem, denominada SysML (*System Modelling Language*) [Sys07] a qual suporta a especificação, análise, desenho e verificação de uma grande variedade de sistemas complexos. Esses sistemas podem incluir *hardware*, *software*, informações, métodos, pessoas e instrumentos. SysML estende UML (*Unified Model Language*) 2.0 [UML05], sendo SysML usada para modelar sistemas que não são totalmente baseados em *software* [Sys07].

No entanto, durante a modelagem da especificação dos sistemas embarcados de tempo-real é indispensável à descrição dos aspectos quantitativos [TZ05]. Para isso, um novo *profile* da UML para Modelagem e Análise dos Sistemas Embarcados de Tempo-Real (MARTE) [MAR07] foi especificado pela OMG. MARTE tem o objetivo de substituir o UML SPT (*Profile for Schedulability, Performance and Time*) [SPT03], sendo usado para auxiliar na construção de modelos que possam ser usados para fazer precisões quantitativas relativas às características dos ERTS, levando em consideração tanto as características de *hardware* quanto de *software*.

Os modelos semiformais por si só não fornecem suporte para avaliação de desempenho das especificações dos sistemas, assim, faz-se necessário o mapeamento desses modelos para modelos formais. Exemplos significativos de modelos amplamente utilizados para análise de desempenho são, por exemplos, a álgebra min-máx [HoCU93], as cadeias de *Markov* [BGdMT98b], a teoria das filas [Wal88] e as redes de Petri temporizadas [MF76]. Contudo, esses modelos formais não são intuitivos e requerem um considerável esforço por parte dos projetistas para entenderem a notação usada. Por outro lado, as anotações semiformais, tais como SysML ou UML, têm sido largamente adotadas no ciclo de desenvolvimento dos ERTS, devido principalmente a sua notação amigável e intuitiva. Assim, é sensato adotar o uso colaborativo dos modelos semiformais e formais.

1.2 MOTIVAÇÃO

Análise de requisitos é uma tarefa crucial em qualquer projeto de sistema embarcado de tempo-real crítico. Normalmente, esses sistemas possuem restrições temporais que precisam ser obrigatoriamente obedecidas para que o sistema funcione corretamente. Além disso, existem sistemas onde energia é uma outra restrição que também precisa ser satisfeita. Assim, a descoberta precoce dos potências problemas pode reduzir os riscos da propagação dos erros da especificação para o código final, evitando dessa forma perdas de recursos financeiros ou vidas humanas [AMCNb].

A modelagem é parte fundamental no desenvolvimento de um sistema; ela consiste

no detalhamento de um fragmento de requisito segundo uma determinada visão, onde um projeto de requisito ou especificação é expressa em termos de um ou mais modelos. Os modelos são muito importantes por diversas razões; dentre elas é possível destacar: (i) ajudam a visualizar o sistema como um todo, (ii) permitem especificar a estrutura ou o comportamento de um sistema e (iii) proporcionam um guia para a construção de um sistema. Assim, os modelos ajudam os projetistas a entenderem a informação, a função e o comportamento de um sistema, tornando-se a tarefa de análise de requisitos mais fácil e sistemática. Portanto, o modelo torna-se a base para o projeto, fornecendo ao projetista uma representação essencial do sistema, a qual pode ser mapeada num contexto de implementação.

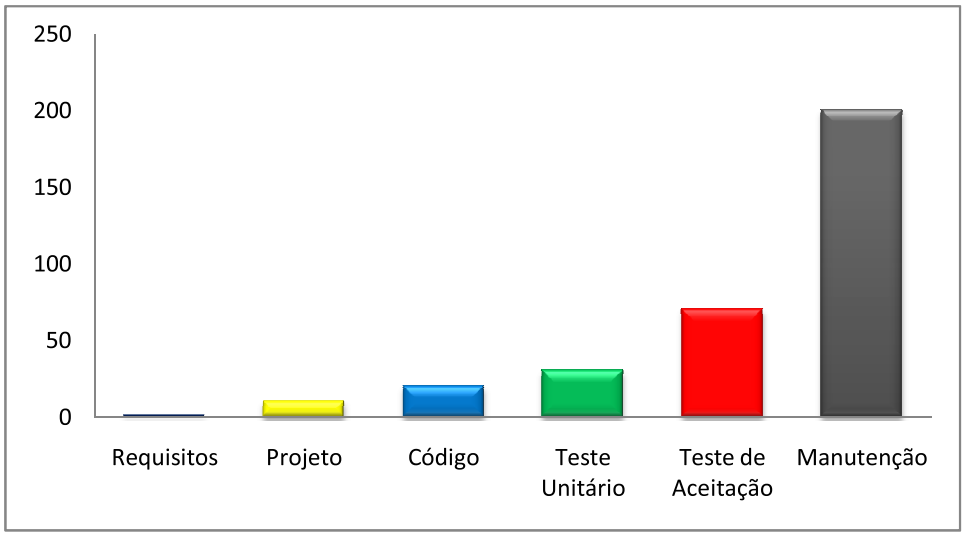
No entanto, através desses modelos a tarefa de realizar análises quantitativas ou verificações é praticamente inviável, assim o mapeamento dos modelos semiformais em modelos formais é uma tarefa fundamental na construção dos ERTS. Entre os benefícios dos modelos formais é possível destacarmos os seguintes:

- permitir a redução da dependência da intuição e do julgamento humano;
- possuir grande poder de abstração;
- permitir a validação do modelo através de simulações;
- possibilitar a detecção de inconsistências e ambiguidades na especificação decorrentes da linguagem natural;
- permitir a prototipação dos ERTS críticos;
- constituir um mecanismo rigoroso e efetivo para modelagem, síntese e análise de sistemas.

No âmbito dos modelos formais, as redes de Petri (RdP) costumam ser bastante utilizadas para a modelagem e a especificação de sistemas concorrentes, assíncronos, distribuídos, paralelos, não-determinísticos e estocásticos. As RdP foram propostas por Carl Adam Petri, em 1962, na sua tese de doutoramento [Pet62]. Desde então, elas vêm sendo aplicadas nas mais diversas áreas, indo desde a ciência da computação, passando pela administração de empresas e indo até áreas da biologia.

Desde seu surgimento, diversas extensões foram propostas para esse formalismo. Entre estas extensões, estão as Redes de Petri Temporizadas com restrições de Energia (ETPN) [TMSO08], que além da noção de tempo, inclui a noção energia. Este tipo de rede permite que se obtenham métricas a partir de um modelo. Pode-se, por exemplo, obter o menor ou maior tempo associado a um determinado caminho do modelo ETPN, como também o consumo de energia associado a esses caminhos. Esses tipos de métricas são muito importantes nos ERTS, pois é possível verificar se os sistemas não violam suas respectivas restrições de tempo e/ou energia. Ademais, respeitando algumas propriedades necessárias [Mur89], os resultados das métricas podem ser obtidos de forma analítica, ou ainda pela sua simulação do modelo.

Diferentemente do senso comum, modelos formais podem ser um meio significativo para a redução do custo global do desenvolvimento dos ERTS. A Figura 1.1 demonstra que quanto mais cedo o erro for detectado e corrigido, menor será o custo para a organização, ou seja, a correção de um erro durante a fase de requisitos acarreta um custo muito menor que a correção do mesmo erro durante a fase de manutenção. Normalmente os modelos formais propõem encontrar erros e incosistências ainda durante as fases iniciais do desenvolvimento dos ERTS, assim é muito importante a adoção de modelos formais no intuito de diminuir o custo final dos projetos. Contudo, segundo [Som06, CW96] os modelos formais ainda não são utilizados em larga escala na indústria. Isso se deve a alguns fatores: (i) suas notações matemáticas que não são muito intuitivas, ou seja, requerem um considerável esforço por parte do projetista para compreende-las e (ii) a falta de ferramentas/métodos eficientes e amigáveis.



Fonte: [Hal07, p. 2]

Figura 1.1: Custo da detecção de um erro de acordo com o estágio no qual ele é descoberto.

1.3 TRABALHOS RELACIONADOS

Avaliação de desempenho de sistemas através de modelos formais pode auxiliar fortemente a redução de problemas. Por exemplo: o não atendimento de prazos, a diminuição da propagação de erros para as outras fases do ciclo de desenvolvimento e o não cumprimento das restrições de desempenho. Portanto, a diminuição de tais problemas podem auxiliar de forma considerável a redução de esforço, tempo e custo associados aos projetos. Assim, com o intuito de diminuir os problemas acima mencionados, diversas pesquisas têm sido realizadas objetivando o mapeamento dos modelos semiformais em modelos formais.

A maioria dos trabalhos que visa o mapeamento de um modelo semiformal para um modelo formal objetiva a análise quantitativa, mais especificamente os aspectos temporais [KP00, KP99, TZH05, TZ05, TZ06, MCBD02a], porém nenhum deles foca na avaliação

temporal e no consumo de energia, que é o objetivo deste trabalho. Atualmente, entre as linguagens semiformais de modelagem usadas para especificar sistemas, UML é a mais amplamente utilizada pela comunidade de engenharia de software e a indústria. No entanto, a essência da UML é modelar sistemas baseados em *software*. Porém, quando se está modelando ETRS, nos quais é necessário se ter uma abordagem interdisciplinar envolvendo as mais diversas áreas (*software*, mecânica, elétrica e eletrônica), SysML é considerada mais adequada para modelar esses tipos de sistemas heterogêneos, pois foi desenvolvida para esse propósito.

Pooley and King [KP00, KP99] apresentam como o Diagrama de Colaboração (DC) em combinação com Diagrama de Estados (DE) da UML podem ser sistematicamente transformados em uma *Stochastic Petri Nets* (SPN). Nessa abordagem, os DEs são transformados em modelos SPNs, e estes são combinados através do DC formando um único modelo. Os estados do DE são mapeados em lugares no modelo SPN e as transições são mapeadas em transições no modelo SPN. Em resumo, uma abordagem intuitiva é apresentada.

Em [TZH05, TZ05, TZ06], Trowitzsch et al. aborda a derivação do DE da UML em uma SPN para avaliação de desempenho de sistemas de tempo-real. Essa abordagem consiste na derivação dos elementos básicos, tais como estados e transições, do DE em representações SPNs correspondentes. Após isso, os fragmentos das SPNs são compostos em um único modelo. O UML SPT [SPT03] é usado como linguagem de especificação para as restrições dos sistemas de tempo-real. Para as análises o TimeNET (*Timed Net Evalution*) [GKZH94] é usado. Messeguer et al. [MCBD02a] foca na derivação sistemática do DE da UML em fragmentos de uma LGSPN (*Labeled Generalized Stochastic Petri Net*). Após isso, todos os fragmentos são compostos em um único modelo que represente todo o comportamento do DE. Somente tempos exponencialmente distribuídos são usados nessa abordagem, isto é, tempos determinísticos não são considerados.

Por outro lado, existem algumas abordagens que almejam análises qualitativas [BP01a, BP01b, DdSS, LPK$^+$00]. Em [BP01a, BP01b], Baresi e Pezzé apresentam um conjunto de regras para traduzir uma especificação descrita em UML para uma rede de Petri predicado/transição. Nessa abordagem os autores propõem a transformação do diagrama de classe em conjunto com o DE em uma rede de Petri predicado/transição. Para cada método de uma classe, é criado um par de lugares de rede de Petri. Um dos lugares indica a requisição do método e o outro indica o retorno do método após sua execução. Além disso, também são identificadas as chamadas externas que cada classe realiza. Para isso, são acrescentados pares de lugares correspondentes às chamadas efetuadas pela classe. Após isso, os estados do DE são mapeados em lugares da Rede de Petri e as transições são mapeadas em transições de redes de Petri. Então, os modelos de estados que representam cada classe são integrados a fim de obter-se uma única rede de Petri. Por fim, simulações, análises de alcançabilidade e verificações do modelo são realizadas.

Similarmente em [DdSS], o autor apresenta uma abordagem parecida ao trabalho de Baresi, no qual é proposta a transformação do diagrama de classe em conjunto com o DE em uma rede de Petri predicado/transição. No entanto, se por um lado Baresi e Pezzé

empregam regras de mapeamento para traduzir os DE desenhados pelo projetista em redes de Petri, em [DdSS] os autores sugerem que o comportamento de cada classe seja diretamente descrito pelo projetista por meio de uma rede de Petri. Já em [LPK+00], Lee et al. propõem a derivação de cenários em redes de Petri temporizadas. Nessa abordagem o diagrama de caso de uso é usado para elicitar os requisitos e criar os cenários. Os cenários são representados pelos diagramas de seqüência. Uma vez criados os cenários, então eles são transformados em redes de Petri temporizadas, onde a linha de vida de uma entidade (*lifeline*) é representada por dois lugares e os eventos de chegada e saída na linha de vida são representados por transições. As análises são realizadas de forma a encontrar informações erradas ou esquecidas nos modelos gerados.

Além disso, existem outras abordagens que almejam a transformação de modelos semiformais para modelos formais, no entanto a UML não é utilizada como linguagem de especificação [AMN+05, AMN+06, VCF+06]. Em [AMN+05, AMN+06], Amorim et al. propõem um conjunto de passos para o mapeamento do *Live Sequence Chart* (LSC) em uma representação *Coloured Petri Net* (CPN) [Jen92] equivalente. A LSC permite especificar anti-cenários, bem como modelar o que deve ocorrer. O objetivo desse trabalho é realizar análises e verificações das propriedades dos sistemas embarcados. Essas análises e verificações são realizadas através do CPN *Tools* [VLM+03]. Por fim, Vijaykumar et al. [VCF+06] apresenta o processo de transformação dos *Statechart* [H+87] em uma cadeia de *Markov*.

1.4 OBJETIVOS

Este trabalho possui o objetivo de apresentar o mapeamento dos Diagramas Comportamentais da SysML (DCS) em modelos ETPN, para a partir desses modelos gerados, realizar análises quantitativas (ex.: tempo de execução e consumo de energia) e verificações. Em outras palavras, a partir dos modelos de alto-nível será possível realizar análises/verificações, antes mesmo do *software/hardware* ser concedido. É importante ressaltar que as análises qualitativas capturam os aspectos lógicos da evolução dos sistemas. Entre as propriedades de interesse, podemos ressaltar a existência ou ausência de *deadlock*, *liveness* e *reversibilidade*, por exemplo. As análises quantitativas, por outro lado, têm por principal objetivo a análise de desempenho.

Esses modelos ETPN gerados pelo processo de mapeamento dos diagramas comportamentais da SysML, deverão ser capazes de fornecer métricas importantes para o processo de tomada de decisão durante o desenvolvimento dos ERTS. Exemplos dessas métricas são: o tempo de execução no pior ou melhor caso e o consumo de energia associado aos *traces*[1] do tempo de execução. Os resultados dessas métricas poderão ser utilizados, por exemplo, na promoção da eficiência, *corretude* e validação dos requisitos.

Este trabalho também tem o objetivo de propôr uma metodologia de modelagem e análise dos ERTS. Essa metodologia apresentará uma série de modelos ETPN básicos gerados pelo processo de mapeamento, que através da composição sistemática, obtém-se

[1]Conjunto ordenado que designa a execução das ações ou eventos.

um modelo ETPN que representa a especificação do sistema, e assim, será possível aplicar técnicas de análises e verificações ainda durante as fases iniciais do ciclo de desenvolvimento dos sistemas. Além disso, a metodologia será composta por diversos passos que vão desde o projeto de requisitos, passando pelo mapeamento dos diagramas comportamentais da SysML, até a análise dos resultados obtidos.

Adicionalmente, os DCS são compostos pelos seguintes diagramas: estados, atividades, seqüência e caso de uso. Esses diagramas são usados para visualizar, especificar, construir e documentar aspectos dinâmicos do sistema. Este trabalho tem o objetivo de adotar os diagramas de estados, atividades e seqüência, pois estes apresentam características adequadas para modelar os aspectos dinâmico dos ERTS críticos. Os diagramas de caso de uso não serão tratados, pois seu objetivo é auxiliar a comunicação entre os analistas e o cliente, ou seja, possui a função de explicar o que o sistema deve fazer, no entanto, não pode especificar como isto será conseguido, diferentemente dos demais diagramas.

Mais especificamente este trabalho possui os seguintes objetivos:

- desenvolver um modelo ETPN para realizar análises (qualitativas e quantitativas) dos sistemas embarcados de tempo-real com restrições de energia;
- desenvolver um método para o mapeamento dos diagramas comportamentais da SysML em modelos ETPN. Neste trabalho os diagramas de estados, atividades e seqüencia são adotados;
- definir uma metodologia que auxilie no processo de avaliação de desempenho das especificações dos ERTS;
- definir métricas para a estimação do tempo de execução e consumo de energia dos ERTS.

1.5 ESTRUTURA DA DISSERTAÇÃO

Capítulo 2 introduz os conceitos fundamentais a serem utilizados na dissertação, tais como: sistemas embarcados, SysML, MARTE e redes de Petri. Capítulo 3 apresenta uma metodologia de avaliação de desempenho das especificações de sistemas críticos. Capítulo 4 apresenta o processo de mapeamento dos diagramas comportamentais da SysML em uma rede de Petri. Capítulo 5 apresentam estudos de caso no qual foi aplicada a metodologia. Finalmente, Capítulo 6 conclui o trabalho e apresenta os trabalhos futuros.

CAPÍTULO 2

FUNDAMENTOS

Este capítulo apresenta os principais conceitos da dissertação. Primeiramente, os sistemas embarcados são introduzidos, destacando suas principais características. Em seguida, os diagramas comportamentais que são usados nesta dissertação para a modelagem dos sistemas críticos são apresentados. Após isso, o novo *profile* da UML MARTE para modelagem e análises de sistemas embarcados de tempo-real, também é introduzido. Por fim, são abordados os principais conceitos que envolvem as Redes de Petri (RdP). Conceitos gerais tais como propriedades comportamentais e estruturais, métodos de análise, exemplos de modelagens e refinamento das RdP são abordados. Além disso, também é apresentada uma extensão das RdP, as Redes de Petri Temporizadas com restrição de Energia (ETPN), a qual é adotada nesta dissertação.

2.1 SISTEMAS EMBARCADOS

Sistemas embarcados são aplicações dedicadas, geralmente responsáveis por realizar uma tarefa específica de maneira contínua e ao custo mínimo. Normalmente, esses dispositivos estão inseridos em máquinas ou em sistemas maiores.

Atualmente, os sistemas embarcados estão presentes no nosso cotidiano sob diferentes formas e com diferentes objetivos. Sistemas aviônicos, sistemas de telemetria, sistemas de anti-travamento em freios, controladores de vôo, telefones celular, calculadoras de mão, microondas, caixas eletrônicos, video-games, periféricos de computadores (impressoras, mouses, teclados, etc) são alguns exemplos de tais dispositivos. Dentre as principais características desses sistemas é possível destacar as seguintes:

- **Funcionalidade Específica**. Os sistemas embarcados são diferentes dos computadores de propósito geral (computador pessoal), pois possuem uma capacidade de processamento restrita e realizam um conjunto de tarefas específicas, geralmente com requisitos específicos.
- **Limitações**. A utilização de sistemas embarcados implica na redução de alguns recursos, tais como: tamanho, peso, desempenho, potência dissipada, entre outros. Além disso, esses sistemas devem ser baratos e ocupar o menor espaço físico possível.
- **Tempo-Real**. Boa parte dos sistemas embarcados devem responder aos eventos produzidos por outros dispositivos, bem como ao ambiente com o qual ele interage. Esses eventos devem cumprir prazos, os *deadlines*. O não cumprimento desses

prazos, pode levar o sistema a um estado inconsistente de funcionamento, podendo resultar em conseqüências severas.

Nos últimos anos, devido a grande pressão mercadológica, somada à contínua evolução tecnológica, tem imposto às empresas a necessidade de projetarem novos sistemas embarcados dentro de curto espaço de tempo, ou seja, em poucos meses. Além disso, novos produtos têm uma vida cada vez mais curta, de modo que o retorno financeiro de seu projeto deve ser obtido também em poucos meses. Assim, a fim de suprir essa demanda mercadológica, várias metodologias têm surgido ao longo dos anos. No entanto, este trabalho concentra-se principalmente na metodologia *hardware-software co-design*, que é descrita a seguir.

2.1.1 Hardware/software co-design

Hardware/software co-design é uma nova metodologia de projeto, cujo principal objetivo consiste em projetar sistemas embarcados que satisfaçam às restrições de projeto através da utilização de componentes de prateleira e componentes de aplicação específica, a fim de reduzir o *time-to-market* [Wol94]. Esta heterogeneidade dos componentes implica que em tais sistemas *hardware* (visando maior desempenho e menor potência) e *software* (visando maior flexibilidade e baixo custo) devem ser desenvolvidos de forma integrada e eficiente. A entrada para este processo é a especificação funcional do sistema. Por outro lado, a saída do processo é um mapeamento entre cada função da especificação e um componente da macro-arquitetura. O uso de técnicas de *hardware-software co-design* permite o projeto do sistema como um todo, e não como partes separadas de *hardware* e *software*.

A metodologia de *hardware/software co-design* pode ser dividida principalmente em quatro fases não-independentes (ver Figura 2.1). Essas fases são descritas a seguir:

- **Especificação.** Esta fase está relacionada com a descrição formal dos requisitos do sistema, em alto nível de abstração. Nesta fase, são identificados os requisitos funcionais e não-funcionais (ex.: desempenho, consumo de energia, etc). A descrição dos requisitos pode ser feita através de linguagens específicas ou modelos formais, tais como redes de Petri, máquinas de estado finito, *statecharts*, álgebra de processos, dentre outros.

- **Particionamento.** Nesta fase é feita a seleção sobre quais componentes serão implementados em *hardware* ou em *software*. Essta divisão pode ser feita de três formas: (i) manualmente, (ii) automaticamente através do uso de ferramentas ou (iii) interativamente através da combinação das outras duas formas. Normalmente, os métodos de particionamento levam em consideração estimadores de desempenho para o particionameno da especificação. É importante destacar que o mal particionamento pode afetar o esforço, tempo e custo associados aos projetos.

- **Hardware/software co-síntese.** Esta fase tem o objetivo de mapear o protótipo virtual em um protótipo real de forma a satisfazer às restrições de projeto e às restrições temporais dos componentes da arquitetura alvo. O protótipo virtual representa a saída do particionamento, ou seja, um conjunto de módulos comunicantes que são implementados ou em *hardware* ou em *software*. As principais tarefas a serem realizadas durante a etapa de *co-síntese* incluem a síntese do *hardware*, a síntese do *software* e a síntese da interface de comunicação entre os diversos componentes. Além disso, nessa fase, algumas decisões podem ser tomadas. Por exemplo: a escolha do processador, a rede de interconexão, os protocolos de comunicação, a interface entre *hardware* e *software*, o escalonamento de processos concorrentes, entre outros.

- **Validação.** Quando se fala em validar um sistema deve se considerar, também, a prototipagem deste. Atualmente, técnicas de validação desse tipo de projetos seguem em geral dois caminhos: a verificação formal e a simulação. Na verificação formal a corretude funcional de um sistema é realizada através de provas matemáticas. Por outro lado, a simulação pode ser usada para verificar a funcionalidade e sua execução requer muitos recursos computacionais. É importante ressaltar que, em cada uma das etapas da metodologia as ferramentas de verificação ou de validação, podem ser usada para a constatação do bom funcionamento do sistema antes de passar à próxima etapa.

2.2 SISTEMAS EMBARCADOS DE TEMPO-REAL

Existe uma grande variedade de sistemas embarcados, dentre eles, destaca-se um grupo de sistemas que são limitados pelo tempo, os chamados sistemas embarcados de tempo-real (ERTS). Esse tipo de sistema não só depende do resultado lógico de processamento, mas também dos valores de tempo em que são produzidos. Em outras palavras, aplicações de tempo-real são aquelas nas quais um estímulo externo é percebido e respondido dentro de um tempo predeterminado. Assim, podemos diferenciar computação rápida de computação de tempo-real. Computação rápida disponibiliza seus resultados o mais rápido possível, por outro lado, computação de tempo-real obtém o resultado da computação respeitando os componentes temporais do sistema. Em geral, esses sistemas são agrupados de acordo com suas características em comum, as quais são descritas a seguir [Sin96]:

- **Tempo de Resposta Adequado.** Os sistemas de tempo-real devem responder a estímulos externos dentro de um intervalo de tempo bem definido, visto que a violação pode levar a conseqüências severas.

- **Previsibilidade.** Um outro requisito dos sistemas de tempo-real é que seu desempenho seja o mais previsível possível, isto é, o comportamento funcional e temporal do sistema deve ser tão determinismo quanto impõe sua especificação.

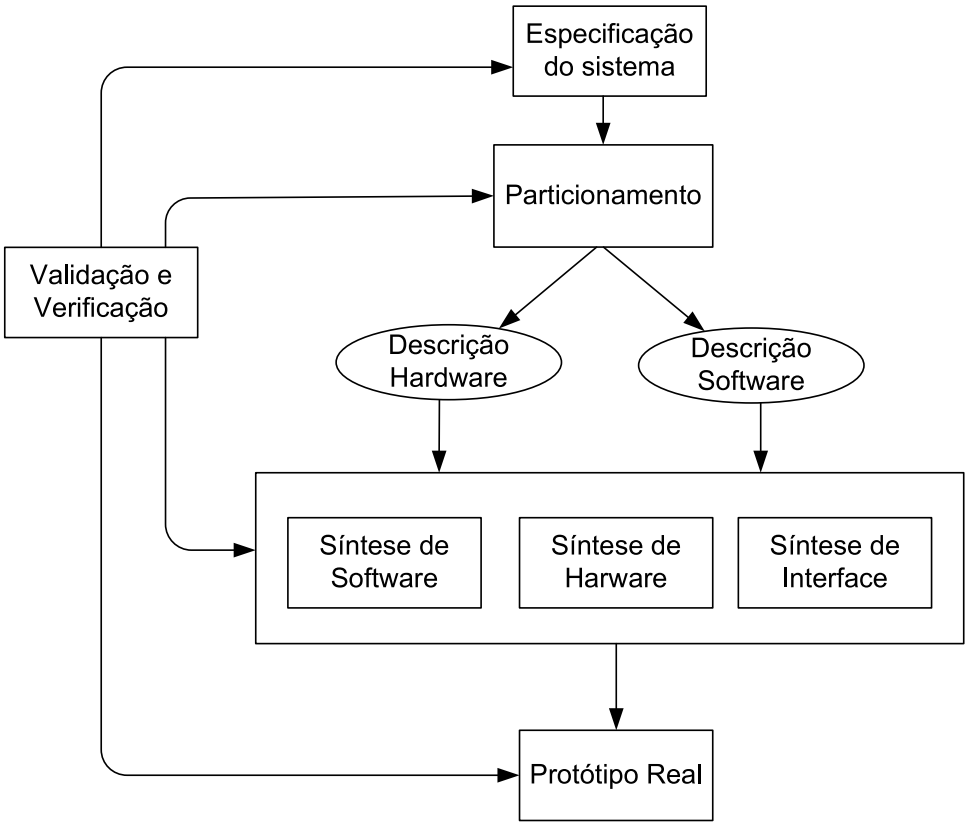


Figura 2.1: *Hardware/software co-design.*

- **Robustez.** O sistema deve ser capaz de continuar seu funcionamento normal, mesmo que pequenas variações no ambiente ocorram, sem degradar o seu serviço. Por exemplo: a degradação de um sinal, a falha num *hardware*, uma mudança no ambiente, etc.

- **Precisão.** O sistema deve fornecer os resultados o mais preciso possível. Às vezes é complexo computar resultados ótimos devido às restrições de tempo, assim o compromisso entre satisfação de requisitos temporais e precisão dos resultados é muito importante.

- **Concorrência.** Sistemas de tempo-real podem ser distribuídos e executar as tarefas de forma paralela. Pois esses sistemas podem ter sensores, independentes ou não, que enviam ao sistema estímulos, requisitando respostas para atuar no ambiente dentro de uma dada janela de tempo.

2.2.1 Tipos de Sistemas de Tempo-Real

Normalmente, os sistemas de tempo-real podem ser classificados em relação ao cumprimento de seus prazos de resposta ao ambiente, isto é, os *deadlines*. *Deadline* é o instante máximo desejado para a conclusão de uma tarefa.

Os sistemas de tempo-real podem ser classificados em duas categorias:

- **Sistemas de Tempo-Real Críticos.** São aqueles sistemas em que é imprescindível que a resposta ocorra em um determinado *deadline*, pois o não cumprimento pode levar, por exemplo, a perdas de vidas humanas ou grande quantias de recursos financeiros. Ex.: serviços médicos, controle de trafego aéreo, controle de processos indústrias, centrais nucleares, entre outros.

- **Sistemas de Tempo Real não Críticos.** São aqueles sistemas em que a resposta é importante, mas o sistema continuará funcionando de forma correta, mesmo que haja uma degradação da qualidade do serviço. Ex.: simuladores, games, aplicativos multimídia, entre outros.

2.3 SYSML

A SysML foi desenvolvida pela OMG em conjunto com o INCOSE (*International Council on Systems Engineering*) como a linguagem que permite a descrição correta e consistente dos sistemas entre diversos participantes do mesmo projeto (ex.: engenheiros de *software*, mecânicos, eletricistas e outros). Ela suporta a especificação, análise, desenho e verificação de sistemas o que inclui *hardware*, *software*, dados, pessoal, processos e infraestrutura [Sys07].

Segundo [Sil06], SysML é uma linguagem de modelagem de propósito geral para aplicações de engenharia de sistemas e fornece um padrão de descrição para uma grande variedade de sistemas complexos. Além de fornecer um padrão de modelagem, a SysML busca o aperfeiçoamento da qualidade do sistema, assegurando a troca de informações e diminuindo a distância semântica entre a engenharia de sistemas, *software*, e outras áreas da engenharia.

Adicionalmente, a linguagem de modelagem SysML é suportada por duas normas de interoperabilidade, a *OMG XMI 2.1 model interchange standard for UML 2 modeling tools* (Padrão de troca de modelo para ferramentas de modelagem UML 2) e a *ISO 10303-233 data interchange standard for systems engineering tools* (Padrão de troca de dados para ferramentas de engenharia de sistemas) [Sys07].

A Figura 2.2 ilustra o relacionamento da UML com a SysML. A SysML reutiliza um subconjunto dos modelos da UML 2.0, chamado de UML4SysML e provê mais duas construções adicionais (diagramas de requisito e paramétrico), resultando num total de 9 tipos de diagramas.

A SysML é composta por 9 tipos de diagramas, que podem ser divididos em três partes: estrutural, comportamental e genérico, como descritos a seguir [Sil06]:

- **Estrutural.** As construções estruturais da SysML representam os elementos conceituais ou físicos. A parte estrutural da SysML é formada pelos seguintes diagramas: pacote, definição de bloco, bloco interno e paramétrico. O diagrama de pacote

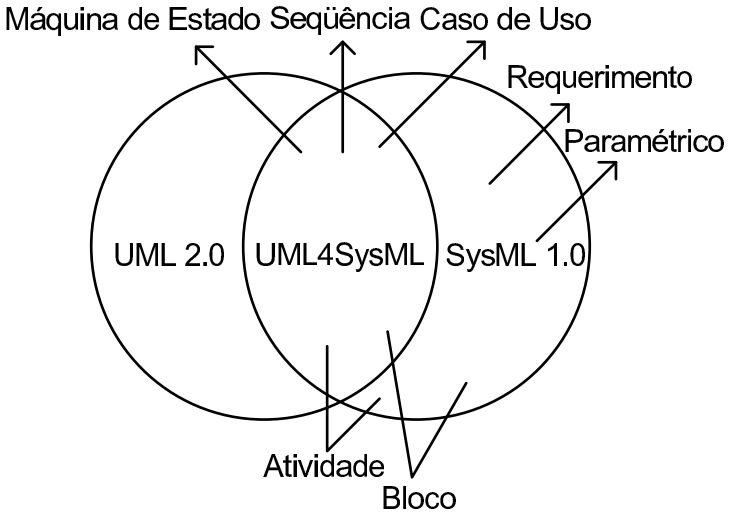


Figura 2.2: Relacionamento da UML com a SysML.

é utilizado para organizar o modelo. O diagrama de definição de bloco descreve o relacionamento entre os blocos. O diagrama de bloco interno representa a estrutura interna de um bloco em termos das suas propriedades e conexões. Finalmente, o diagrama paramétrico é usado para expressar as restrições impostas pelos sistemas.

- **Comportamental.** As construções comportamentais da SysML representam os comportamentos dos elementos no tempo e no espaço. A parte comportamental da SysML é formada pelos seguintes diagramas: atividades, seqüência, estados e caso de uso. O diagrama de atividades descreve o fluxo de controle e o fluxo de entradas e saídas entre as ações. O diagrama de seqüência mostra a colaboração dinâmica entre as várias entidades do sistema. O diagrama de estados representa uma máquina de estados, enfatizando o fluxo de controle de estado para estado, tipicamente utilizado para representar o ciclo de vida de um bloco. Finalmente, o diagrama de caso de uso mostra como o sistema a ser desenvolvido vai interagir com seu ambiente (usuários, sistemas e outros).

- **Genérico.** As construções genéricas da SysML representam as construções que podem ser aplicadas nas partes estruturais e comportamentais. A parte genérica da SysML é formada pelo diagrama de requisito, relacionamento de alocação e construções auxiliares para os modelos. O diagrama de requisitos representa um requisito baseado em texto (definido textualmente). O relacionamento de alocação representa as relações que mapeiam um elemento do modelo em outro. As construções auxiliares são elementos e notações para itens de fluxo, modelo de dado de referência, ponto de vista e observações, tipos de dados adicionais, dimensionamento de quantidades, distribuição probabilística e propriedade para restrição de valor.

A seguir, na Figura 2.3, é apresenta toda a estrutura dos diagramas da SysML. Alguns

diagramas foram importados dos diagramas da UML 2.0, sem modificações. Outros diagramas, no entanto, ou foram adicionados, ou modificados para atender as necessidades da engenharia de sistema.

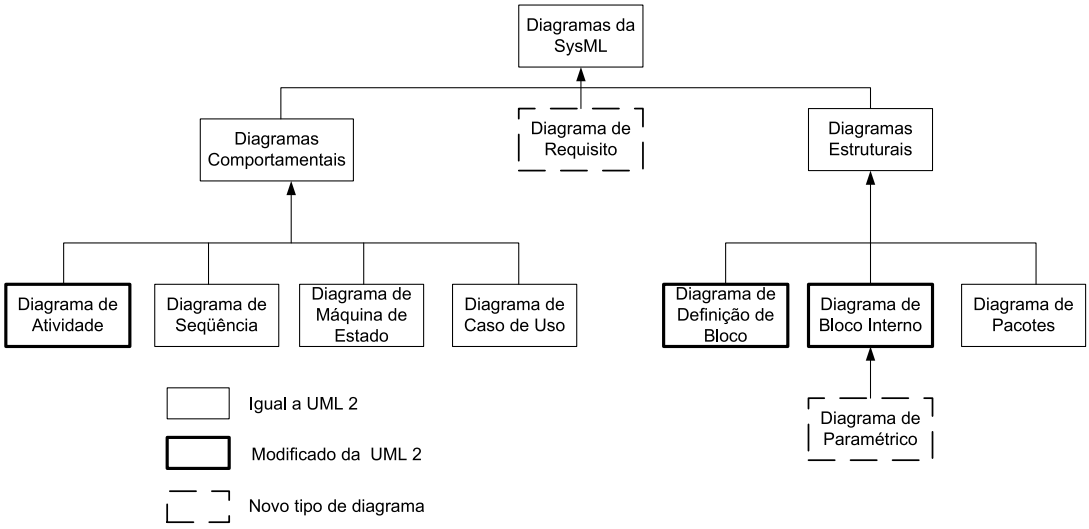


Figura 2.3: Estrutura da SysML.

Nas subseções seguintes os diagramas comportamentais da SysML (estados, atividades e seqüência) adotados nesta dissertação são brevemente descritos.

2.3.1 Diagrama de Estados

A UML 2.0 possui dois tipos de Diagrama de Estados (DE), o comportamental e o protocolo. O DE da SysML é importado do DE da UML 2.0 sem modificações ou extensões. No entanto, O DE da SysML não inclui o diagrama do tipo protocolo [Sys07]. O DE Comportamental da SysML é utilizado para representar o comportamento dinâmico do sistema. Ele descreve todos os estados possíveis em que uma entidade particular pode estar, e como os estados da entidade mudam em relação aos eventos que a atingem, ou seja, os DEs apresentam as seqüências de estados que uma entidade assume em sua existência em resposta a estímulos recebidos. As entidades representam os elementos que são modelados.

Os DEs são baseados nos *statecharts* [Har87], que são um formalismo visual concedido por David Harel para especificar sistemas de tempo-real do tipo reativo. Sistemas do tipo reativo são caracterizados pelo fato de estar continuamente reagindo a estímulos externos e internos. Exemplos: sistemas embutidos, aeronaves e máquinas industriais, sistemas de controle de tráfego aéreo e urbano, entre outros. Os *statecharts*, por sua vez, são uma evolução dos clássicos Diagramas de Transição de Estados (DTE). No entanto, os DTE não permitem uma visão de profundidade, hierarquia ou modularidade dos sistemas modelados. Em outras palavras, a medida em que a complexidade do sistema cresce linearmente, o número de estados e transições cresce exponencialmente, gerando diagramas

grandes e confusos. Além disso, os DTEs não suportam concorrência, sendo seqüenciais por natureza.

Os DEs são compostos basicamente pelos seguintes elementos [GUE04]:

- **Estado.** Representa a situação em que uma entidade se encontra em um determinado momento durante sua participação em um processo. A SysML define três tipos de estados: estado simples, estado composto e sub-máquina. Estados simples são o mais simples de todos os estados; eles não possuem sub-estados, conforme descrito na Figura 2.4. Por outro lado, estados compostos são estados que contêm dois ou mais estados. Finalmente as sub-máquinas, que são estados compostos seqüenciais ou concorrentes, cujos sub-estados são descritos em outro diagrama.
- **Transição.** Representa um evento que causa uma mudança no estado da entidade, gerando um novo estado. Uma transição é representada por uma seta ligando dois estados (ver Figura 2.4).
- **Estado Inicial.** É um pseudoestado cuja função é determinar o início de um DE ou de um estado composto (ver Figura 2.4).
- **Estado Final.** O estado final é um estado cuja função é determinar o fim de um DE ou de um estado composto (ver Figura 2.4).

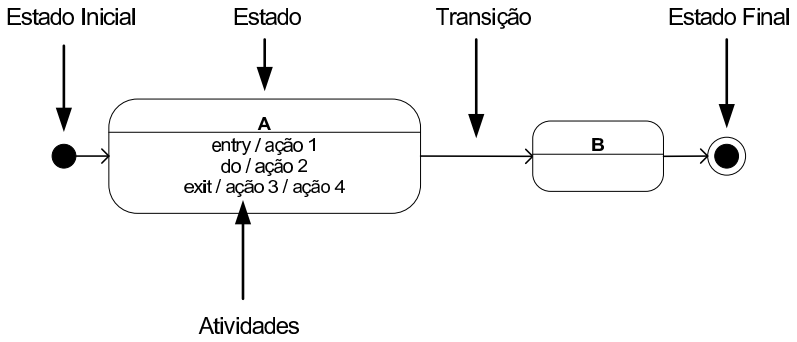


Figura 2.4: Exemplo do diagrama de estados.

O Capítulo 4 apresenta os principais elementos que compõem os DE, assim como o processo de mapeamento em uma rede de Petri temporizada.

2.3.2 Diagrama de Atividades

Segundo [GUE04], o Diagrama de Atividades (DA) era considerado um caso especial do diagrama de estados. No entanto, a partir da UML 2.0 tornou-se um diagrama totalmente independente, deixando inclusive de se basear em máquinas de estados e passou a se basear em redes de Petri.

De forma semelhante aos DEs, os DAs são utilizados para a modelagem dos aspectos dinâmicos de um sistema. No entanto, o DA descreve o fluxo de controle de um processo. Além disso, esse diagrama possui muitos recursos que permitem representar estruturas complexas, tais como processos paralelos e condicionais, exceções, eventos, entre outros. É importante destacar que o DA possui o enfoque no fluxo entre atividades, enquanto o DE no fluxo de estados.

As modificações no diagrama de atividades da SysML em relação ao diagrama de atividades da UML 2.0 incluem os seguintes aspectos: controle com dados, sistemas contínuos, probabilidade, atividades como blocos e linha do tempo. Mais informações podem ser encontradas em [Sys07].

Os DAs são compostos basicamente pelos seguintes elementos:

- **Atividades.** Atividade refere-se à execução de um processamento não atômico, envolvendo uma ou mais ações. Uma ação consiste em um processamento que resulta em uma mudança de estado no sistema. Ações e atividades têm a mesma representação gráfica (ver Figura 2.5).
- **Transição.** Representa o fluxo de uma atividade/ação para outra (ver Figura 2.5).
- **Decisão.** Representa um ponto do fluxo de controle, onde deve ser realizada uma tomada de decisão (ver Figura 2.5).
- **Bifurcação.** Representa a divisão de um fluxo de controle em dois ou mais fluxos de controle concorrentes e independentes (ver Figura 2.5).
- **Sincronização.** Representa a sincronização de dois ou mais fluxos concorrentes (ver Figura 2.5).

O Capítulo 4 apresenta em detalhes os principais elementos que compõem o diagrama de atividade, assim como o processo de mapeamento em uma rede de Petri temporizada.

2.3.3 Diagrama de Seqüência

O Diagrama de Seqüência (DS) descreve a seqüência temporal da troca de mensagens/chamadas entre as entidades. Na SysML, o DS é importado do DS da UML 2.0 sem modificações ou extensões.

Em um diagrama de seqüência, os seguintes elementos podem ser encontrados:

- **Linha de Vida.** São representados por linhas verticais tracejadas, que descrevem o tempo de vida de uma entidade. A linha de vida também é conhecido por *lifeline* (ver Figura 2.6).
- **Ativação.** Indica que a entidade está ocupada realizando uma tarefa. As ativação são representadas por barras verticais preenchidas nas *lifelines* (ver Figura 2.6).

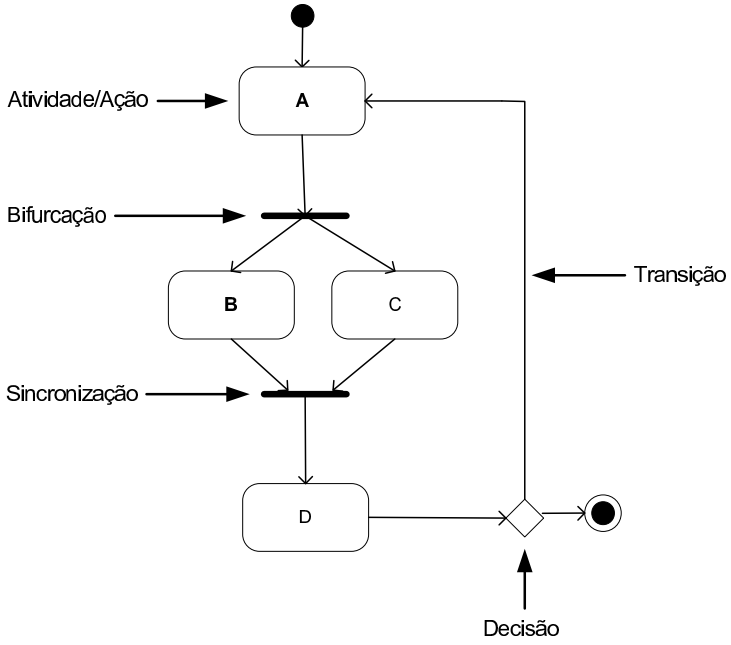


Figura 2.5: Exemplo do diagrama de atividade.

- **Mensagens/Chamadas.** Representam uma comunicação entre as entidades (emissor e receptor). Uma mensagem/chamada é representada por uma seta horizontal do emissor para o receptor (ver Figura 2.6).
- **Mensagem de Retorno.** Indica o retorno de uma mensagem. São representadas por linhas horizontais tracejadas (ver Figura 2.6).

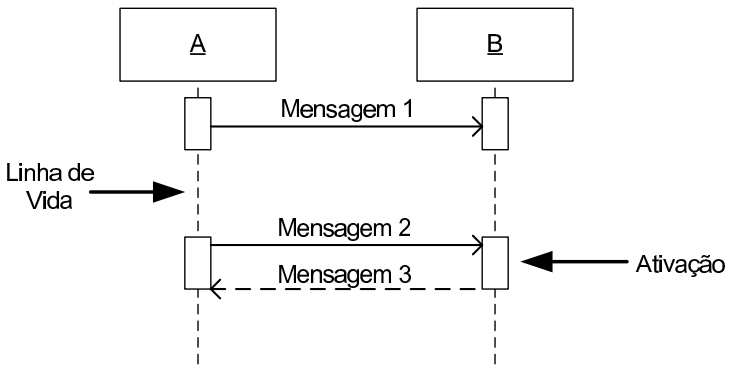


Figura 2.6: Exemplo do diagrama de seqüencia.

De forma semelhante aos demais diagramas, o Capítulo 4 apresenta em detalhes os principais elementos que compõem o diagrama de seqüência, assim como o processo de mapeamento em uma rede de Petri temporizada.

2.4 MARTE

A linguagem de modelagem UML possui um mecanismo que permite a extensão da linguagem, denominado *profile*. Através da criação desse *profile*, os projetistas podem criar elementos que expressam conceitos específicos de um determinado domínio, sem que haja a necessidade da criação de uma nova linguagem de modelagem. Nesse sentido, em Junho de 2007, a OMG criou um novo *profile* da UML 2.0, cuja principal função é auxiliar a construção de modelos que possam ser usados para fazer precisões quantitativas relativas às características de sistemas embarcados de tempo-real, levando em consideração tanto as características de *hardware* quanto de *software*. Esse novo *profile* é chamada de MARTE (*Modeling and Analysis of Real-time and Embedded systems*). É importante destacar que esse *profile* é compatível com os demais *profiles* ou padrões existentes, tais como: SysML, SAE AADL, EAST-ADL v2, OSEK, ARINC 653, e POSIX.

Os principais objetivos de MARTE são:

- permitir a especificação das informações quantitativas diretamente nos diagramas da SysML/UML;
- permitir a realização das análises, a despeito das diversas características dos sistemas embarcados de tempo-real.
- facilitar a comunicação entre os envolvidos na construção do sistema por meio de uma linguagem padronizada;
- permitir interoperabilidade entre diferentes ferramentas de análise e projeto.

Em princípio, a SysML é considerada totalmente adequada para modelar sistemas embarcados de tempo-real. No entanto, a adoção de MARTE é fundamental para padronizar as notações, e assim, facilitar tanto o processo de modelagem, quanto o processo automático de geração de modelo e código. Os mecanismos de extensão (estereótipos, valores marcados e restrições) são utilizados para garantir que a SysML ou UML possa evoluir, sem a necessidade de redefinir sua estrutura para atender as necessidades de mudança, assim como atender aos novos requisitos que possam surgir durante o desenvolvimento do projeto.

Estereótipos são usados para marcar, classificar, ou introduzir novos elementos na hierarquia de classes do metamodelo para permitir que usuários estendam a capacidade de modelagem da linguagem. Metamodelo é o modelo que descreve os elementos de uma linguagem. Os valores marcados especificam valores e palavras relativas às características do sistema, sendo normalmente associados a um estereótipo específico. Restrições podem ser associadas aos estereótipos restringindo os elementos correspondentes do metamodelo. As restrições podem ser expressas em qualquer linguagem, incluindo a linguagem natural ou a *Object Constraint Language* (OCL).

A fim de facilitar o uso das anotações do *profile* MARTE, que são usadas com os mais diversos objetivos (ex.: escalonamento de tarefas, avaliação de desempenho dos sistemas

de tempo-real, entre outros), a especificação foi dividida em unidades de extensão, ou seja, pacotes nos quais contêm anotações para especificar as características em comum dos ERTS. A Tabela 3.1 apresenta todas as unidades de extensão de MARTE. É importante ressaltar que alguns desses pacotes devem ser usados em conjunto, com o objetivo de se obter uma especificação mais completa.

Tabela 2.1: Unidades de extensão.

Abreviatura	Nome
NFP	Propriedades Não Funcionais
Time	Modelagem de Tempo
GRM	Modelagem de Recurso Genérico
Alloc	Modelagem de Alocação
GCM	Modelo de Componente Genérico
HLAM	Modelagem de Aplicação de Alto Nível
SRM	Modelagem de Recurso de *Software*
HRM	Modelagem de Recurso de *Hardware*
RTM	Modelagem de Objetos de Tempo-Real
GQAM	Modelagem de Análise Quantitativa Genérica
SAM	Modelagem de Análise de Escalonamento
PAM	Modelagem de Análise de Desempenho
VSL	Linguagem de Especificação de Valor
CHF	Recursos de Tratamento de *Clock*
RSM	Modelagem de Estrutura Repetitiva

Fonte: [MAR07, p. 2]

Este trabalho tem o objetivo de utilizar a unidade de extensão para Modelagem de Recursos Genéricos (GRM), mais especificamente o estereótipo *ResourceUsage*. O GRM tem o objetivo de oferecer conceitos gerais que são necessários para modelar plataformas de aplicações embarcadas de tempo-real. Os valores marcados do estereótipo *ResourceUsage*, que são utilizados nesta dissertação, são descritos na Tabela 2.2.

Tabela 2.2: Valores marcados do estereótipo *ResourceUsage* usados nesta dissertação.

Valores Marcados	Descrição
execTime	Tempo de execução de uma tarefa.
energy	Consumo de energia para a realização de uma tarefa.

Fonte: [MAR07, p. 92]

A Figura 2.7 ilustra um exemplo de um diagrama de atividade com restrições de tempo e anotações de energia especificados pelo *profile* MARTE. Para esse exemplo o esteriótipo (*ResourceUsage*) e os valores marcados (*execTime* e *energy*) foram usados.

O estereótipo *ResourceUsage* descreve, respectivamente, o tempo execução e o consumo de energia da atividade A, neste caso, 10 segundos e 20 *joules*. Os valores marcados são: $\{execTime = (10,'s')\}$ e $\{energy = (20,'j')\}$. De forma semelhante, a transição *t1* que sai da atividade A com destino à atividade B, possui também um tempo execução e um consumo de energia de, respectivamente, 30 segundos e 40 *joules* associados a transição *t1*. Os valores marcados são: $\{execTime = (30,'s')\}$ e $\{energy = (40,'j')\}$. Mais informações sobre todos os estereótipos e valores marcados suportados por MARTE podem ser encontrados em [MAR07].

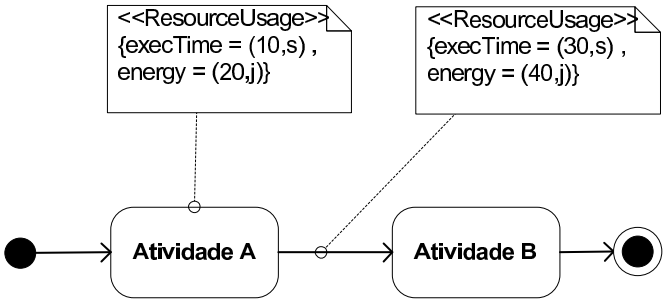


Figura 2.7: Diagrama de atividade com anotações de MARTE.

2.5 REDES DE PETRI

O conceito das RdP foi inicialmente introduzido por Carl Adam na sua tese de doutoramento intitulada de *Kommunikation mit Automaten* (Comunicação com Autômatos), em 1962 na Universidade de Damstadt, Alemanha [Pet62]. Deste então, esse formalismo tem sido amplamente utilizado em diferentes áreas, tais como Ciência da Computação, Engenharia Elétrica, Administração, Química, entre outras.

Diversas variantes do modelo de RdP clássico têm sido desenvolvidas ao longo do tempo, tais como redes temporizadas [MF76], estocásticas [Mar89], alto-nível [Jen91] e orientadas a objetos [Jan]. Isso é devido a necessidade de suprir as diferentes áreas de aplicação, além de prover facilidades de comunicação e transferência de métodos e ferramentas de uma área para outra.

As vantagens do uso das RdPs são diversas, entre elas é possível destacar as seguintes [GV01]:

- RdPs fornecem uma formalismo de modelagem que permite sua representação gráfica e são fundamentadas matematicamente;
- RdPs fornecem mecanismos de refinamento e abstração que são de grande importância para o projeto de sistemas complexos;
- existe uma grande variedade de ferramentas disponíveis para as RdPs, tanto comerciais quanto acadêmicas para modelagem, análise e verificação;

- RdPs têm sido utilizadas nas mais diversas áreas. Portanto, vários resultados são encontrados na literatura para os diversos domínios da sua aplicação;
- existem várias extensões do modelo básico das RdPs.

As RdPs são compostas pelos seguintes elementos [MLC96, Mur89]:

- **Lugares.** Representam os elementos passivos da rede, tendo como principal função o armazenamento de *tokens*, os quais são removidos e adicionados à medida que as transições são disparadas. Graficamente, os lugares são representados por círculos (ver Figura 2.8 (a)).
- **Transições.** Representam os elementos ativos da rede, ou seja, as ações realizadas pelo sistema. Para que uma transição esteja habilitada, é necessário que todas as suas pré-condições sejam satisfeitas, caso uma pré-condição não seja satisfeita, a transição estará desabilitada. Uma vez que a transição satisfaz todas as pré-condições, ela poderá disparar, removendo uma determinada quantidade de *tokens* dos lugares e colocando em outros, gerando assim as pós-condições. Graficamente, são representadas por traços ou barras (ver Figura 2.8 (b)).
- **Arcos.** Representam o fluxo dos *tokens* pela rede (ver Figura 2.8 (c)).
- **Tokens.** Representam o estado em que o sistema se encontra em determinado momento (ver Figura 2.8 (d)).

Figura 2.8: Elementos de uma rede de Petri.

Na representação gráfica, as transições e os lugares podem ser conectados por múltiplos arcos (arcos multivalorados) que podem ser compactados em um único arco rotulado (Figura 2.9).

Formalmente, as redes de Petri podem ser definidas conforme descreve a Definição 2.1.

Definição 2.1. (Redes de Petri) Uma rede de petri é uma 5-tupla, $PN = (P, T, F, W, M_0)$, onde:

- $P = \{p_1, p_2, ..., p_n\}$, é um conjunto finito de lugares;
- $T = \{t_1, t_2, ..., t_n\}$, é um conjunto finito de transições;

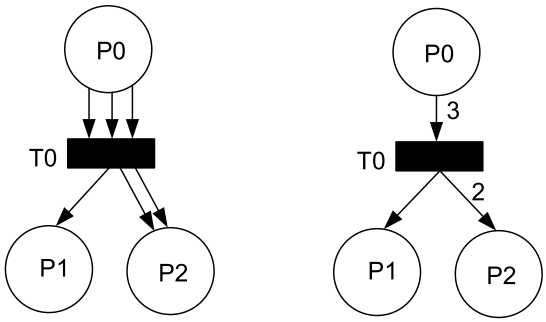


Figura 2.9: Arco multivalorado.

- $F \subseteq (P \times T) \cup (T \times P)$, é um conjunto de arcos;
- $W : F \rightarrow \{1, 2, 3, 4...n\}$, é a função de atribuição de peso aos arcos.
- $M_0 : P \rightarrow \{0, 1, 2, 3...\}$, é a marcação inicial, onde: $P \cap T = \varnothing$ e $P \cup T \neq \varnothing$.

A seguir, na Figura 2.10, é apresentado um exemplo de uma RdP, na qual o funcionamento de uma lâmpada é modelado. Nesse modelo, os lugares e as transições representam, respectivamente, os estados da lâmpada (*aceso* ou *apagado*) e as ações que alteram o estado da lâmpada (*ligar* ou *desligar*). O estado atual do modelo é representado por uma marca (*token*) no lugar correspondente à situação atual do modelo, como descrito na Figura 2.10 (a). Visto que o estado atual do modelo é *aceso*, a única transição que poderá ser disparada é *desligar*. Uma vez que essa transição seja disparada, então o modelo passará do estado *aceso* (ver Figura 2.10 (a)) para o estado *apagado* (ver Figura 2.10 (b)).

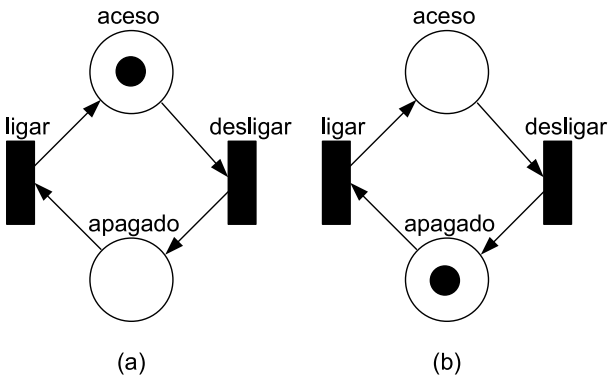


Figura 2.10: Exemplo de uma rede de Petri.

Dependendo do sistema modelado, as transições e os lugares de saída e entrada podem ter significados diferentes [Mur89], conforme descritos na Tabela 2.3.

Existem outras formas de representar os elementos de uma RdP. Por exemplo, o conjunto de lugares de entrada e de saída de uma determinada transição pode ser definido

Tabela 2.3: Interpretações para os lugares e transições.

Lugares de Entrada	Transições	Lugares de Saída
pré-condições	eventos	pós-condições
dados de entrada	passo e computação	dados e saída
sinal de entrada	processamento de sinal	sinal de saída
disponibilidade de recursos	tarefa	liberação de recursos
condição	cláusula lógica	conclusões
buffers	processador	buffers

Fonte: [Mur89, p. 542]

conforme apresenta a Definição 2.2. Similarmente, o conjunto de transições de entrada e de saída de um determinado lugar pode ser definido conforme apresenta a Definição 2.3.

Definição 2.2. (Lugares de Entrada e de Saída) Os conjuntos de lugares de entrada e de saída de uma transição $t_i \in T$ podem ser representados da seguinte forma:

$$\bullet t_j = \{p_i \in P | I(p_j, t_i) \in F\}$$

$$t_j \bullet = \{p_i \in P | O(t_j, p_i) \in F\}$$

Definição 2.3. (Transições de Entrada e de Saída) Os conjuntos de transições de entrada e de saída de um lugar $p_i \in P$ podem ser representados da seguinte forma:

$$\bullet p_i = \{t_j \in T | O(t_j, p_i) \in F\}$$

$$p_i \bullet = \{t_j \in T | I(p_i, t_j) \in F\}$$

2.5.1 Rede de Petri Marcada

Uma marca (*token*) é um conceito primitivo em redes de Petri, tal qual lugar e transição. As marcas são informações atribuídas aos lugares. Uma marcação associa um k (inteiro não-negativo) a cada lugar da rede. Abaixo são apresentadas as seguintes definições formais: marcação, vetor de marcação e rede de Petri marcada, respectivamente, nas Definições 2.4, 2.5 e 2.6.

Definição 2.4. (Marcação) Seja P o conjunto de lugares de uma RdP. Define-se formalmente marcação como uma função que mapeia o conjunto de lugares P a inteiros não negativos $M : P \rightarrow \mathbb{N}$.

Definição 2.5. (Vetor Marcação) Seja P o conjunto de lugares de uma RdP. A marcação pode ser definida formalmente como um vetor $M = (M(p_1), ..., M(p_n)))$, onde $n = \#(P)$, para todo $p_i \in P$, tal que $M(p_i) \in \mathbb{N}$.

Definição 2.6. (Rede Marcada) Define-se uma rede de Petri marcada pela dupla $RM(R; M_0)$, onde R é a estrutura da rede e M_0 é a marcação inicial.

2.5.2 Transições: Habilitação e Disparo

De forma geral, o comportamento dos mais diversos sistemas pode ser descrito em termo dos seus possíveis estados internos e os processos de mudança desses estados. Para simular o comportamento dinâmico de um sistema, a marcação em uma rede de Petri é modificada de acordo com as regras descritas a seguir:

- uma transição $t \in T$ é habilitada, se cada lugar de entrada p da transição t ($\bullet t$) é marcado como no mínimo $w(p,t)$ *tokens*;
- uma transição habilitada pode ou não disparar. O disparo da transição depende da RdP que está sendo considerada, em outras palavras, a transição pode estar associada a um guarda temporal (redes temporais) ou a um predicado lógico, que indicará quando um evento associado deve ocorrer;
- o disparo de uma transição habilitada t remove $w(p,t)$ *tokens* dos lugares de entrada p da transição $t(\bullet t)$, e adiciona $w(p,t)$ *tokens* nos lugares de saída p da transição $t(t\bullet)$.

Definição 2.7. (Habilitação de Transição) Seja $PN = (P, T, F, W, M_0)$ uma rede, $t \in T$ uma transição e M_k uma marcação. Se $M_k[t\,, M_k(p_i) \geq I(p_i, t), \forall p_i \in P$.

Definição 2.8. (Regras de Disparo de Transições) Seja $PN = (P, T, F, W, M_0)$ uma rede, $t \in T$ uma transição e uma marcação M. A transição t pode disparar quando ela está habilitada. Disparando uma transição habilitada, a marcação resultante é $M = M_0 - I(p_j, t) + O(p_j, t) \forall p \in P$. Se uma marcação M é alcançada por M_0 pelo disparo de uma transição t_i, ela é denotada por $M_0[t > M$.

Se uma transição t_1 está habilitada para uma marcação M e uma segunda transição t_2 está habilitada para a marcação M_1, obtida após o disparo de t_1, dizemos que a seqüência $sq = t1; t2$ está habilitada para M. Em outras palavras, se $M[t_1 > M_1$ e $M_1[t_2 > M_2$ então $M[t_1; t_2 > M_2$. Portanto, é designado o disparo de uma seqüência $sq \in T$ por $M[sq > M'$.

Definição 2.9. (Seqüência Disparáveis) A seqüência sq está habilitada, possibilitando a obtenção de uma marcação M'', ($M[sq > M''$) se, e somente se, ocorrem os casos abaixo:

- $sq = \lambda$, onde λ é uma seqüência vazia, tal que $M'' = M$;
- $sq = sq't$, onde sq é uma seqüencia de transições $t \in T$ e existe M' tal que $M[sq' > M'$ e $M'[t > M''$.

Se $M[sq > M''$, diz-se que sq é uma seqüência disparável para M.

2.5.3 Grafo de Alcançabilidade

Geralmente um grafo rotulado e direcionado é adotado para representar todas as possíveis marcações que a rede de Petri pode alcançar. Este grafo é geralmente chamado de grafo de alcançabilidade.

Definição 2.10. (Grafo de Alcançabilidade) Podemos definir um grafo de alcançabilidade como sendo uma tupla (V, E), onde V representa o conjunto de vértices representados pelas marcações possíveis, e E é o conjunto de arestas rotuladas.

Como exemplo considere $M = \{m_o = |1,0,0|, m_1 = |0,1,0|, m_2 = |0,0,1|\}$ sendo o conjunto das marcações alcançáveis da rede de Petri representada na Figura 2.11. O respectivo grafo de alcançabilidade é representado na Figura 2.12.

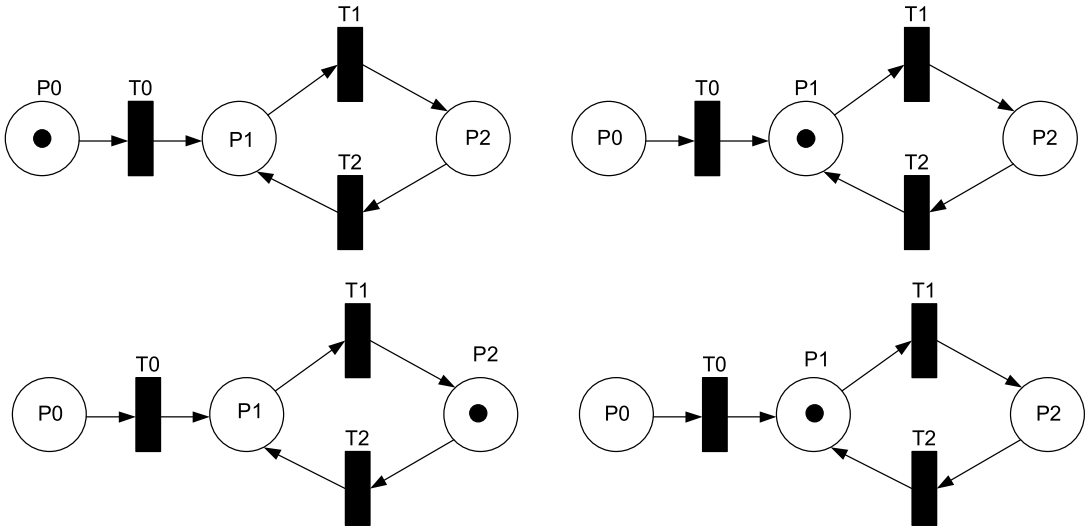


Figura 2.11: Exemplo de rede de Petri.

2.5.4 Redes Elementares

As redes elementares são blocos básicos que permitem a modelagem de sistemas mais complexos. Nesta seção são apresentadas algumas das redes elementares, tais como seqüência, distribuição, junção, escolha não-determinística, atribuição e confusão.

2.5.4.1 Seqüência A seqüência é uma rede que representa ações consecutivas, desde que uma dada condição seja satisfeita. Em outras palavras, após a execução de cada ação, uma nova condição poderá ser disparada, permitindo assim a execução de uma nova ação. A Figura 2.13 apresenta um exemplo dessa rede, onde um *token* no lugar p_0 habilita a transição t_0, e com o disparo dessa transição uma nova condição é estabelecida (p_1 é marcado). Essa nova condição pode permitir o disparo de uma nova condição associada ao lugar p_1.

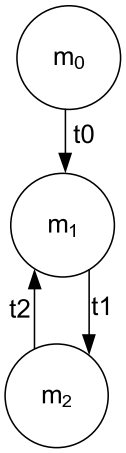


Figura 2.12: Exemplo de grafo de alcançabilidade.

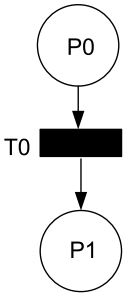


Figura 2.13: Seqüência.

2.5.4.2 Distribuição Esta rede permite a criação de processos paralelos a partir de um processo pai. Como apresentado na Figura 2.14, o disparo da transição t_0 adiciona um *token* no lugar p_1 e outro no lugar p_2. Essas novas condições (p_1 e p_2) permitem a execução de novos processos paralelos.

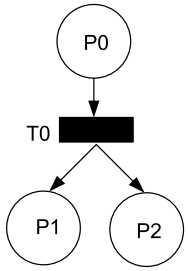


Figura 2.14: Distribuição.

2.5.4.3 Junção Esta rede modela a sincronização de processos paralelos (ver Figura 2.15). Ela combina duas ou mais redes, permitindo que outro processo continue sua execução somente após o término de todos os processos paralelos que o antecedem. Como

demonstrado na Figura 2.15, a transição t_0 estará habilitada se ambas as pré-condições contiverem tokens (p_0 e p_1). Se essa condição for satisfeita, então a transição t_0 poderá ser disparada, retirando um *token* dos lugares p_0 e p_1 e colocando em p_2.

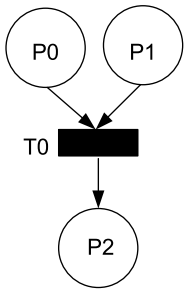


Figura 2.15: Junção.

2.5.4.4 Escolha Não-Determinística Nesta subseção é apresentada uma rede elementar que pode ser denominada de conflito, escolha ou decisão, dependendo da aplicação. A Figura 2.16 representa a escolha não determinística, onde o disparo de uma transição desabilita o disparo de uma outra transição. Havendo um token em p_O, t_0 e t_1 torna-se conflitante, isto é, o disparo de uma transição elimina a possibilidade da outra.

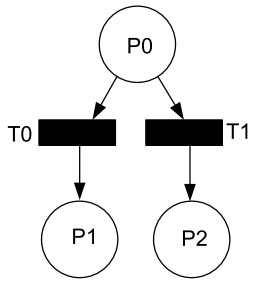


Figura 2.16: Escolha.

2.5.4.5 Atribuição Atribuição é uma rede elementar que permite que dois ou mais processos habilitem um terceiro processo. Na Figura 2.17, tanto a transição t_0 quanto a transição t_1 são independentes, porém ambas têm um lugar de saída em comum. Portanto, após o disparo de qualquer uma dessas transições, cria-se uma condição (p_2 é marcado) que possibilita o disparo de uma outra transição.

2.5.4.6 Confusion Esta rede modela a situação onde dois eventos estão ao mesmo tempo em conflito e em concorrência. A Figura 2.18 apresenta dois tipos de *confusion*: (a) simétrico e (b) assimétrico. No modelo da Figura 2.18 (a) ambas as transições t_0 e t_2 são concorrentes, e cada uma dessas transições está em conflito efetivo com t_1, pois o disparo de t_1 impossibilita o disparo de t_0 e t_2. Por outro lado, na Figura 2.18 (b), t_0 e t_2 são concorrentes, no entanto se t_2 disparar antes de t_0, t_0 e t_1 haverá conflito efetivo.

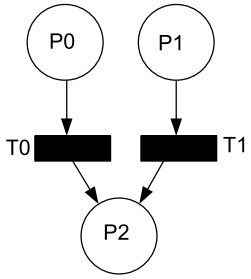


Figura 2.17: Atribuição.

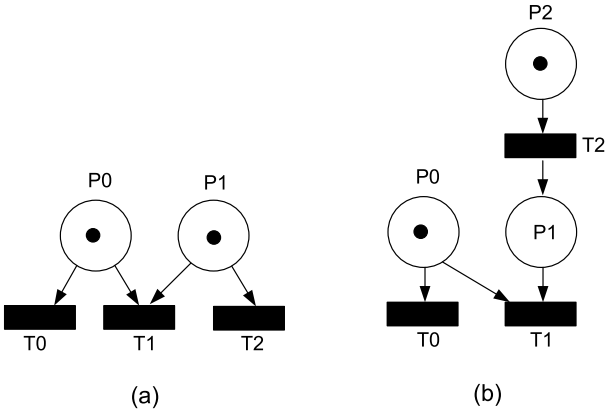


Figura 2.18: Confusion simétrico e assimétrico.

2.5.5 Exemplos de Modelagens com Redes de Petri

Nesta seção são apresentados alguns exemplos clássicos e seus respectivos modelos em redes de Petri. Esses modelos são construídos em função das redes elementares apresentadas na Seção 2.5.4.

2.5.5.1 Processos Paralelos Esta subseção apresenta a modelagem de processos paralelos, onde o modelo do processo global é obtido através da aplicação de regras de construção das redes elementares, tais como: seqüência, distribuição e junção. A Figura 2.19 apresenta um exemplo de atividade paralela, onde as transições t_1 e t_3 representam as atividades paralelas. Ao se disparar a transição t_0, criam-se duas marcas nos lugares p_1 e p_5, e então, é possível executar as atividades t_1 e t_3 de forma independente. O disparo da transição t_2, no entanto, depende de duas pré-condições, ou seja, que haja marca no lugar p_2 e p_4. Uma vez disparado a transição t_2, então as atividades são sincronizadas em p_3.

2.5.5.2 Exclusão Mútua Às vezes, durante a execução das atividades paralelas, é necessário que as atividades do processo coopere entre si para a obtenção de uma solução conjunta. Essa cooperação deve ser feita de maneira mutuamente exclusiva para evitar

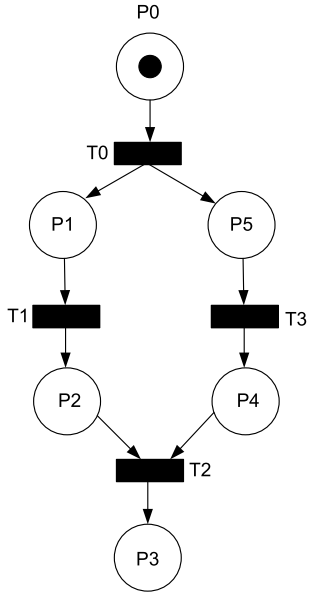


Figura 2.19: Processos Paralelos

resultados indesejáveis. A Figura 2.20 apresenta um exemplo de duas atividades paralelas dividindo um recurso em comum. Esse recurso é representando por um *token* no lugar p_8, onde só pode ser acesso por uma atividade por vez.

2.5.5.3 Protocolo de Comunicação Protocolo de comunicação é uma área onde as RdP têm sido largamente utilizadas para representar e especificar características dos sistemas, assim como análises de propriedades.

Entidades comunicantes podem ser modeladas de diversas formas: (i) uma única transição representando a comunicação (ver Figura 2.21 (a)); (ii) a mensagem sendo explicitamente representada por um lugar (ver Figura 2.21 (b)); ou (iii) através de lugares que representam tanto o envio da mensagem (*send*) quanto o reconhecimento da mensagem (*acknowledgement*) (ver Figure 2.21 (c)).

2.5.6 Propriedades das Redes de Petri

Diversas propriedades podem ser obtidas a partir dos sistemas modelos, permitindo assim revelar as mais diversas características do sistema. Essas propriedades podem ser subdivididas em comportamentais e estruturais, as quais são descritas nas subseções seguintes.

2.5.6.1 Propriedades Comportamentais Essa subseção descreverá as principais propriedades comportamentais baseadas em [Mur89]. As propriedades comportamentais são aquelas que dependem da marcação.

Alcançabilidade

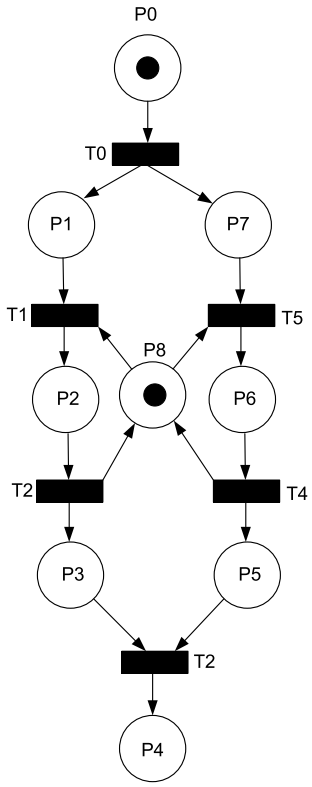


Figura 2.20: Exclusão Mútua

A propriedade de alcançabilidade indica a possibilidade de atingir uma determinada marcação pelo disparo de um número finito de transições a partir de uma dada marcação inicial. Uma marcação M_0 é dita alcançável a partir de M_i, se existir uma seqüência de disparo que transforme M_0 em M_i. A seqüência de disparo é denotada pelo conjunto $\sigma = \{t1, t2, ..., tn\}$. Nesse caso, M_i é alcançável a partir de M_0 por σ. Onde σ é formalmente descrito por $M_0[\sigma > M_i$.

Limitação e Safeness

Uma rede é dita *k-limitada* se todos os seus lugares forem limitados, ou seja, o número de *tokens* em cada lugar não deve ultrapassar um número finito k, para qualquer marcação alcançável a partir de M_0. Uma rede de Petri é dita safeness se $k = 1$.

Liveness

Uma rede é dita *live* se, não importa quais marcações sejam alcançáveis a partir de um marcação inicial m_0, for possível disparar qualquer transição através do disparo de alguma seqüência de transições $L(M_0)$. O conceito de *deadlock* está fortemente conectado ao conceito de *liveness*. No entanto, o fato de um sistema ser livre de *deadlock* não resulta que este seja *liveness*, contudo um sistema *liveness* implica em um sistema livre de *deadlocks*. A análise de *liveness* de uma rede permite verificar se os eventos modelados efetivamente ocorrem durante o funcionamento do sistema, ou se foram definidos eventos mortos no modelo.

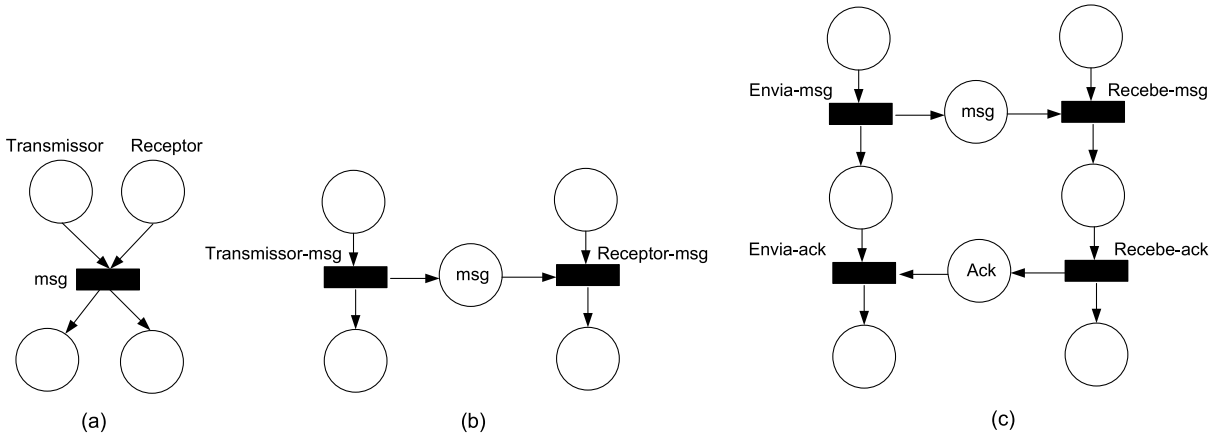


Figura 2.21: Protocolos de Comunicação

Liveness é uma propriedade fundamental para avaliar os sistemas de tempo-real. Contudo, é impraticável observar essa propriedade em sistemas muito complexos. Dessa forma, a propriedade *liveness* pode ser definida em níveis, conforme apresentados abaixo:

- morta (L0-*live*). Se t nunca pode ser disparada em qualquer seqüencia $L(M_0)$;
- L1-*live* (potencialmente disparável). Se t pode ser disparável em pelo menos alguma seqüência $L(M_0)$;
- L2-*live*. Se dado um inteiro positivi k, t puder ser disparado pelo menos k vezes em alguma seqüência $L(M_0)$;
- L3-*live*. Se t aparece um número infinito de vezes em alguma seqüência de disparo $L(M_0)$;
- L4-*live* ou simplesmente *live*. Se t é potencialmente disparável para todas as marcações da rede, ela é dita *liveness*.

Cobertura

A propriedade de cobertura está fortemente conectada ao conceito de alcançabilidade e *liveness*. Quando se deseja saber se alguma marcação M_i, pode ser obtida a partir de uma marcação M_j, temos o problema denominado cobertura de uma marcação. Uma marcação M_i é dita coberta se existe uma marcação M_j tal que $M_j > M_i$. Fora isso, em alguns sistemas, deseja-se apenas observar o comportamento de determinados lugares. Para isso, restringe-se a pesquisa a apenas um conjunto de lugares de particular interesse (cobertura de submarcações).

Reversibilidade e Home State

Uma rede é dita reversível se, para cada marcação M em $R(M_0)$, M_0 é alcançável a partir de M. Assim, a rede possui a capacidade de retornar à marcação inicial. Além disso, em

algumas aplicações não é necessário voltar à marcação inicial, mas sim a uma marcação específica. Essa marcação específica é denominada *Home State*.

2.5.6.2 Propriedades Estruturais As propriedades estruturais são aquelas que não dependem da marcação, ou seja, possuem dependência exclusivamente da topologia da rede. Abaixo serão descritas as principais propriedades estruturais baseadas em [Mur89].

- **Limitação Estrutural.** Uma rede é dita limitada estrutural se o número de *tokens* é limitado para qualquer marcação inicial.
- **Conservação.** A conservação é uma importante propriedade das RdP, pois permite a verificação da não destruição de recursos através da conservação de *tokens*.
- **Repetitividade.** Uma rede é considerada repetitiva se para uma marcação e uma seqüência de transições disparáveis, todas as transições dessa rede são disparadas ilimitadamente.
- **Consistência.** Uma rede é dita consistente se dada uma seqüência de transições disparáveis a partir de uma marcação inicial M_0 retornar a M_0, porém todas as transições da rede são disparadas pelo menos uma vez.

2.5.7 Métodos de Análise

Os métodos de análise das RdP podem ser classificados em três grupos: análise baseada na árvore de cobertura, os métodos baseados na equação fundamental das RdP e as técnica de redução. Neste trabalho, são introduzidos os seguintes grupos: análise baseada na árvore de cobertura e as técnicas de redução.

2.5.7.1 Árvore de Cobertura O método de análise denominado árvore de cobertura baseia-se nas construções de uma árvore que possibilite a representação de todas as possíveis marcações de uma rede [MLC96].

Para uma dada rede de Petri, com uma marcação inicial, é possível obtermos diversas marcações para um grande número de transições potencialmente habilitadas. Essas marcações podem ser representadas por uma árvore, onde os nós são as marcações e os arcos as transições disparadas. A árvore de cobertura é um gráfico utilizado para representar finitamente um número infinito de marcações. Para possibilitar a representação finita das marcações, através da árvore, é utilizado o símbolo ω.

A árvore de cobertura pode ser construída seguindo-se o algoritmo abaixo:

1. Rotule a marcação inicial M_0 como a raiz e descrimine-a como nova.
2. Enquanto houver marcações novas, faça:

(a) Selecione uma nova marcação M.

i. Se M é idêntica a uma marcação no caminho desde a raiz até M, então rotule-a como antiga e selecione uma nova marcação.

(b) Se nenhuma transição está habilitada para M, rotule-a esta marcação como final.

(c) Enquanto houver transição habilitada para M, faça o seguinte para cada transição habilitada:

i. Obtenha a nova marcação M' que resulta do disparo de uma transição t habilitada para M.

ii. Se no caminho da raiz para a marcação M existe uma marcação M'' tal que para todo lugar $p \in P$ $M'(p) \geq M''(p)$ e $M' \neq M''$ (M'' é acessível de M') então substitui-se $M'(p)$ por ω para cada p tal que $M'(p) > M''(p)$.

iii. Introduza M' como um nó e crie um arco rotulado com t de M para M' e rotule a marcação M' como nova.

Para uma rede de petri limitada, a árvore de cobertura é denominada *árvore de alcançabilidade*, dado que esta contém todas as possíveis marcações da rede. A Figura 2.22 (b) mostra a árvore de cobertura para o exemplo da Figura 2.22 (a).

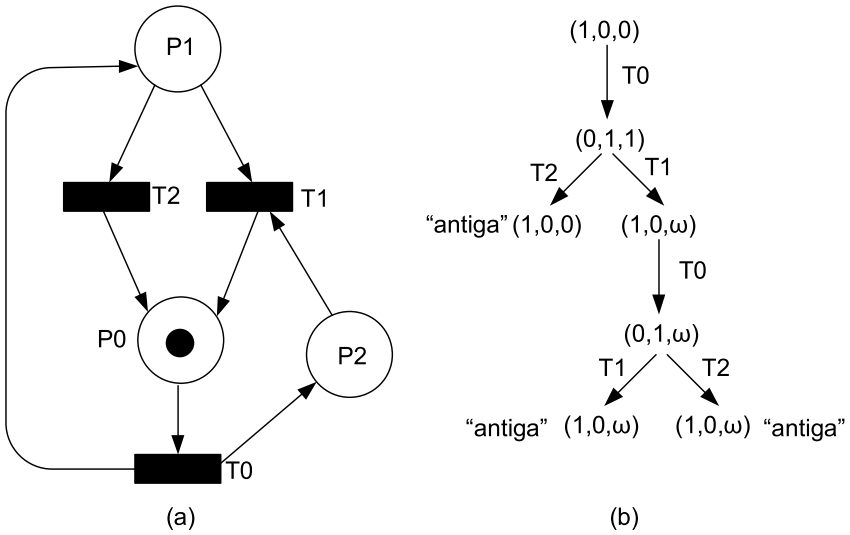


Figura 2.22: Árvore de cobertura.

Definição 2.11. (Árvore de Cobertura) Seja $RM = (P, T, F, W, M_0)$ uma rede de Petri marcada. Define-se árvore de cobertura pelo par $AC = (S, A)$, onde S representa as marcações e A os arcos rotulados por $t_j \in T$.

Algumas propriedades podem ser analisadas usando árvore de cobertura, tais como *limitação*, *segurança*, *deadlock* e *alcançabilidade*. A identificação dessas propriedades é realizada da seguinte forma:

- uma rede (N, M_0) é ilimitada, se ω não aparecer na árvore de cobertura;
- uma rede (N, M_0) é segura, se apenas 0´s e 1's aparecem na árvore de cobertura;
- uma transição t é morta, se ela não aparecer na árvore de cobertura;
- se uma marcação M é alcançável a partir de M_0, então existe o nó M', tal que $M \leq M'$.

O principal problema desse método consiste na alta complexidade computacional, mesmo quando se usam técnicas de redução das RdP.

2.5.7.2 Redução Reduções das RdP são transformações aplicadas ao modelo de um sistema com o objetivo de simplificá-lo, preservando as propriedades do sistema a ser analisado. Normalmente essa técnica é utilizada para facilitar a análise de sistemas complexos. Por outro lado, é possível transformar um modelo abstrato em um modelo refinado, mantendo-se suas propriedades [Mur89]. Existem várias técnicas de transformação para as RdPs. A Figura 2.23 mostra algumas delas, dentre elas:

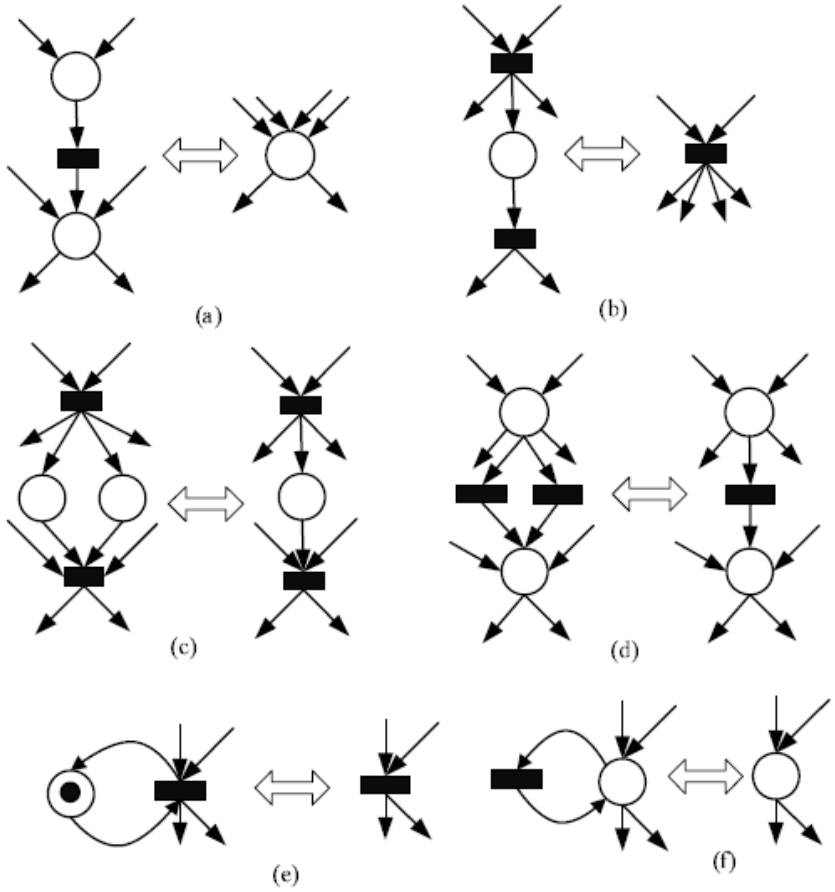
- Lugares em série, como apresentado na Figura 2.23 (a).
- Transições em série, como apresentado na Figura 2.23 (b).
- Lugares paralelos, como apresentado na Figura 2.23 (c).
- Transições paralelas, como apresentado na Figura 2.23 (d).
- Lugares de *self-loops*, como apresentado na Figura 2.23 (e).
- Transições de *self-loops*, como apresentado na Figura 2.23 (f).

2.5.8 Extensões Temporizadas das Redes de Petri

Originalmente as redes de Petri propostas por Adam Petri não possuíam notação de tempo ou probabilidade, devido às dificuldades que a temporização adicionaria a análise da rede. Os primeiros trabalhos utilizando as redes de Petri com tempo são os de P.M Merlin e D.J Faber [MF76] e J.D Noe e G.J Nutt [NN73]. Existem diferentes formas de incorporar tempo às redes de Petri, divergindo em relação à localização e ao tipo de tempo associado.

Quanto à localização, o tempo pode ser associado aos lugares, às transições e aos *tokens* [vdAvHR00].

- **Lugares.** O lugar permite que seus *tokens* sejam consumidos após um dado tempo que é associado ao lugar.



Fonte: [Mur89, p. 553]

Figura 2.23: Redução e refinamento para as redes de Petri.

- **Transições.** O disparo da transição ocorre depois de um tempo de atraso correspondente a partir do momento em que a transição se torna habilitada.
- **Tokens.** Os *tokens* possuem uma informação de tempo disponível que indica quando o *token* estará disponível para ser consumido.

Na maioria dos modelos das redes de Petri temporizadas, o tempo é associado às transições, visto que em poucos modelos o tempo está associado aos lugares ou arcos [vdAvHR00]. Quanto ao tipo de tempo, pode ser determinístico [Ram74, Zub80], intervalar [MF76] ou estocástico [Gia02, MBC$^+$98, MCB84, FN80].

- **Determinísticos.** Os tempos determinísticos indicam tempos absolutos relativos à execução dos eventos correspondentes.

- **Intervalares.** Os tempos intervalares usam intervalos para descrever os tempos máximos e mínimos para a duração das atividades.
- **Estocásticos.** Neste modelo, cada *delay* é descrito por uma distribuição de probabilidade.

Com adição de tempo as transições das RdP, surgem novos problemas com relação às regras de disparo das transições. Embora a transição que tem o menor tempo dispare primeiro, o problema está em decidir o que será feito com as transições que forem desabilitadas e com as não envolvidas no conflito. Para a resolução desse tipo de problema, três tipos de abordagens podem ser adotados [vdAvHR00]:

- **Resampling.** Após cada disparo, os *timers* de todas as transições são re-iniciados. Nesse caso, não há a necessidade de memória.
- **Enabling Memory.** Após cada disparo, os *timers* das transições que ficaram desabilitadas são re-iniciados.
- **Age Memory.** Após cada disparo, os *timers* de todas as transições mantêm seus valores presentes.

Um novo conceito associado às redes de Petri com tempo é o de grau de habilitação. Esse conceito determina o número de vezes que uma determinada transição pode ser disparada numa determinada marcação, antes de se tornar desabilitada. As semânticas de temporização indicam quantos disparos podem ser feitos por unidade de tempo numa transição, como mostra a seguir:

- **Single-Serve.** Apenas um *token* é disparado por vez, ou seja, a capacidade de um lugar/transição é 1.
- **Multiple-server.** É possível fazer k disparos por vez, ou seja, a capacidade de um lugar/transição é um k inteiro.
- **Infinite-Server.** É possível fazer infinitos disparos de uma única vez.

2.5.9 Redes de Petri Temporizadas com Restrição de Energia

As redes de Petri temporizadas foram inicialmente propostas por Merlin e Faber em [MF76]. No entanto, este trabalho adota o modelo no qual a informação temporal é representada por um retardo [TMSO08].

Definição 2.12. (Rede de Petri Temporizada) Uma rede de Petri temporizada (TPN) [MF76] é representado por uma tupla $\mathcal{P}= (N, I)$, onde N é a rede de Petri base e $I : T \rightarrow \mathbb{N} \times \mathbb{N}$ representa as restrições de tempo, onde $I(t) = (EFT(t), LFT(t))\ \forall t \in T$ e $EFT(t) \leq LFT(t)$. Os limites inferior(EFT) e superior(LFT) representam os tempos máximo e mínimo de disparo de uma respectiva transição. EFT e LFT denotam *Earliest Firing Time* e *Latest Firing Time*, respectivamente.

Definição 2.13. (Redes de Petri Temporizadas com Restrição de Energia) Uma Redes de Petri Temporizadas com restrição de Energia (ETPN) é representada por $\mathcal{P}_{\mathcal{E}} = (\mathcal{P}, \mathcal{E})$. $\mathcal{P}$ representa a rede de Petri temporizada, e $\mathcal{E}$:$T \rightarrow \mathbb{R}_+ \cup \{0\}$ é uma função que associa valores de consumo de energia as transições.

Existem dois modos de disparo nas ETPN: *strong* e *weakest*. No modo de disparo *strong*, uma transição habilitada (ver Definição 2.14) não pode ser disparada antes do seu EFT(t), é deve ser disparada antes ou no seu $LFT(t)$. Por outro lado, o modo de habilitação *weakest* não força o disparo da transição, ou seja, uma transição habilitada pode ou não disparar. Esta dissertação adotada o modo de disparo *strong*.

Definição 2.14. (Transições Habilitadas) Seja P uma rede de Petri temporizada, e m_i uma marcação alcançável. O conjunto de transições habilitadas para a marcação m_i é denotada por:

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

Nas redes de Petri, uma transição $t \in T$ é habilitada, se cada lugar de entrada $p \in P$ contém no mínimo $w(p, t)$ *tokens*. No entanto, nas redes de Petri temporizadas, para que a transição seja disparada, além da marcação, as restrições tempo precisam ser respeitadas. O tempo decorrido, uma vez que a transição esteja habilitada, é representado por um vetor de *clock* $c \in (\mathbb{N} \cup \{\#\})^{|T|}$, onde # representa o valor indefinido para as transições. O vetor de *clock*, por exemplo, da rede da Figura 2.24 (a) contém um elemento: $c(t_1) = 0$. Esta dissertação adotada o mecanismo de *enabling memory* para as atualizações dos *clocks*. Além disso, a semântica de disparo adotada é a *single-serve*, ou seja, apenas um *token* é disparado por vez.

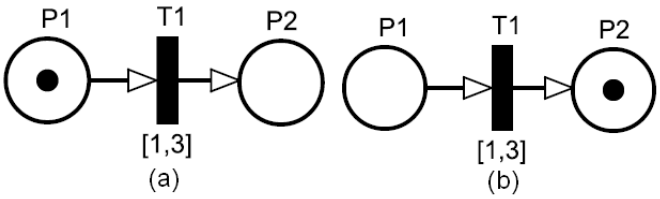


Figura 2.24: Exemplo de rede de Petri temporizada.

Nesse momento, é importante ressaltar a diferença entre disparo de intervalo dinâmico e estático. O intervalo dinâmico de disparo da transição t, $I_D(t) = [DLB(t), DUB(t)]$, é dinamicamente modificada sempre que a variável $c(t)$ do respectivo clock é incrementada, e t não dispara. $DLB(t)$ é o *Dynamic Lower Bound*, e $DUB(t)$ é o *Dynamic Upper Bound*. O intervalo dinâmico de disparo é computado da seguinte forma: $I_D(t) = [DLB(t), DUB(t)]$, onde $DLB(t) = max(0, EFT(t) - c(t))$, $DUB(t) = LFT(t) - c(t)$. Sempre que $DLB(t) = 0$, t pode disparar, no entando, quando $DUB(t) = 0$, t deve disparar, devido ao disparo *strong* adotado. Na Figura 2.24 (a), assumindo que $c(t_1)$ é incrementado em uma unidade de tempo ($c(t_1) = 1$), $I_D(t_1) = [max(0, 1-1), 3-1] = [0, 2]$.

Definição 2.15. (Estado) Sejam $\mathcal{P}_\mathcal{E}$ uma ETPN, $M \subseteq \mathbb{N}$ o conjunto de marcação de $\mathcal{P}_\mathcal{E}$, $C \subseteq (\mathbb{N} \cup \{\#\})^{|T|}$ o conjunto de vetores de clock, e $E \subseteq \mathbb{R}_+ \cup \{0\}$ o conjunto de energia acumulada. O conjunto de estados S de $\mathcal{P}_\mathcal{E}$ é dado por $S \subseteq (M \times C \times E)$.

Definição 2.16. (Domínio de disparo) Seja $s = (m, c, e)$ um estado de uma ETPN. O domínio de disparo para uma transição t de um estado especifico s, é definido pelo seguinte intervalo de tempo:

$$FD_s(t) = [DLB(t),\, \min(DUB(t_k))], \forall t_k \in ET(m).$$

A transição t do estado s só poderá ser disparada dentro do intervalo expresso por $FD_s(t)$. Considerando o modelo ETPN da Figura 2.24 (a), o estado inicial $s_0 = (m_0 = [1,0], c_0 = [0], e_0 = 0)$, t_1 é disparavel quando $c(t_1) = 1$ e deve disparar quando $c(t_1) = 3$ ($FD_s(t) = [1,3]$).

Para um melhor entendimento das rede de Petri temporizadas um exemplo é apresentado (ver Figura 2.25 (a)), onde:

- $P = \{p_0, p_1, p_2, p_3, p_4\}$;
- $T = \{t_0, t_1, t_2, t_3, t_4\}$;
- $F = \{(p_0,t_0), (t_0,p_1), (t_0,p_2), (p_1,t_1), (p_1,t_2), (p_2,t_3), (t_1,p_3), (t_2,p_3), (t_3,p_3), (p_3,t_4), (t_4,p_4)\}$;
- $W(f) = \{(p_0,t_0,1), (t_0,p_1,1), (t_0,p_2,1), (p_1,t_1,1), (p_1,t_2,1), (p_2,t_3,1), (t_1,p_3,1), (t_2,p_3,1), (t_3,p_3,1), (p_3,t_4,2), (t_4,p_4,1)\}$;
- $m_0(p) = \begin{cases} 1, & \text{se } p = p_0; \\ 0, & \text{caso contrário.} \end{cases}$
- $I(t) = \{(t_0,0,0), (t_1,1,3), (t_2,2,4), (t_3,2,3), (t_4,0,0)\}$;

O estado inicial é representado por $s_0 = ([1,0,0,0,0],\ \overset{t0}{[\,0,} \ \overset{t1}{\#,} \ \overset{t2}{\#,} \ \overset{t3}{\#,} \ \overset{t4}{\#\,]}^T)$, onde t_0 é a única transição habilitada ($ET(m_0 = \{t_0\})$) para a marcação m_0. Devido ao modo de disparo *strong*, a transição t_0 deve disparar, visto que $I_D(t_0) = [0,0]$. A Figura 2.25 (b) apresenta o novo estado $s_1 = (|0,1,1,0,0|,\ \overset{t0}{[\,\#,} \ \overset{t1}{0,} \ \overset{t2}{0,} \ \overset{t3}{0,} \ \overset{t4}{\#\,]}^T)$ alcançado pelo disparo de t_0 no instante $\theta = 0$. Os intervalos dinâmico de disparo das transições habilitadas são $I_D(t_1) = [1,3]$, $I_D(t_2) = [2,4]$ e $I_D(t_3) = [2,3]$. Além disso, o domínio de disparo das transições são $FD_s(t_1) = |1,3|$, $FD_s(t_2) = |2,3|$ e $FD_s(t_3) = |2,3|$. Considerando um incremento em duas unidades de tempo ($\theta = 2$) sem o disparo de nenhuma transição, os novos valores dos intervalos dinâmico são $I_D(t_1) = [max(0, 1-2), 3-2] = [0,1]$, $I_D(t_2) = [max(0, 2-2), 4-2] = [0,2]$, e $I_D(t_3) = [max(0, 2-2), 3-2] =$

$[0, 1]$. Nesse momento, assumindo que a transição t_2 é disparada, uma nova marcação $s_5 = ([0, 0, 1, 1, 0], \overset{t0}{[\ \#}, \overset{t1}{\#}, \overset{t2}{\#}, \overset{t3}{2}, \overset{t4}{\#\ ]}^T)$ é alcançada (ver Figura 2.25 (c)). Após isso, considerando o disparo de t_3 no instate $\theta = 1$ no estado s_5, o estado $s_3 = ([0, 0, 0, 2, 0], \overset{t0}{[\ \#}, \overset{t1}{\#}, \overset{t2}{\#}, \overset{t3}{\#}, \overset{t4}{0\ ]}^T)$ é obtido (ver Figura 2.25 (d)). Por fim, o estado $s_5 = ([0, 0, 0, 0, 1], \overset{t0}{[\ \#}, \overset{t1}{\#}, \overset{t2}{\#}, \overset{t3}{\#}, \overset{t4}{\#\ ]}^T)$ (ver Figura 2.25 (e)) é alcançado devido o disparo de t_4 no instante $\theta = 0$.

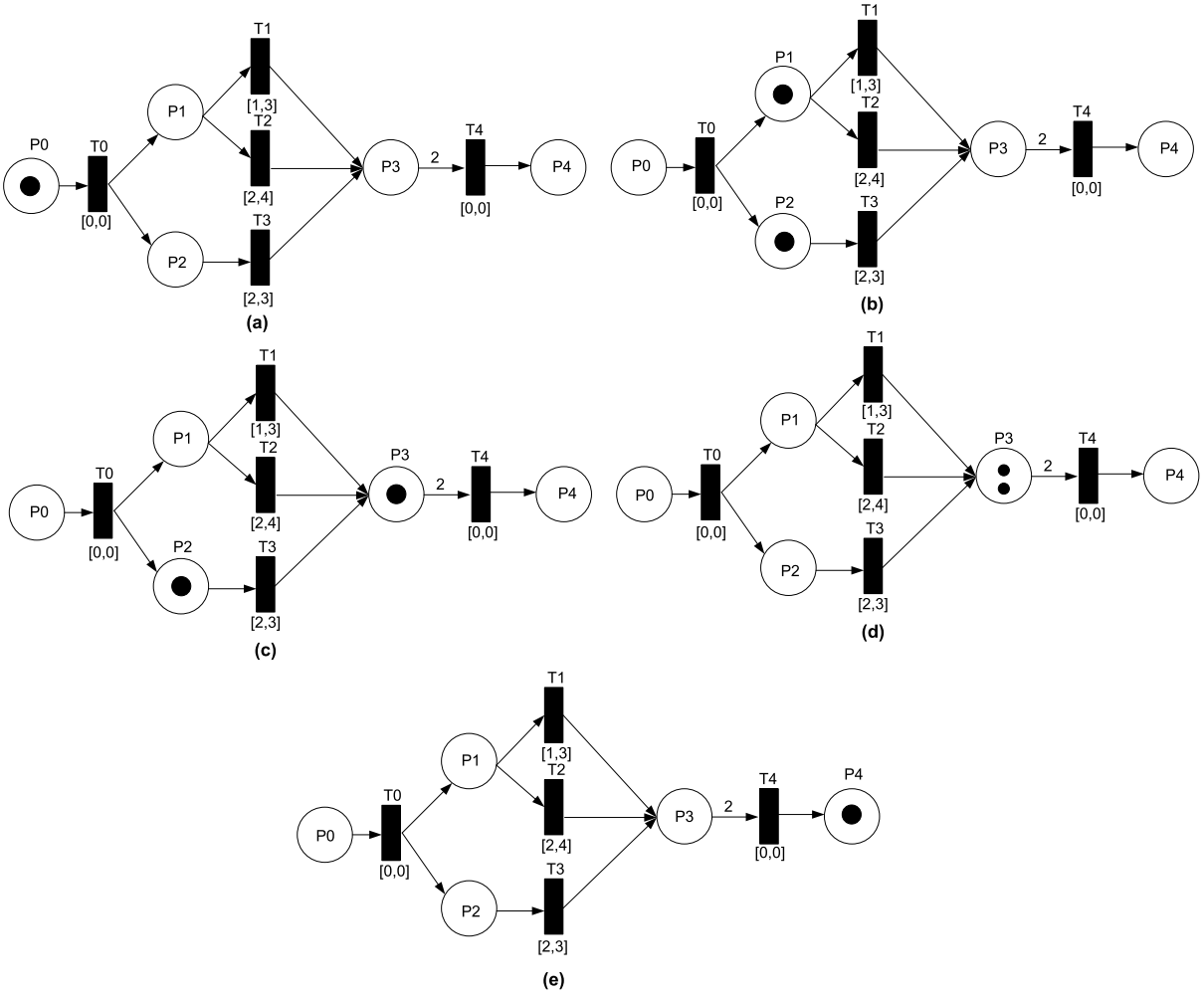


Figura 2.25: Exemplo de rede de Petri temporizada.

E importante destacar que outros estados podem ser alcançados pelo disparo das transições em outros instantes de tempo. Por exemplo, no estado s_1 (ver Figura 2.26), t_1 pode ser disparada nos tempo de 1, 2 e 3, alcançando os estados s_2, s_5 e s_6, respectivamente. A Figura 2.26 apresenta o grafo de alcançabilidade do exemplo.

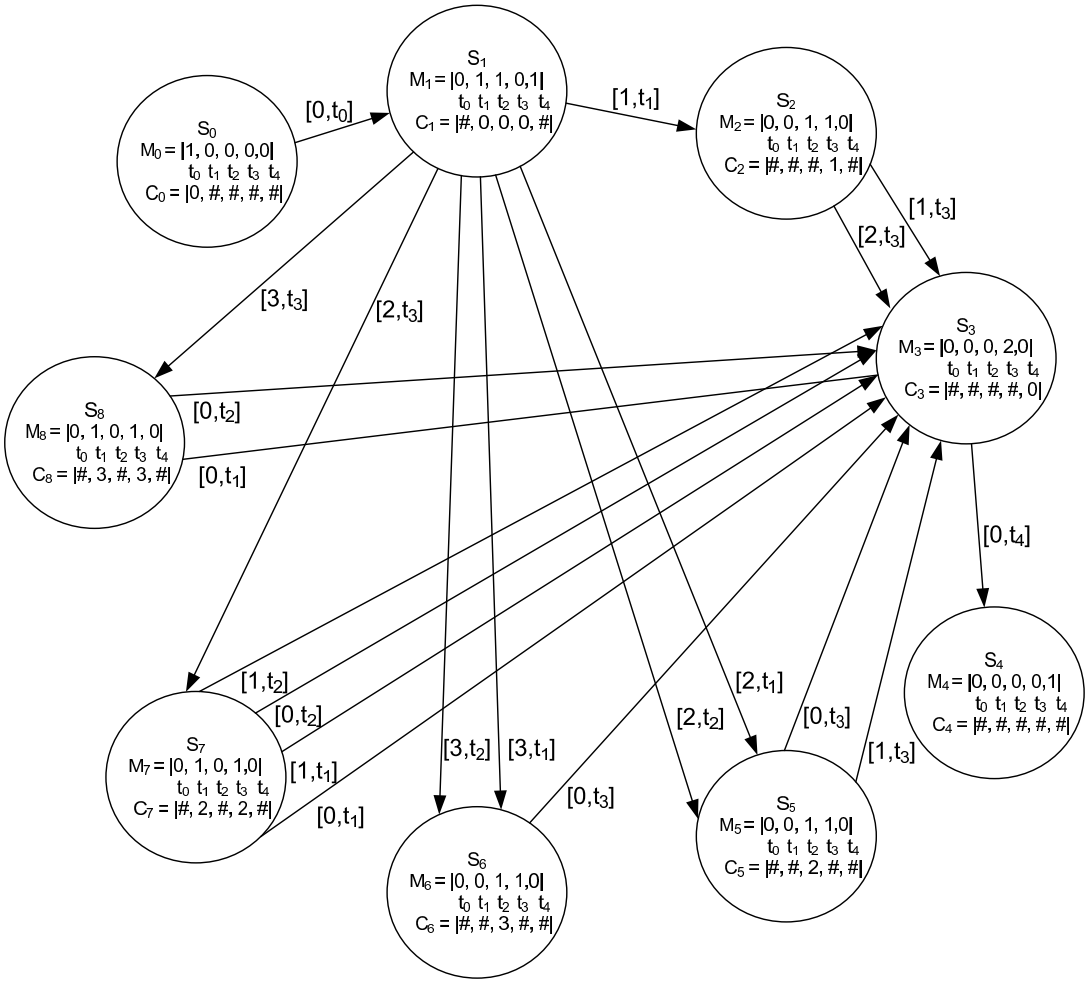


Figura 2.26: Grafo de alcançabilidade da Figura 2.25.

2.6 CONSIDERAÇÕES FINAIS

Este capítulo apresentou os principais conceitos que envolvem esta dissertação. Primeiramente, os sistemas embarcados foram apresentados, destacando suas principais características, assim como suas principais restrições. Subseqüentemente, a SysML foi introduzida, apresentado os diagramas (Estados, Atividades e Seqüencia) que são usados no processo de mapeamento em uma rede de Petri temporizadas com restrições de energia. Após isso, MARTE foi apresentado, detalhando as anotações que são usadas neste trabalho. Por fim, as redes de Petri também foram introduzidas, mostrando que elas são uma ferramenta bem estabelecida para modelagem e análise de vários tipos de sistemas, tais como sistemas concorrentes, assíncronos, distribuídos, paralelos, não-determinísticos e estocásticos.

CAPÍTULO 3

METODOLOGIA DE AVALIAÇÃO DE DESEMPENHO

Este capítulo apresenta uma metodologia para avaliação de desempenho das especificações de sistemas críticos. A abordagem consiste, primeiramente, no projeto de requisitos. Subseqüentemente, os diagramas comportamentais da SysML são criados, incluindo as restrições tempo e anotações de energia. Após isso, os elementos básico dos diagramas são mapeados em modelos ETPN individuais. Por fim, todos os modelos ETPN são compostos em um único modelo que representa o comportamento por completo de cada diagrama, e assim, as análises e verificações podem ser realizadas nesses modelos gerados.

3.1 INTRODUÇÃO

Este trabalho está inserido no contexto de projetos de sistemas embarcados de tempo-real críticos, em particular, os que têm restrições quanto ao consumo de energia. Como já apresentado, é de particular interesse deste trabalho métricas relacionadas tanto ao consumo de energia quanto ao tempo de execução. Para tanto é proposto o mapeamento dos modelos semiformais das especificações dos sistemas críticos em modelos de desempenho. Esses modelos de desempenho são baseados em redes de Petri temporizadas, com anotações de consumo de energia, chamadas de ETPN.

É importante salientar que os resultados obtidos pela metodologia proposta são adquiridos ainda durante as fases iniciais do sistema, conseqüentemente, são de extrema importância, pois possibilitam uma exploração rápida do espaço de projeto pela avaliação de diferentes alternativas de implementações, bem como a verificação de violações de restrições. Isso tudo pode ser realizado sem a necessidade da real implementação do sistema, tornando mais ágil e barato todo o processo de desenvolvimento.

A seguir, na Figura 3.1, é apresentado a metodologia MEMBROS (*Methodology for embedded Critical Software Construction*). Essa metodologia contextualiza o ambiente no qual este trabalho está inserido, destacando suas principais atividades bem como a integração com outros trabalhos. MEMBROS organiza suas atividades basicamente em três módulos: (i) análise de requisitos, (ii) avaliação de desempenho e consumo de energia, e (iii) síntese de software. É importante destacar que esta dissertação objetiva as atividades relativas à análise de requisitos. A seguir, uma visão geral da metodologia é apresentada.

Inicialmente, as atividades relativas à análise de requisitos são executadas. A primeira atividade a ser levada em consideração nesse conjunto de atividades é o projeto de requisitos do ERTS. Uma vez realizada essa atividade, então os requisitos são modelados

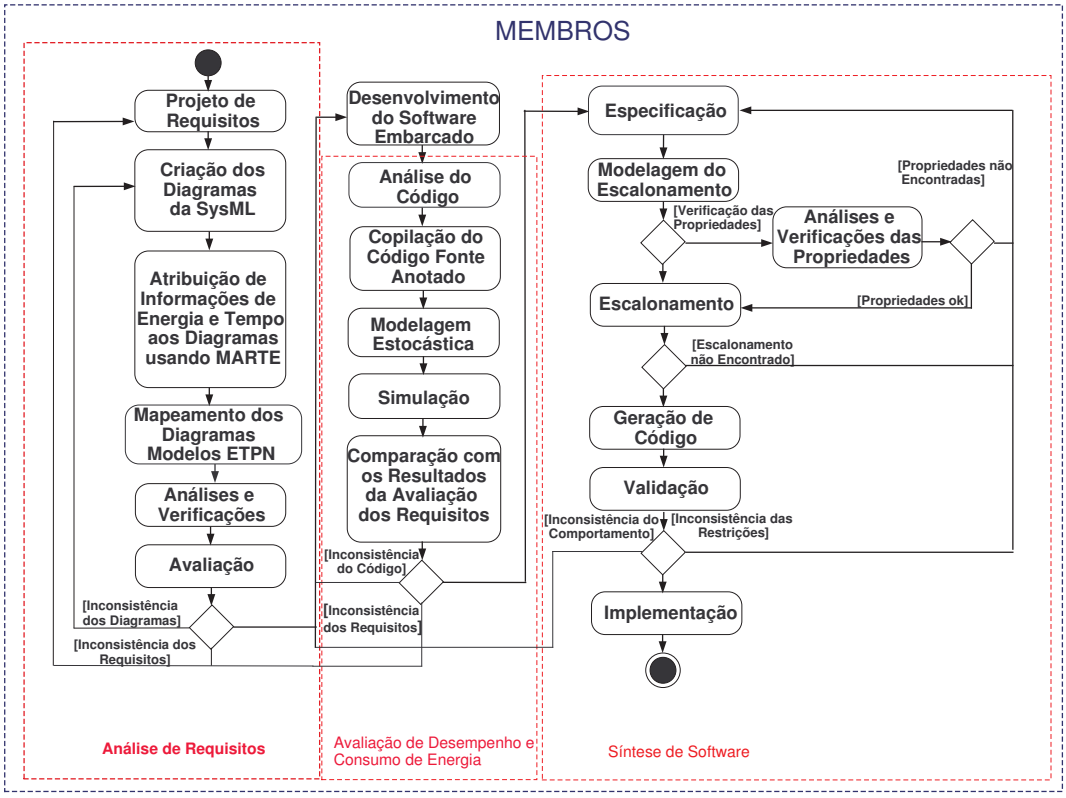


Figura 3.1: Metodologia MEMBROS.

através dos DCS. É importante ressaltar que esses diagramas fornecem ao projetista uma linguagem amigável e intuitiva para a modelagem dos sistemas críticos, sem a necessidade do entendimento do formalismo das RdP. Após a modelagem, às anotações de tempo e energia são atribuídas aos DCS utilizando MARTE. Subseqüentemente, os DCS anotados são mapeados em modelos ETPN a fim de realizar análises e verificações. O próximo passo da metodologia consiste na avaliação dos resultados obtidos dos modelos ETPN. Se esses resultados estão de acordo com o especificado nos requisitos, então as atividades relacionadas à análise de requisitos são finalizadas. Caso os resultados não estejam em conformidade com o especificado, a metodologia proposta volta para os primeiros passos, podendo realizar ajustes tanto nos requisitos quanto nos DCS. Uma vez realizados todos os ajustes necessários, então os passos subseqüentes da metodologia são executados novamente com o objetivo de verificar a conformidade com os requisitos. Essas atividades do módulo de análise de requisitos são detalhadas nas seções seguintes. Adicionalmente, as atividades desse módulo podem ser encontradas nas seguintes publicações: [AMC+08, AMCNb, AMCNc, AMCNa].

Subseqüentemente, as atividades relacionadas ao módulo de avaliação de desempenho são realizadas. Primeiramente o software embarcado é implementado levando em consideração os resultados obtidos nas atividades anteriores. Uma vez que a implementação do código fonte esteja concluída, o projetista analisa o código de modo a atribuir valores de

probabilidade para as estruturas iterativas e condicionais. As anotações de probabilidade permitem que o código compilado seja avaliado no contexto do tempo e do consumo de energia, de modo que esses custos possam ser estimados antes da execução do código sobre a plataforma de *hardware*. Após isso, o código compilado é automaticamente convertido em um modelo *Coloured Petri Net* (CPN) [Jen91]. Adicionalmente, as CPNs são uma extensão das rede Petri de alto nível, cujo objetivo nesta abordagem é fornecer uma base para a simulação estocástica do software embarcado. Embora não esteja representada na Figura 3.1, uma atividade de caracterização de arquitetura também é considerada para possibilitar a construção de uma biblioteca de blocos básicos da CPN, pois esses blocos fornecem toda a base para a geração automática dos modelos estocásticos das CPNs. A partir do modelo CPN (gerado pela composição dos blocos básicos), a simulação estocástica do código compilado é realizada considerando as características da plataforma alvo. Por fim, se os resultados da simulação estão de acordo com os desejados, então a síntese de *software* é realizada. Mais informações sobre as atividades do módulo de avaliação de desempenho e consumo de energia podem ser encontradas em [dOJ06, JML+05, JMBC04, CMA+08a, CMA+08c, CMA+08b].

Por fim, as atividades relacionadas ao módulo de síntese de *software* são realizadas. Em resumo, síntese de *software* é a tarefa de traduzir uma especificação de alto nível em código fonte. Em geral, a síntese de *software* é composta por dois subgrupos de atividades: (i) tratamento das tarefas, e (ii) geração de código. Tratamento das tarefas é responsável pelo escalonamento de tarefas, gerenciamento de recursos e comunicação entre tarefas. Por outro lado, a geração de código trata da geração estática de código fonte para cada tarefa.

A primeira atividade a ser realizada na síntese de *software* é a medição. Essa atividade almeja obter informações dos tempos das tarefas, assim como informações relativas ao consumo de energia do *hardware*. Em seguida, o projetista define a especificação das restrições do sistema, que consiste na especificação das restrições temporais e de energia de cada tarefa, além de descrever o comportamento de cada uma delas. Mais especificamente, o modelo de especificação considera: restrições de tempo das tarefas bem como seu código fonte, relações entre tarefas, informações relacionadas à plataforma de *hardware* e restrições de energia do sistema. Subseqüentemente, a especificação é traduzida em um modelo capaz de representar atividades concorrentes, informação de tempo, relações entre tarefas, tais como precedência e exclusão mútua, como também restrições de energia. Adicionalmente, o modelo adotado é uma extensão das rede Petri temporizadas, anotada com valores de consumo de energia assim como anotações de código (TPNE). Após gerar o modelo TPNE, o projetista pode tanto executar as análises/verificações das propriedades quanto realizar a atividade de escalonamento. A etapa de Checagem das propriedades é adotada para verificação e análise de propriedades no modelo de Redes de Petri obtido na etapa de modelagem. Basicamente a checagem das propriedades garante que o algoritmo de escalonamento sempre vai terminar, encontrando uma escala viável ou não. Este módulo adota uma abordagem de escalonamento de *pre-runtime* a fim de descobrir um escalonamento exeqüível que satisfaça as restrições de tempo energia, bem como as relações entre tarefas. Após isso, o escalonamento exeqüível é adotado como uma entrada

para o mecanismo de geração automática de código, de tal forma que o código resultante contém apenas os serviços estritamente necessários para execução da aplicação, ou seja, a etapa de geração de código tem como resultado o código do despachante customizado para a execução das tarefas de tempo-real crítico. Por fim, a aplicação é validada numa plataforma compatível com DVS (*Dynamic Voltage Scaling*) a fim de checar o comportamento do sistema. Estando o comportamento correto, o sistema pode ser preparado para a fase de implantação, caso contrário a especificação é ajustada e as etapas seguem o fluxo da metodologia descrito na Figura 3.1. Uma vez validado o sistema, ele pode ser implantado para utilização no ambiente real. Mais informações relativas à síntese de *software* podem ser encontradas em [dSB05, Tav06, TMB+05, TBM+07]

Em seguida, como apresentado na Figura 3.2, o módulo de análise de requisitos é detalhado. Esse módulo é composto por diversos passos que vão desde o projeto de requisitos, passando pelo mapeamento dos diagramas comportamentais da SysML, até a análise dos resultados obtidos.

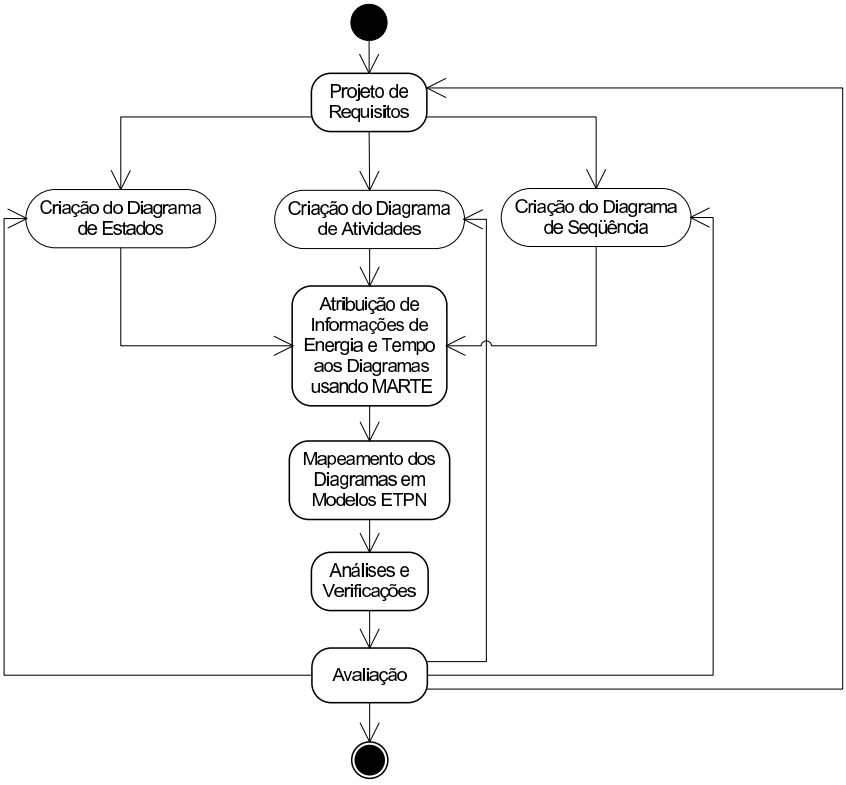


Figura 3.2: Módulo de análise de requisitos.

3.2 PROJETO DE REQUISITOS

A primeira etapa da metodologia é o projeto de requisitos. Essa etapa, engloba as tarefas que lidam com a investigação e definição dos requisitos de novos sistemas. Mais especificamente, essa etapa inclui três tipos de atividades:

- **Elicitação dos requisitos.** É a tarefa de comunicar-se com os usuários e clientes pra determinar quais são os requisitos do sistema. Essa tarefa procura identificar os fatos que compõem os requisitos do sistema, incluindo as restrições do sistema, de forma a prover um correto e completo entendimento do que é demandado daquele sistema. Entre as diversas técnicas de elicitação dos requisitos, é possível destacar as seguintes: entrevista, leitura de documentos, questionários, análise de protocolos, participação ativa dos usuários, cenários, sessões *brainstorming*, observações e análise sociais e reuso de requisitos.

- **Especificação dos requisitos.** Os requisitos, na maioria dos casos, são documentados através de uma linguagem natural, descrevendo exatamente o que deve ser feito, assim como o que se espera receber como resultado. Essa etapa é muito importante na análise de requisito, pois requisitos mal especificados produzem retrabalho, custo e atrasos no projeto.

- **Verificação dos requisitos.** Consiste em verificar o estado (obscuro, incompleto, ambíguo, ou contraditório) dos requisitos, ou seja, verificar se o requisito contém erros ou está mal escrito. Caso contenha algum desses problemas, estes devem ser corrigidos.

A etapa de projeto de requisitos é uma tarefa crucial para os sistemas embarcados críticos, pois erros, em alguns casos, podem levar a resultados catastróficos, tais como o caso do Therac-25, que aplicava doses radioativas letais nos pacientes [LT93], ou caríssimo, como o do Ariane 5, cujas falhas de lançamento custaram milhões ao programa espacial europeu [JM97]. Dentre as principais dificuldade dessa etapa, é possível destacar as seguintes: (i) usuários podem não ter uma idéia precisa do sistema por eles requerido, (ii) usuários têm dificuldades de descrever seu conhecimento sobre o domínio do problema, (iii) usuários e analistas têm diferentes pontos de vista do problema (por terem diferentes formações) e (iv) usuários podem antipatizar-se com o novo sistema e se negarem a participar da elicitação.

Os requisitos, de modo geral, podem ser classificados em dois grandes grupos: os requisitos funcionais e os não funcionais. O requisitos funcionais são aqueles que descrevem o comportamento do sistema, suas ações para cada entrada, ou seja, é aquilo que descreve o que tem que ser feito pelo sistema. Por outro lado, os requisitos não funcionais são aqueles que expressam como deve ser feito. Em geral se relacionam com padrões de qualidade como confiabilidade, desempenho, robustez, entre outros. São muito importantes, pois definem se o sistema será eficiente para a tarefa que se propõe a fazer ou não. Um sistema ineficiente sem sombra de dúvida não será usado.

É importante observar que embora este trabalho foque de forma mais expressiva na análise dos requisitos não funcionais (tempo de execução e consumo de energia), ele também foca nos requisitos funcionais, através das análises qualitativas obtida dos modelos gerados pelo processo de mapeamento, tais como: *deadlock*, *repetitividade* e *reversibilidade*, entre outros.

As informações produzidas nessa etapa são extremamente importantes, pois a maior parte dos problemas, os de maior impacto negativo e os mais onerosos têm origem nas etapas iniciais do desenvolvimento de sistemas [DRP99a]. Justamente nas etapas de especificação dos requisitos é onde as principais atividades são definidas e onde os requisitos do produto devem ser identificados e mapeados com objetividade e clareza.

3.3 CRIAÇÃO DOS DIAGRAMAS COMPORTAMENTAIS DA SYSML

A etapa seguinte da metodologia envolve a criação dos diagramas comportamentais da SysML, sendo que esses são usados para visualizar, especificar, construir e documentar os aspectos dinâmicos de um sistema que é a representação das partes que sofrem alterações, por exemplo, o fluxo de mensagens ao longo do tempo e a movimentação física de componentes em uma rede.

Após a elicitação dos requisitos e a elaboração da especificação do sistema, os requisitos são detalhados a fim de facilitar a análise e o projeto dos sistemas. Esse detalhamento é realizado através de modelos semiformais. Neste trabalho, como já mencionado anteriormente, a modelagem é realizada através dos diagramas comportamentais da SysML.

A construção dos diagramas da SysML, não é uma tarefa trivial, pois requer um considerável conhecimento do sistema a ser modelado, bem como boa experiência do projetista na área de modelagem para facilitar a identificação (estados, transições, mensagens etc) do que realmente precisa ser modelado. Este trabalho, contudo, apresenta um conjunto de passos a serem seguidos para a construção desses diagramas. Para o diagrama de estados, basicamente, os passos adotados foram:

1. primeiramente, os estados relevantes para o cenário a ser modelado são identificados. Para cada estado, os eventos internos (*entry*, *do*, *exit* e transições internas) e as ações também são identificadas;
2. após isso, os eventos são identificados. Cada evento, é representado por uma transição. Essas transições podem ter expressões de guarda se há fatores que influenciam a ocorrência do evento;
3. caso haja expressões de guarda, os seus atributos são identificados;
4. subseqüentemente, os elementos encontrados são analisados a fim de encontrar estados compostos seqüencias ou concorrentes;
5. por fim, os estados inicial e final tanto para o diagrama de estados, quanto para os estados compostos são identificados, e então os elementos encontrados são compostos em um único modelo que representa o DE do cenário a ser modelado.

De forma semelhante, foram adotados um conjunto de passos para a construção do diagrama de atividades. Os passos, basicamente, são os seguintes:

1. primeiramente, as atividades relevantes para o cenário a ser modelado são identificadas;

2. após isso, os eventos são identificados. Cada evento, é representado por uma transição. Essas transições podem ter expressões de guarda se há fatores que influenciam a ocorrência do evento;

3. caso haja expressões de guarda, os seus atributos são identificados;

4. subseqüentemente, os elementos encontrados são analisados a fim de encontrar sincronização ou bifurcação das atividades;

5. por fim, os estados inicial e final são identificados, e então os elementos encontrados são compostos em um único modelo que representa o DA do cenário a ser modelado.

Por fim, um conjunto de passos também foram adotados para construir o diagrama de seqüencia. Os passos, basicamente, foram os seguintes:

1. primeiramente, os particiapante relevantes para o cenário a ser modelado são identificados. Para cada particiapante, a linha de vida bem como seu foco de ativação são identificados;

2. após isso, os eventos são identificados. Cada evento, é representado por uma mensagem ou chamada. Essas mensagens ou chamadas podem ter expressões de guarda se há fatores que influenciam a ocorrência do evento;

3. caso haja expressões de guarda, os seus atributos são identificados;

4. subseqüentemente, os elementos encontrados são analisados a fim de encontrar combinações entre as mensagens ou chamadas.

5. por fim, os elementos encontrados são compostos em um único modelo que representa o DS do cenário a ser modelado.

Não necessariamente todos os diagramas tratados nessa etapa da metodologia precisaram ser construídos para um determinado cenário, entretanto, todos os diagramas (estados, atividades e seqüência) são fundamentais para que sejam tomadas as melhores decisões, pois esses diagramas possuem características diferentes, conseqüentemente, revelam diferentes aspectos da dinâmica do sistema. Os diagramas comportamentais são detalhados no Capítulo 4

3.4 ATRIBUIÇÃO DE INFORMAÇÕES DE ENERGIA E TEMPO AOS DIAGRAMAS USANDO MARTE

Até este ponto da metodologia, as restrições dos sistemas embarcados de tempo-real não foram especificadas. No entanto, durante a construção desses sistemas é indispensável a descrição dos aspectos quantitativos [TZ05]. Para isso, um novo profile da UML para Modelagem e Análise dos Sistemas Embarcados de Tempo-Real (MARTE) foi utilizado.

MARTE é usado para especificar as restrições dos diagramas comportamentais gerados na etapa anterior. Através dessa especificação, será possível, nas fases seguintes da metodologia, realizar estimativas (consumo de energia e tempo de execução) ainda durante fases iniciais do ciclo de desenvolvimento do sistema, sem a necessidade da implementação real para a obtenção dessas medidas. Em projetos de sistemas embarcados, faz-se necessária a previsão (consumo de energia e tempo de execução) ainda nas fases iniciais do projeto, pois possibilita a tomada de decisões rápidas a fim de encontrar uma das possíveis soluções que atenda as restrições de projeto.

Este trabalho, como já mencionado anteriormente, possui como um dos seus objetivos prover meios para realizar estimativas do tempo de execução e consumo de energia dos ETRS. Dentre os diversos estereótipos disponíveis por MARTE, o estereótipo *ResourceUsage* é usado para representar tanto o tempo de execução quanto o consumo de energia. A Tabela 3.1 apresenta os valores marcados, usados nesta dissertação, do estereótipo *ResourceUsage*.

Tabela 3.1: Valores marcados usados para representar as restrições dos ERTS.

Estereótipo	Valores Marcados
ResourceUsage	execTime
	energy

As anotações de MARTE podem ser especificadas aos diversos elementos dos diagramas comportamentais da SysML. Por exemplo: especificar um tempo de execução a uma atividade do DA, especificar um consumo de energia às transições de estados do DE ou especificar um tempo de execução ao disparado de uma chamada/mensagem entre duas entidades no DS, entre outras.

Para que se tenha sistemas robustos e de fácil manutenção, entre várias outras características desejáveis, é imprescindível que haja um planejamento cuidadoso do sistema como um todo. Nesse planejamento, devem ser inclusos as restrições impostas ao sistema a ser implementado, pois os sistemas embarcados de tempo-real críticos com restrições de consumo de energia possuem, além dos requisitos funcionais, requisitos não funcionais (tempo e energia) que devem ser atendidos para o correto funcionamento.

3.5 MAPEAMENTO DOS DIAGRAMAS COMPORTAMENTAIS DA SYSML EM MODELOS ETPN

Os modelos semiformais com restrições de tempo e anotações de energia obtidos na etapa anterior, por si só, não fornecem suporte para avaliação de desempenho das especificações dos sistemas, assim, se faz necessário o mapeamento desses modelos semiformais para modelos formais. Modelos formais são convenientes, pois podem refletir as diversas características do sistema a ser modelado, dentre elas, aspectos comportamentais, temporais e de consumo de energia. Além disso, o uso de modelos formais proporcionam a detecção de erros ou inconsistências dos requisitos e, conseqüentemente, podem ser utilizados como um meio para a redução tanto da incidência de erros de interpretação quanto de implementação.

Portanto, como já é de conhecimento prévio, a detecção precoce dos erros no ciclo de vida do desenvolvimento dos ERTS é muito importante. Segundo [Gro95] a metade dos fracassos dos projetos são devidos a erros nos requisitos, assim a integração dos modelos semiformais com os modelos formais promove a eficiência, *corretude* e validação dos requisitos ainda nas fases iniciais do desenvolvimento dos ERTS.

Para possibilitar a integração dos modelos semiformais e formais, o mapeamento dos diagramas comportamentais da SysML (estados, atividades e seqüência) em redes de Petri temporizadas com restrições de energia é realizado. O método proposto consiste, primeiramente, na derivação dos elementos básicos de cada diagrama (estados, transições, *lifeline*, mensagens, chamadas, fragmentos combinados, estado inicial, estado final etc.) em modelos ETPN individuais. Após isso, esses modelos ETPN são compostos em um modelo único que representa o respectivo diagrama. As anotações quantitativas do *profile* MARTE, tais como tempo e energia, são levadas em consideração e inclusas no modelo ETPN. O Capítulo 4 apresenta em detalhes o mapeamento dos elementos dos diagramas comportamentais da SysML. Por fim, os modelos obtidos que representam o comportamento dos diagramas são adotados para realizar análises e verificações. As análises e verificações dos modelos ETPN são descritas na etapa seguinte da metodologia.

3.6 ANÁLISES E VERIFICAÇÕES

Nesta etapa, os modelos ETPN gerados pelo processo de mapeamento são adotados para realizar análises e verificações.

As TPN possibilitam o cálculo de métricas temporais. No entanto, métricas relativas ao consumo de energia não são diretamente obtidas. A fim de solucionar esse problema, foi utilizado uma rede de Petri que possibilite a captura dessas métricas, chamada de ETPN. Formalmente a ETPN é definida por uma tripla $\mathcal{P}_{\mathcal{E}} = (\mathcal{P}, \mathcal{I}, \mathcal{E})$ em que I e $\mathcal{E}$ são funções que associam tempo e energia ao disparo das transições do modelo, respectivamente.

Nesta dissertação a ferramenta INA (*Integrated Net Analyser*) [SR99] é usada para realizar as análises qualitativas e quantitativas. INA permite analisar propriedades estruturais e comportamentais das redes de Petri, bem como verificar um conjunto de

propriedades associadas as características estruturais dos modelos gerados. As análises qualitativas capturam os aspectos lógicos da evolução dos sistemas. Entre as propriedades de interesse, podemos ressaltar a análise da existência *deadlock*, *liveness* que são importantíssimas em sistemas críticos. Além disso, propriedades como *reversibilidade*, *limitação* e *conservação* são importantes em sistemas de controle. Essas propriedades são detalhas no Capítulo 2.

A análise quantitativa possui como principais objetivos a avaliação de desempenho e da utilização de recursos dos sistema. Neste trabalho, as estimativas de BPET (*Best Case Execution Time*) e WPET (*Worst Case Execution Time*) [EES+03] são adotadas para os modelos gerados pelo processo de mapeamento. Uma vez encontrados os *traces* dos tempos de execução, então para o melhor caso o ECBC *(Energy Consumption for Best Case)* é calculado e para o pior caso o ECWC *(Energy Consumption for Worst Case)* é calculado. Tabela 3.2 apresenta um resumo das métricas adotadas nesta dissertação.

Tabela 3.2: Métricas.

Métrica	Descrição
BCET	*Best Case Execution Time*
WCET	*Worst Case Execution Time*
ECBC	*Energy Consumption for Best Case*
ECWC	*Energy Consumption for Worst Case*

A fim de facilitar o entendimento das ETPN e das métricas definidas anteriormente, um pequeno exemplo fictício é utilizado (ver Figura 3.3). Cada uma das transições do modelo ETPN possui um tempo (I) e um consumo de energia ($\mathcal{E}$) associado, onde sempre o primeiro valor associado a transição do modelo é o tempo e o segundo é o consumo de energia. Como pode ser observado na figura, intervalos estreitos (*thin interval*) (intervalo cujos os valores do limite superior e inferior são iguais [Dav05]) são usados, com o intuito de diminuir o espaço de estado. A Figura 3.4 apresenta o gráfico de estados temporizado do exemplo. Além disso, a Tabela 3.3 resume os atributos das transições do modelo ETPN da Figura 3.3.

Sendo definidos os atributos da ETPN utilizada no exemplo, é possível realizar o cálculo das métricas de desempenho através das funções discutidas anteriormente. Primeiramente será calculado o BCET como mostra a Equação 3.1. O BCET calcula o menor tempo entre uma marcação inicial M_0 a uma marcação M_i. Para o exemplo em particular da Figura 3.3, o cálculo é dado pela soma dos tempos das transições percorridas da marcação $M_0 = [1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]$ até a marcação $M_7 = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1]$ (ver Figura 3.4). É importante destacar que caso houvesse paralelismo, o cálculo seria realizado de outra forma, por exemplo, pelo grafo de estados temporizado.

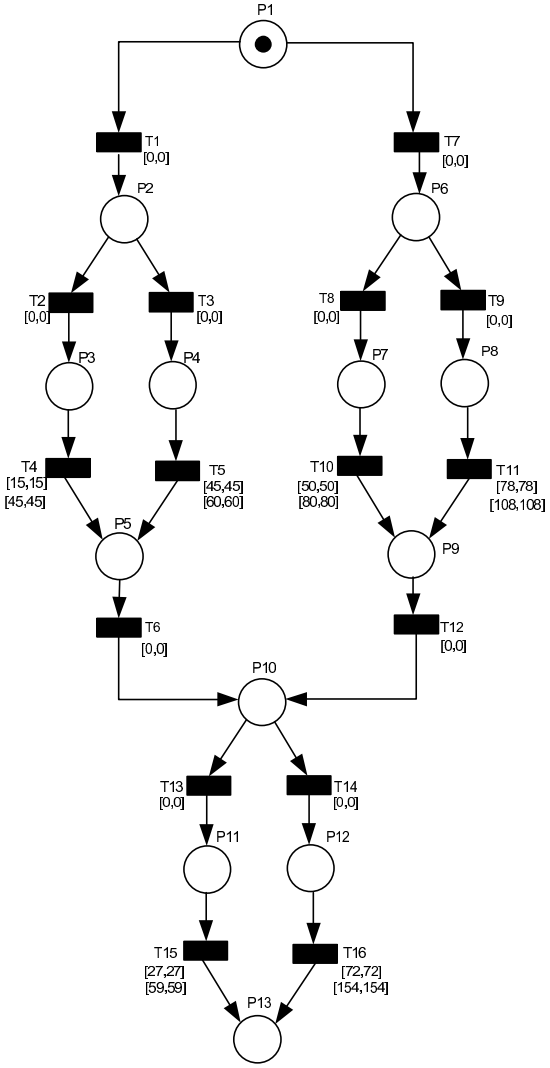


Figura 3.3: Modelo ETPN de um diagrama de estados.

$$
\begin{aligned}
BCET &= T1 + T2 + T4 + T6 + T13 + T15 \\
BCET &= 0 + 0 + 15 + 0 + 0 + 27 \\
BCET &= 42
\end{aligned}
\tag{3.1}
$$

O WCET calcula o maior tempo entre uma marcação inicial M_0 a uma marcação M_i. O cálculo é dado pela soma dos tempos das transições percorridas da marcação $M_0 = [1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]$ até a marcação $M_7 = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1]$. A Equação 3.2 mostra o cálculo do tempo de execução no pior caso pelo disparo das transições.

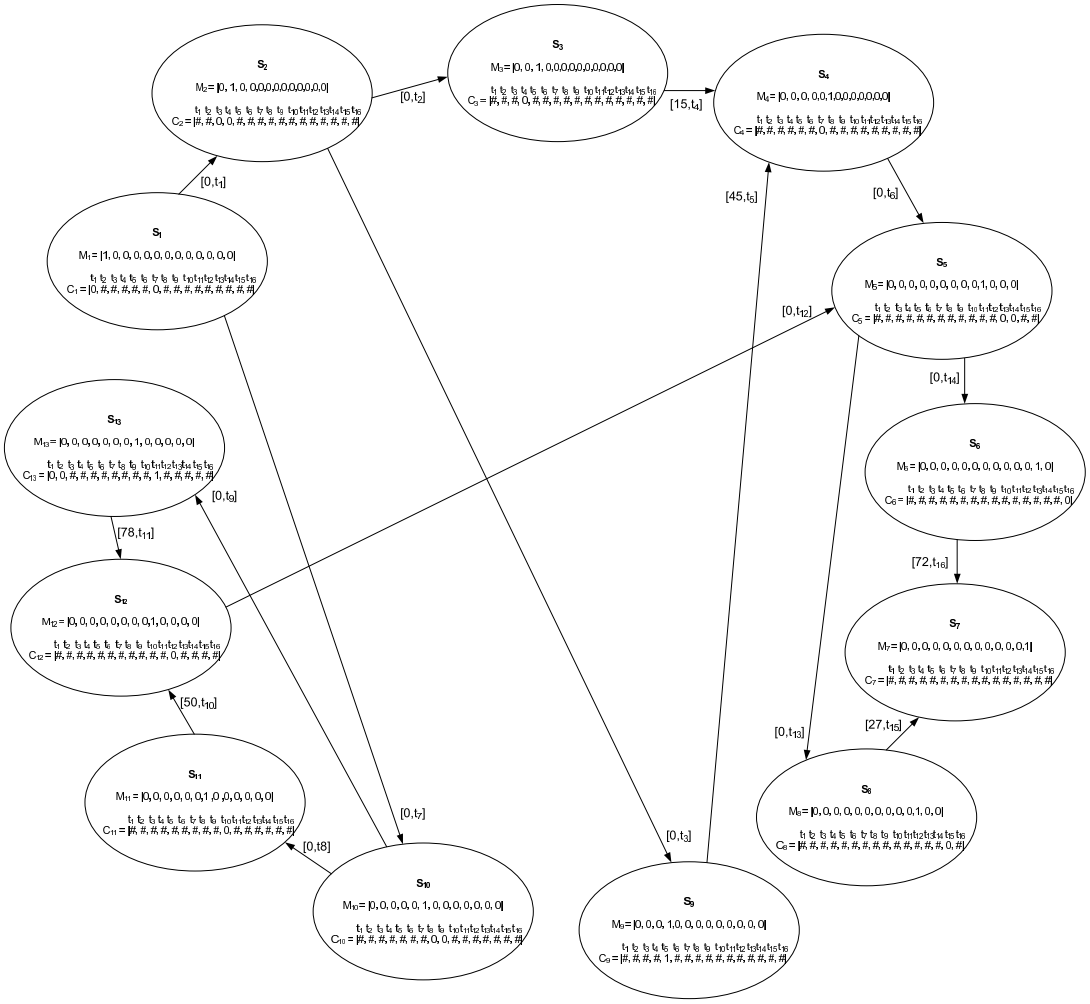


Figura 3.4: Gráfico de estados do modelo ETPN da Figura 3.3.

$$WCET = T7 + T9 + T11 + T12 + T14 + T16 \tag{3.2}$$
$$WCET = 0 + 0 + 78 + 0 + 0 + 72$$
$$WCET = 150$$

Uma vez encontrados os *traces* dos tempos de execução, então para o melhor caso o ECBC *(Energy Consumption for Best Case)* é calculado e para o pior caso o ECWC *(Energy Consumption for Worst Case)* é calculado. As Equações 3.3 e 3.4 apresentam os cálculos para o ECBC e ECWC. A Tabela 3.4 apresenta os resultados das métricas obtidas do exemplo da Figura 3.3.

Tabela 3.3: Atributos das transições do exemplo.

Transição	I	$\mathcal{E}$
T1	(0,0)	(0,0)
T2	(0,0)	(0,0)
T3	(0,0)	(0,0)
T4	(15,15)	(45,45)
T5	(0,0)	(0,0)
T6	(0,0)	(0,0)
T7	(0,0)	(0,0)
T8	(0,0)	(0,0)
T9	(0,0)	(0,0)
T10	(50,50)	(80,80)
T11	(78,78)	(108,108)
T12	(0,0)	(0,0)
T13	(0,0)	(0,0)
T14	(0,0)	(0,0)
T15	(27,27)	(59,59)
T16	(72,72)	(154,154)

$$\begin{aligned} ECBC &= T1 + T2 + T4 + T6 + T13 + T15 \\ ECBC &= 0 + 0 + 45 + 0 + 0 + 59 \\ ECBC &= 104 \end{aligned} \tag{3·3}$$

$$\begin{aligned} ECWC &= T7 + T9 + T11 + T12 + T14 + T16 \\ ECWC &= 0 + 0 + 108 + 0 + 0 + 154 \\ ECWC &= 262 \end{aligned} \tag{3·4}$$

Tabela 3.4: Métricas.

Métrica	Resultado
BCET	42 segundos
WCET	150 segundos
ECBC	104 *joules*
ECWC	262 *joules*

É importante enfatizar que através dessas análises será possível realizar estimativas/verificações, antes mesmo do *software/hardware* ser concedido. Esse aspecto traz uma série de benefícios: orienta a escolha dos componentes de *hardware*/*software* do sistema a ser desenvolvido, auxilia na definição da política de escalonamento e permite a depuração das especificações.

3.7 AVALIAÇÃO

O objetivo desta etapa da metodologia é avaliar os resultados obtidos na fase anterior a fim de validar as especificações dos sistemas.

Se os resultados obtidos da análise quantitativa e qualitativa dos DCS estão de acordo com o especificado nos requisitos, então o desenvolvimento do ERTS é iniciado, pois a especificação atende aos requisitos não funcionais (tempo e energia) e funcionais (ex.: ausência de *deadlock*). Caso os resultados não estejam em conformidade com o especificado, a metodologia proposta volta para os primeiros passos, podendo realizar ajustes tanto nos requisitos quanto nos DCS. Uma vez realizados todos os ajustes necessários, então os passos subseqüentes da metodologia são executados novamente com o objetivo de verificar a conformidade com os requisitos.

3.8 CONSIDERAÇÕES FINAIS

Este capítulo apresentou uma metodologia para avaliação de desempenho das especificações dos sistemas embarcados crítico, levando em consideração as restrições de tempo e anotações de energia. Em resumo, esta metodologia baseia-se, principalmente, na composição sistemática dos modelos básicos gerados pelo processo de mapeamento dos diagramas comportamentais da SysML, para então se obter um modelo que representa a especificação do sistema, e assim, realizar análises e verificações.

CAPÍTULO 4

MAPEAMENTO DOS DIAGRAMAS DA SYSML EM UM MODELO ETPN

Este capítulo apresenta o mapeamento dos diagramas comportamentais SysML (estados, atividades e seqüencia) em modelos ETPN, incluindo as restrições de tempo e anotações de energia. Para tanto, são apresentados os principais elementos que compõe os diagramas, bem como as regras de mapeamento desses elementos. Dessa forma, pode-se modelar cenários complexos, sem que se tenha um profundo conhecimento em RdP.

4.1 MAPEAMENTO DO DIAGRAMA DE ESTADOS EM UMA ETPN

Esta seção apresenta os principais elementos que compõem o DE, assim como as regras de mapeamento em modelos ETPN [AMC+08]. Os diagramas de estados mostram os possíveis estados de um entidade e as transições responsáveis pelas suas mudanças de estado. As entidades, como mencionado anteriormente, representam os elementos que são modelados.

4.1.1 Mapeamento dos Estados

Um estado é uma condição de uma entidade em que ela realiza alguma atividade ou espera um evento. Graficamente os estados são representados por elipses ou retângulos com bordas arredondadas, apresentando duas ou três divisões (ver Figura 4.1). A primeira divisão mostra o nome do estado. A segunda divisão apresenta as atividades internas (*entry*, *do* e *exit*). *Entry* representa as ações realizadas no momento em que a entidade entra no estado. *Exit* identifica as ações executadas antes da entidade mudar de estado e *do* ilustra as atividades executadas enquanto a entidade se encontra no estado. Por fim, a terceira divisão que identifica as transições internas. As transições internas são descritas na seção 4.1.2.

A Figura 4.1 apresenta um exemplo de um estado. Como é possível perceber na figura, o estado *Ocupado* contém atividades internas (*entry*, *do* e *exit*), transição interna e ações que podem ser executadas tanto durante as atividades quanto durante a transição interna. Além disso, o estado contém restrições de tempo e anotações de energia, isto é, o estado possui um tempo de execução no pior caso de 45 segundos e no melhor caso de 5 segundos. De maneira similar, o estado possui um consumo de energia no pior e melhor caso de, respectivamente, 70 e 18 *joules*. É importante ressaltar que uma ação é algo instantâneo, que não demanda tempo, enquanto uma atividade dura um certo tempo.

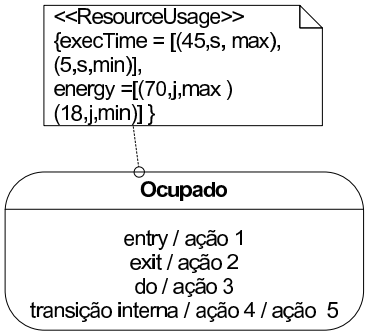


Figura 4.1: Exemplo de um estado simples.

A Figura 4.2 (a) apresenta o mapeamento de um estado simples em um modelo ETPN. Um intervalo estreito (intervalo cujos os valores do limite superior e inferior são iguais) do tempo de execução no pior e melhor caso de, respectivamente, [35,35] e [20,20] são atribuídos às transições-PN *t_ex_W_A* e *t_ex_B_A*. Em outras palavras, o tempo de execução máximo e mínimo das atividades (*entry*, *do* e *exit*) e da transição interna do estado simples *A*. De forma semelhante, a Figura 4.2 (b) ilustra o mapeamento de um estado simples em um modelo ETPN. No entanto, nesse caso, as anotações de MARTE são usadas para representar tanto as restrições do tempo de execução, quanto as anotações de consumo de energia do estado *A*. Neste exemplo, o consumo de energia no melhor e pior caso atribuídos às transições-PN são iguais a [70,70] e [18,18], respectivamente.

Porém, se as restrições de tempo forem omitidas dos estados simples, então esses estados são mapeados em transições-PN, nas quais o tempo máximo e mínimo é zero (ver Figura 4.2 (c)). Os lugares *in_A*, *W_A*, *B_A* e *out_A*, representam, respectivamente, a entrada no estado *A*, o estado de pior caso, o estado de melhor caso e a saída do estado *A*.

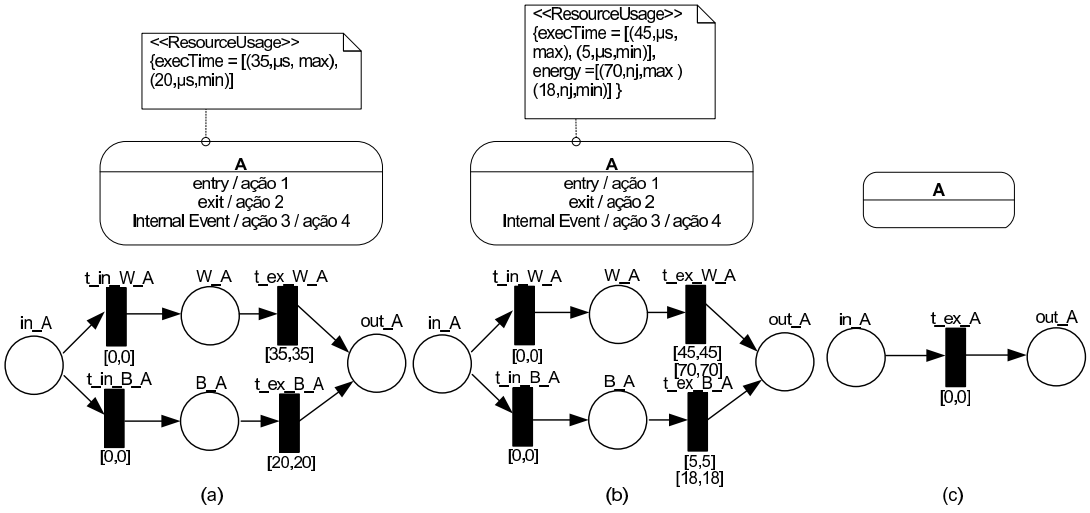


Figura 4.2: Mapeamento dos estados simples.

Adicionalmente, as transições-PN *t_in_W_A* e *t_in_B_A* são adicionadas ao modelo ETPN devido à semântica das redes de Petri temporizadas (semântica de disparo *strong* (ver Seção 2.5.9)), a fim de permitir que as transições-PN com retardos maiores sejam capazes de disparar. Para tanto, um intervalo estreito igual a [0,0] é atribuído a essas transições-PN. Além disso, as transições-PN *t_ex_W_A* e *t_ex_B_A* do modelo ETPN (ver Figura 4.2 (a)) são atribuídas com um intervalo estreito de, respectivamente, [35,35] e [20,20], pois o espaço de estados é substancialmente menor do que aquele gerado se apenas uma única transição-PN fosse adotada com o intervalo igual a [35,20], como pode ser observado na Figura 5.16.

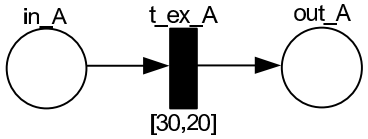


Figura 4.3: Modelo ETPN comparativo.

Definição 4.1. (Estado com restrições) O modelo do estado é uma ETPN definida como $ER = (P^{ER}, T^{ER}, F^{ER}, W^{ER}, {m_0}^{ER}, I^{ER}, \mathcal{E}^{ER})$, onde:

- $P^{ER} = \{in\_A, W\_A, B\_A, out\_A\}$;
- $T^{ER} = \{t\_in\_W\_A, t\_in\_B\_A, t\_ex\_W\_A, t\_ex\_B\_A\}$;
- $F^{ER} = \{(in\_A, t\_in\_W\_A), (in\_A, t\_in\_B\_A), (t\_in\_W\_A, W\_A), (t\_in\_B\_A, B\_A), (W\_A, t\_ex\_W\_A), (B\_A, t\_ex\_B\_A), (t\_ex\_W\_A, out\_A), (t\_ex\_B\_A, out\_A)\}$;
- $W^{ER}(f) = 1, \forall\ f \in F^{ER}$;
- ${m_0}^{ER}(p) = 0, \forall\ p \in P^{ER}$;
- $I^{ER}(t) = (EFT^{ER}(t), LFT^{ER}(t)), \forall t \in T^{ER}$, onde:
  $EFT^{ER}(t) = LFT^{ER}(t) = \begin{cases} X_E^{ER}, & \text{se } t = t\_ex\_W\_A; \\ X_L^{ER}, & \text{se } t = t\_ex\_B\_A; \\ 0, & \text{caso contrário.} \end{cases}$
- $\mathcal{E}^{ER}\ (t) = \begin{cases} Y_E^{ER}, & \text{se } t = t\_ex\_W\_A; \\ Y_L^{ER}, & \text{se } t = t\_ex\_B\_A; \\ 0, & \text{caso contário.} \end{cases}$

onde X_E^{ER} e X_L^{ER} são os tempos de execução no melhor e pior caso associados às transições *t_ex_B_A* e *t_ex_W_A*, respectivamente, e Y_E^{ER} e Y_L^{ER} são os consumos de energia no melhor e pior caso associados às transições *t_ex_B_A* e *t_ex_W_A*, respectivamente.

Definição 4.2. (Estado sem restrições) O modelo do estado sem restrições é uma ETPN definida como $ESR = (P^{ESR}, T^{ESR}, F^{ESR}, W^{ESR}, {m_0}^{ESR}, I^{ESR}, \mathcal{E}^{ESR})$, onde:

- $P^{ESR} = \{in\_A, out\_A\}$;
- $T^{ESR} = \{t\_ex\_A\}$;
- $F^{ESR} = \{(in\_A, t\_ex\_A), (t\_ex\_A, out\_A)\}$;
- $W^{ESR}(f) = 1, \forall\ f \in F^{ESR}$;
- ${m_0}^{ESR}(p) = 0, \forall\ p \in P^{ESR}$;
- $I^{ESR}(t) = (EFT^{ESR}(t), LFT^{ESR}(t)), \forall t \in T^{ESR}$, onde:
  $EFT^{ESR}(t) = LFT^{ESR}(t) = 0 \forall t \in T^{ESR}$
- $\mathcal{E}^{ESR}(t) = 0 \forall t \in T^{ESR}$

4.1.2 Transições Internas

As transições internas são transições que não provocam uma mudança de estado. Normalmente, o evento associado à transição interna é acompanhado de uma ação. Essa ação é executada quando ocorre o evento, porém não provoca nenhuma mudança no estado da entidade.

A Figura 4.4 demonstra um exemplo de uma transição interna causada pelo disparo do evento *dar troco*. As atividades *entry* e *exit* não são executadas e a atividade em execução *do* não é interrompida. Percebe-se então que a entidade ao chegar no estado *Máquina de Refrigerante* executa as ações *Colocar Dinheiro* e *Colocar Copo*, por conseguinte, enquanto a entidade se encontra no estado, é executada a ação *Colocar Refrigerante no Copo*. Durante a execução dessa ação, é possível executar também a transição internar *dar troco*, na qual disparará a ação *Devolver Troco*. Durante a execução da ação da transição interna, nenhuma outra ação poderá ser executada, apenas a ação *Colocar Refrigerante no Copo* e a própria transição interna. Por fim, antes da entidade mudar de estado, ela executa a ação *Retirar Copo*.

Figura 4.4: Exemplo de transição interna.

4.1.3 Mapeamento das Transições

Segundo [GUE04], uma transição representa um evento que causa uma mudança no estado da entidade, gerando um novo estado. As transições são graficamente representadas por uma seta apontando para o estado destino. Adicionalmente, uma transição pode ter

várias origens (nesse caso, ela representa uma junção de vários estados simultâneos) e vários alvos (nesse caso, ela representa uma forquilha para vários estados simultâneos). Uma transição pode possuir um rótulo com a seguinte sintaxe:

Evento(Argumentos)[Condição]/Ação

O rótulo pode ser dividido nas seguintes partes: evento, argumentos, condição e ação. Evento é algo que acontece em algum ponto no tempo e que pode modificar o estado de uma entidade. Os eventos podem ser dos seguintes tipos: sinais, chamadas, eventos temporais e eventos de mudanças. Sinais são eventos simbólicos sinalizados explicitamente. Chamadas são invocações de operações. Eventos temporais representam a passagem de tempo (ex.: *after(30segundos)*). Eventos de mudanças são representados por uma condição que se torna verdadeira (ex.: *when(saldo > 0)*). Argumentos são valores recebidos junto com o evento. Condição representa uma condição lógica que precisa ser satisfeita para habilitar a transição. Por fim, a ação consiste em um processamento atômico que resulta em uma mudança de estado no sistema. Além disso, as transições podem conter mais de uma ação e podem existir com ou sem eventos, com ou sem condições de guarda e com ou sem ações.

Na Figura 4.5, é apresentado uma transição relativa ao processo de abertura de conta. Neste exemplo existem dois estados, no primeiro estado, a entidade é consultada pelo CPF, ou seja, a entidade executa o método de consulta por CPF (*ConCPF()*). Após essa consulta, o funcionário pode atualizar o cadastro, este evento causa uma transição de estados, o que gera o novo estado *Atualizando Pessoa*. Nesse estado a entidade executa o método *Gravar()*. É importante ressaltar que durante a transição de estados as anotações de MARTE são usadas, ou seja, a transição possui o tempo de execução no pior caso de 10 segundos e no melhor caso de 3 segundos. De maneira similar, a transição de estados também possui um consumo de energia. Neste exemplo, o consumo de energia no pior e melhor caso são de, respectivamente, 50 e 30 *joules*.

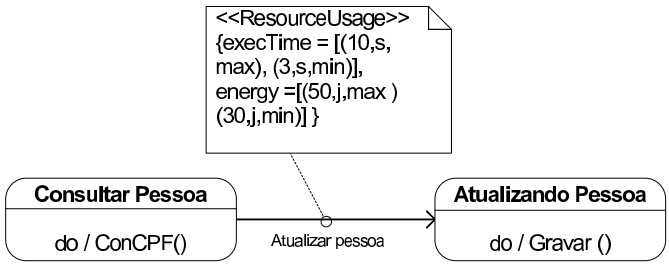


Figura 4.5: Exemplo de uma transição.

A Figura 4.6 (a) ilustra o mapeamento de uma transição do Diagrama de Estados (transição-DE) com restrições de tempo. A transição-DE que sai do estado A com destino ao B é mapeada, basicamente, em duas transições-PN (*t_ex_W_t1* e *t_ex_B_t1*), onde intervalos estreitos, os quais representam o tempo de execução no pior e melhor caso, iguais a [40,40] e [10, 10] segundos são atribuídos a essas transições-PN. A Figura 4.6 (b)

também apresenta o mapeamento de uma transição-DE, no entanto, as transições-PN possuem restrições de tempo e anotações de energia. É importante destacar que alguns elementos gerados pelo mapeamento da transição-DE e dos demais elementos que serão apresentados posteriormente são exatamente iguais aos gerados pelo mapeamento do estado simples, tais como os lugares *W_A* e *B_A* e as transições-PN *t_in_W_A* e *t_in_B_A*. Assim esses elementos não serão explicados novamente.

Além disso, transições-DE que não possuem restrições de tempo são mapeadas em transições-PN, nas quais o tempo máximo e mínimo é zero (ver Figura 4.6 (c)). Adicionalmente, transições-DE que não possuem eventos rotulados são mapeadas conforme a Figura 4.7, onde a transição-PN *t_A_B* é usada para representar a transição-DE. Os lugares *out_A* e *in_B* representam, respectivamente, a saída do estado A e à entrada no estado *B*.

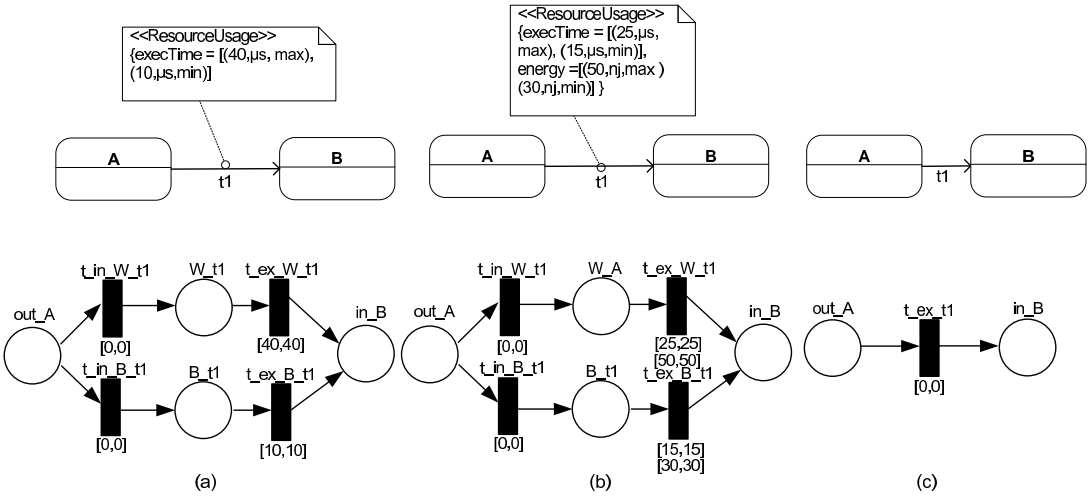


Figura 4.6: Mapeamento das transições.

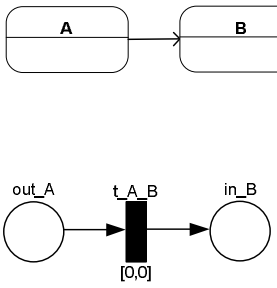


Figura 4.7: Mapeamento das transições sem rótulo.

Definição 4.3. (Transição com restrições) O modelo da transição com restrições é uma ETPN definida como TR= $(P^{TR}, T^{TR}, F^{TR}, W^{TR}, {m_0}^{TR}, I^{TR}, \mathcal{E}^{TR})$, onde:

- $P^{TR} = \{out\_A, W\_t1, B\_t1, in\_B\}$;

- $T^{TR} = \{t\_in\_W\_t1, t\_in\_B\_t1, t\_ex\_W\_t1, t\_ex\_B\_t1\}$;
- $F^{TR} = \{(out\_A, t\_in\_W\_t1), (out\_A, t\_in\_B\_t1), (t\_in\_W\_t1, W\_t1), (t\_in\_B\_t1, B\_t1), (W\_t1, t\_ex\_W\_t1), (B\_t1, t\_ex\_B\_t1), (t\_ex\_W\_t1, in\_B), (t\_ex\_B\_t1, in\_B)\}$;
- $W^{TR}(f) = 1, \forall\ f \in F^{TR}$;
- ${m_0}^{TR}(p) = 0, \forall\ p \in P^{TR}$;
- $I^{TR}(t) = (EFT^{TR}(t), LFT^{TR}(t)), \forall t \in T^{TR}$, onde:
  $EFT^{TR}(t) = LFT^{TR}(t) = \begin{cases} X_E^{TR}, & \text{se } t = t\_ex\_W\_t1; \\ X_L^{TR}, & \text{se } t = t\_ex\_B\_t1; \\ 0, & \text{caso contrário.} \end{cases}$
- $\mathcal{E}^{TR}\ (t) = \begin{cases} Y_E^{TR}, & \text{se } t = t\_ex\_W\_t1; \\ Y_L^{TR}, & \text{se } t = t\_ex\_B\_t1; \\ 0, & \text{caso contário.} \end{cases}$

onde X_E^{TR} e X_L^{TR} são os tempos de execução no melhor e pior caso associados às transições *t_ex_B_t1* e *t_ex_W_t1*, respectivamente, e Y_E^{TR} e Y_L^{TR} são os consumos de energia no melhor e pior caso associados às transições *t_ex_B_t1* e *t_ex_W_t1*, respectivamente.

Definição 4.4. (Transição sem restrições) O modelo da transição sem restrição é uma ETPN definida como TSR= $(P^{TSR}, T^{TSR}, F^{TSR}, W^{TSR}, {m_0}^{TSR}, I^{TSR}, \mathcal{E}^{TSR})$, onde:

- $P^{TSR} = \{out\_A, in\_B\}$;
- $T^{TSR} = \{t\_ex\_t1\}$;
- $F^{TSR} = \{(out\_A, t\_ex\_t1), (t\_ex\_t1, in\_B)\}$;
- $W^{TSR}(f) = 1, \forall\ f \in F^{TSR}$;
- ${m_0}^{TSR}(p) = 0, \forall\ p \in P^{TSR}$;
- $I^{TSR}(t) = (EFT^{TSR}(t), LFT^{TSR}(t)), \forall t \in T^{TSR}$, onde:
  $EFT^{TSR}(t) = LFT^{TSR}(t) = 0\ \forall t \in T^{TSR}$;
- $\mathcal{E}^{TSR}(t) = 0\ \forall t \in T^{TSR}$.

4.1.4 Mapeamento das Auto-transições

Uma auto-transição é uma transição que sai do seu estado corrente e retorna ao mesmo estado, provocando de fato uma reentrada no estado corrente. A diferença entre as transições internas e as auto-transições é que as primeiras não saem do estado, enquanto as auto-transições saem do estado atual, podendo executar alguma ação e depois retornar ao mesmo estado.

O mapeamento de uma auto-transição é ilustrado na Figura 4.8. Nesse exemplo, a auto-transição *sel1* possui o tempo de execução no pior caso de 30 μs e no melhor caso de 10 μs. As regras de mapeamento para esse tipo de transição são similares às mencionadas na Seção 4.1.3 para as transições de estados, porém o modelo ETPN da auto-transição retorna ao estado A através da transição-PN *t_sel1_A*. É importante ressaltar que a auto-transição só executa uma única vez.

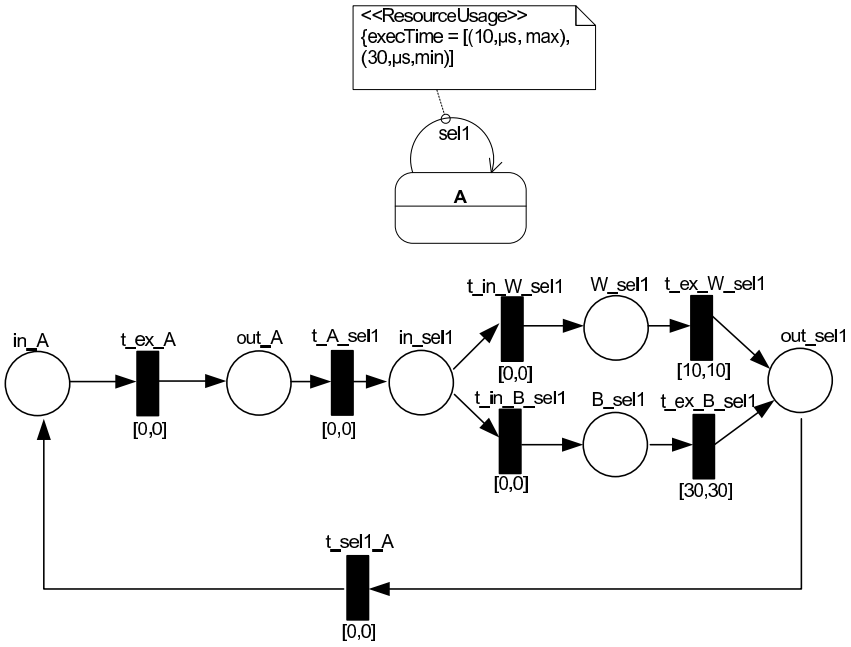


Figura 4.8: Mapeamento das auto-transições.

Definição 4.5. (Auto-transição com restrições) O modelo da auto-transição com restrições é uma ETPN definida como ATR= ($P^{ATR}, T^{ATR}, F^{ATR}, W^{ATR}, {m_0}^{ATR}, I^{ATR}$, $\mathcal{E}^{ATR}$), onde:

- $P^{ATR} = \{in\_A, out\_A, in\_sel1, W\_sel1, B\_sel1, out\_sel1\}$;
- $T^{ATR} = \{t\_ex\_A, t\_A\_sel1, t\_in\_W\_sel1, t\_in\_B\_sel1, t\_ex\_W\_sel1, t\_ex\_B\_sel1,$ $t\_sel1\_A\}$;
- $F^{ATR} = \{(in\_A, t\_ex\_A), (t\_ex\_A, out\_A), (out\_A, t\_A\_sel1), (t\_A\_sel1, in\_sel1),$ $(in\_sel1, t\_in\_W\_sel1), (in\_sel1, t\_in\_B\_sel1), (t\_in\_W\_sel1, W\_sel1), (t\_in\_B\_sel1,$

$B\_sel1), (W\_sel1, t\_ex\_W\_sel1), (B\_sel1, t\_ex\_B\_sel1), (t\_ex\_W\_sel1, out\_sel1),$
$(t\_ex\_B\_sel1, out\_sel1), (out\_sel1, t\_sel1\_A), (t\_sel1\_A, in\_A)\}$;

- $W^{ATR}(f) = 1, \forall\ f \in F^{ATR}$;
- ${m_0}^{ATR}(p) = 0, \forall\ p \in P^{ATR}$;
- $I^{ATR}(t) = (EFT^{ATR}(t), LFT^{ATR}(t)), \forall t \in T^{ATR}$, onde:
  $EFT^{ATR}(t) = LFT^{ATR}(t) = \begin{cases} X_E^{ATR}, & \text{se } t = t\_ex\_W\_sel1; \\ X_L^{ATR}, & \text{se } t = t\_ex\_B\_sel1; \\ 0, & \text{caso contrário.} \end{cases}$
- $\mathcal{E}^{ATR}(t) = 0\ \forall t \in T^{ATR}$.

onde X_E^{ATR} e X_L^{ATR} são os tempos de execução no melhor e pior caso associados às transições *t_ex_B_sel1* e *t_ex_W_sel1*, respectivamente.

4.1.5 Mapeamento dos Estados Inicial e Final

O estado inicial é um pseudo-estado cuja função é indicar (apontar para) o ponto de partida na qual o DE ou os estados compostos serão analisados. Esse pseudo-estado é representado por um círculo preenchido (ver Figura 4.9). O estado inicial é mapeado em um lugar (*staIni_A*) no modelo ETPN, no qual contém a marcação inicial igual a um *token*. *Tokens* são usados nos modelos para simular o comportamento dinâmico dos sistemas. Além disso, a transição-PN *t_in_A* é usada para representar a transição-DE entre o estado inicial e o estado simples *A*. A Figura 4.9 apresenta o mapeamento.

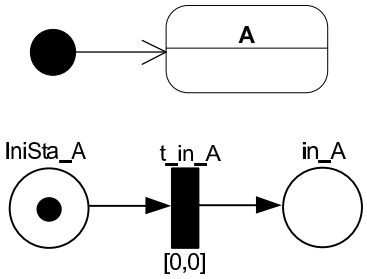


Figura 4.9: Mapeamento do estado inicial.

O estado final é um pseudo-estado cuja função é determinar o fim tanto do DE quanto dos estados compostos. Esse pseudo-estado é representado por um círculo não preenchido envolvendo um segundo círculo preenchido, como pode ser observado na Figura 4.10. O estado final é mapeado em um lugar (*endSta_A*) no modelo ETPN, onde a presença de um token nesse lugar representa o fim do DE ou do estado composto. Além disso, a transição-PN *t_end_A* é usada para representar a transição-DE entre o estado simples *A* e o estado final.

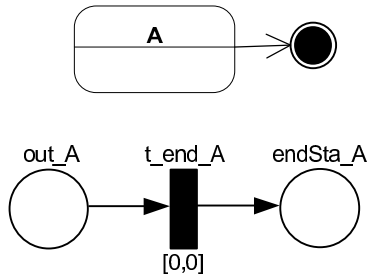


Figura 4.10: Mapeamento do estado final.

4.1.6 Mapeamento dos Estados Compostos

O diagrama de estados da SysML admite o conceito de estados aninhados, permitindo visões de alto nível e de baixo nível do comportamento e estado da entidade. Um estado composto pode pode ser decomposto em: estados seqüenciais ou concorrentes.

Estados compostos seqüenciais são aqueles para os quais existe uma subdivisão em estados mais simples. A Figura 4.11 apresenta um exemplo de um estado composto. O estado *Atualizando Pessoa* foi sub-dividido em dois estados (*Validando CPF* e *Gravando Pessoa*); esses dois subestados são utilizados para detalhar o estado *Atualizando Pessoa*.

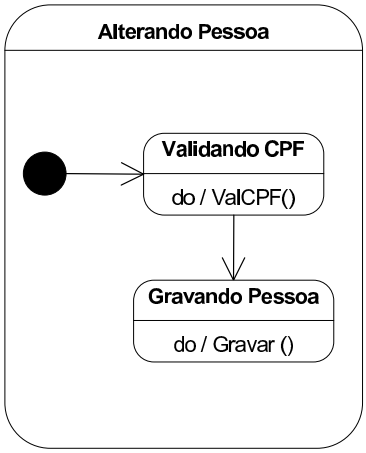


Figura 4.11: Exemplo de um estado composto seqüencial.

Fonte: [GUE04, p. 144]

No exemplo da Figura 4.12, é apresentado o mapeamento de um estado composto seqüencial. Como pode ser observado na figura, o estado B foi sub-divido em dois subestados (C e D), no entanto o subestado C possui restrições de tempo. As regras de mapeamento dos subestados são iguais às outras mencionadas anteriormente para os estados simples e transições de estados.

Definição 4.6. (Estados compostos seqüencial com restrições) O modelo dos estados compostos seqüencial com restrições é uma ETPN definida como $CON = (P^{CON}, T^{CON}, F^{CON}, W^{CON}, {m_0}^{CON}, I^{CON}, \mathcal{E}^{CON})$, onde:

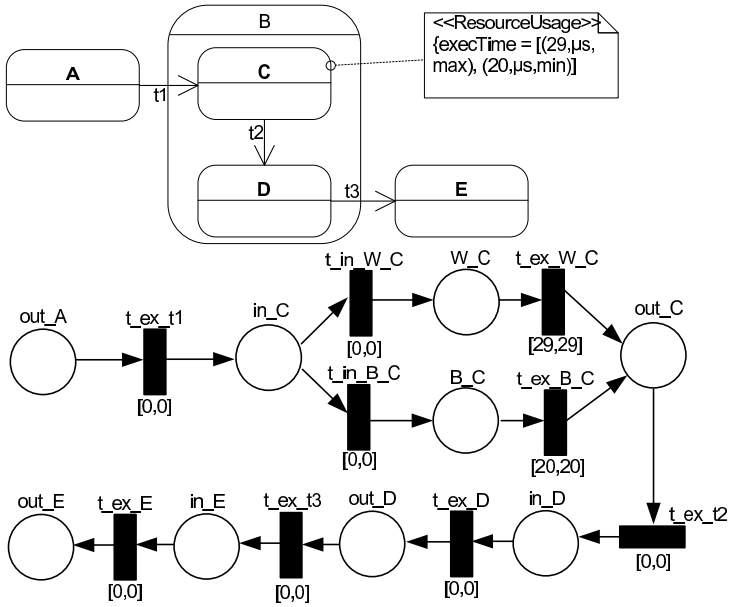


Figura 4.12: Mapeamento de um estado composto seqüencial.

- $P^{CON} = \{out\_A, in\_C, W\_C, B\_C, out\_C, in\_D, out\_D, in\_E, out\_E\}$;
- $T^{CON} = \{t\_ex\_t1, t\_in\_W\_C, t\_in\_B\_C, t\_ex\_W\_C, t\_ex\_B\_C, t\_exe\_t2, t\_exe\_D, t\_exe\_t3, t\_exe\_E\}$;
- $F^{CON} = \{(out\_A, t\_ex\_t1), (t\_ex\_t1, in\_C), (in\_C, t\_in\_W\_C), (in\_C, t\_in\_B\_C), (t\_in\_W\_C, W\_C), (t\_in\_B\_C, B\_C), (W\_C, t\_ex\_W\_C), (B\_C, t\_ex\_B\_C), (t\_ex\_W\_C, out\_C), (t\_ex\_B\_C, out\_C), (out\_C, t\_ex\_t2), (t\_ex\_t2, in\_D), (in\_D, t\_ex\_D), (t\_ex\_D, out\_D), (out\_D, t\_ex\_t3), (t\_ex\_t3, in\_E), (in\_E, t\_ex\_E), (t\_ex\_E, out\_E)\}$;
- $W^{CON}(f) = 1, \forall\ f \in F^{CON}$;
- ${m_0}^{CON}(p) = 0, \forall\ p \in P^{CON}$;
- $I^{CON}(t) = (EFT^{CON}(t), LFT^{CON}(t)), \forall t \in T^{CON}$, onde:
  $EFT^{CON}(t) = LFT^{CON}(t) = \begin{cases} X_E^{CON}, & \text{se } t = t\_ex\_W\_C; \\ X_L^{CON}, & \text{se } t = t\_ex\_B\_C; \\ 0, & \text{caso contrário.} \end{cases}$
- $\mathcal{E}^{CON}(t) = 0\ \forall t \in T^{CON}$.

onde X_E^{CON} e X_L^{CON} são os tempos de execução no melhor e pior caso associados às transições $t\_ex\_B\_C$ e $t\_ex\_W\_C$, respectivamente.

Um estado concorrente é um estado que divide seus estados em regiões. Cada região contém diagramas de estados distintos, que podem ser executados concorrentemente. A

Figura 4.13 apresenta um exemplo, onde para que o carro arranque é necessário que as ações (*Soltar embreagem* e *Pressionar acelerador*) dos estados *Soltando embreagem* e *Pressionando Acelerador* ocorram em paralelo.

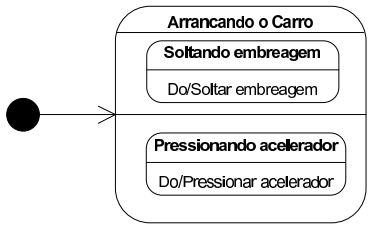


Figura 4.13: Exemplo de estados concorrentes.

Fonte: [GUE04, p. 146]

Um exemplo do mapeamento dos estados concorrentes é apresentado na Figura 4.14. Esse tipo de mapeamento é adotado para modelar atividades concorrentes, onde duas transições-PN *t_in_par_C_D* e *t_sin_C_D* são usadas para representar, respectivamente, o início e o fim das atividades concorrentes. Além disso, o diagrama de cada região é mapeado de acordo com as regras de mapeamento mencionadas anteriormente.

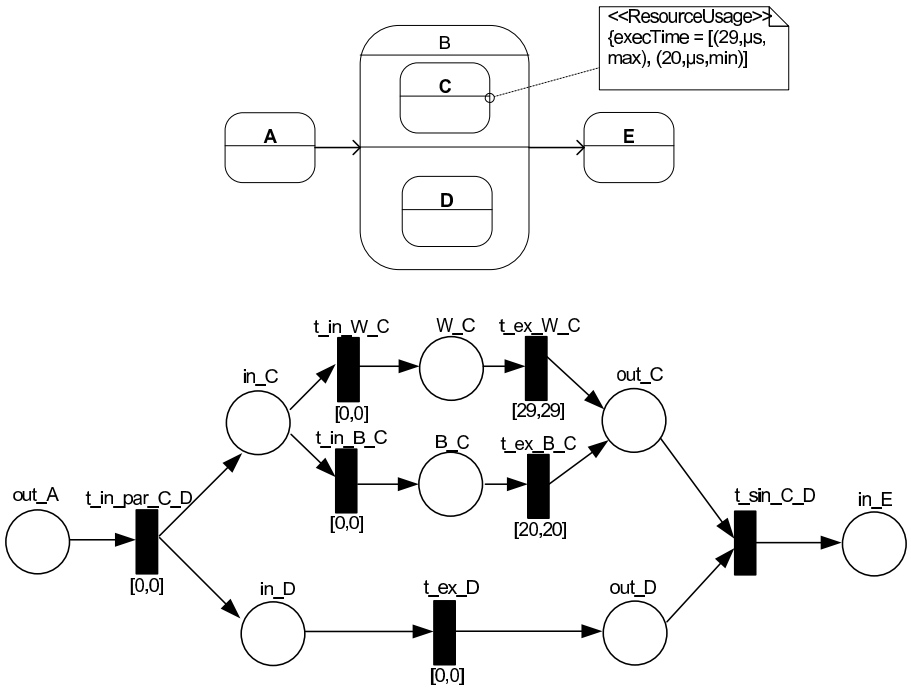


Figura 4.14: Mapeamento dos estados concorrentes.

Definição 4.7. (Estados concorrentes com restrições) O modelo dos estados concorrentes com restrições é uma ETPN definida como $COC = (P^{COC}, T^{COC}, F^{COC}, W^{COC}, {m_0}^{COC}, I^{COC}, \mathcal{E}^{COC})$, onde:

- $P^{COC} = \{out\_A, in\_C, W\_C, B\_C, out\_C, in\_D, out\_D, in\_E\}$;
- $T^{COC} = \{t\_in\_par\_C\_D, t\_in\_W\_C, t\_in\_B\_C, t\_ex\_W\_C, t\_ex\_B\_C, t\_sin\_C\_D, t\_exe\_D, t\_exe\_t3, t\_exe\_E\}$;
- $F^{COC} = \{(out\_A, t\_in\_par\_C\_D), (t\_in\_par\_C\_D, in\_C), (t\_in\_par\_C\_D, in\_D), (in\_C, t\_in\_W\_C), (in\_C, t\_in\_B\_C), (t\_in\_W\_C, W\_C), (t\_in\_B\_C, B\_C), (W\_C, t\_ex\_W\_C), (B\_C, t\_ex\_B\_C), (t\_ex\_W\_C, out\_C), (t\_ex\_B\_C, out\_C), (out\_C, t\_sin\_C\_D), (in\_D, t\_ex\_D), (t\_ex\_D, out\_D), (out\_D, t\_sin\_C\_D), (t\_sin\_C\_D, in\_E)\}$;
- $W^{COC}(f) = 1, \forall\ f \in F^{COC}$;
- ${m_0}^{COC}(p) = 0, \forall\ p \in P^{COC}$;
- $I^{COC}(t) = (EFT^{COC}(t), LFT^{COC}(t)), \forall t \in T^{COC}$, onde:
  $EFT^{COC}(t) = LFT^{COC}(t) = \begin{cases} X_E^{COC}, & \text{se } t = t\_ex\_W\_C; \\ X_L^{COC}, & \text{se } t = t\_ex\_B\_C; \\ 0, & \text{caso contrário.} \end{cases}$
- $\mathcal{E}^{COC}\ (t) = \begin{cases} Y_E^{COC}, & \text{se } t = t\_ex\_W\_C; \\ Y_L^{COC}, & \text{se } t = t\_ex\_B\_C; \\ 0, & \text{caso contário.} \end{cases}$

onde X_E^{COC} e X_L^{COC} são os tempos de execução no melhor e pior caso associados às transições $t\_ex\_B\_C$ e $t\_ex\_W\_C$, respectivamente, e Y_E^{COC} e Y_L^{COC} são os consumos de energia no melhor e pior caso associados às transições $t\_ex\_B\_C$ e $t\_ex\_W\_C$, respectivamente.

4.2 MAPEAMENTO DO DIAGRAMA DE ATIVIDADE EM UMA ETPN

Esta seção apresentará os principais elementos que compõem o Diagrama de Atividades (DA), assim como as regras de mapeamento em um modelo ETPN [AMCNc]. O objetivo desse diagrama é mostrar o fluxo de controle de uma atividade para outra, com suporte para comportamento condicional e paralelo. É importante ressaltar que alguns elementos do diagrama de atividades são semelhantes aos utilizado no diagrama de estados, tais como: estado final, estado inicial, escolha, transição, entre outros. Assim, esses elementos não serão apresentados novamente.

4.2.1 Atividades

As atividades representam a execução de um processamento não atômico, envolvendo uma ou mais ações. Uma ação consiste em um processamento que resulta em uma mudança de estado no sistema.

A seguir, na Figura 4.15, é apresentado um fluxo simples de um diagrama de atividades. Esse diagrama descreve um exemplo simplificado do processo de publicação de um artigo. A atividade de publicar um artigo se inicia com o estudo do assunto no qual se deseja publicar. Uma vez entendido o assunto e obtidos os resultados necessários, inicia-se a atividade de escrita. Por fim, a última atividade consiste na submissão do artigo.

Figura 4.15: Exemplo de um DA.

As regras de mapeamento para as atividades do diagrama de atividades são iguais às apresentadas na Seção 4.1.1 para os estados do diagrama de estados. Como pode ser observado na Figura 4.16, a diferença em relação aos estados do diagrama de estados é que as transições-PN e os lugares do modelo ETPN representam as atividades do diagrama de atividades.

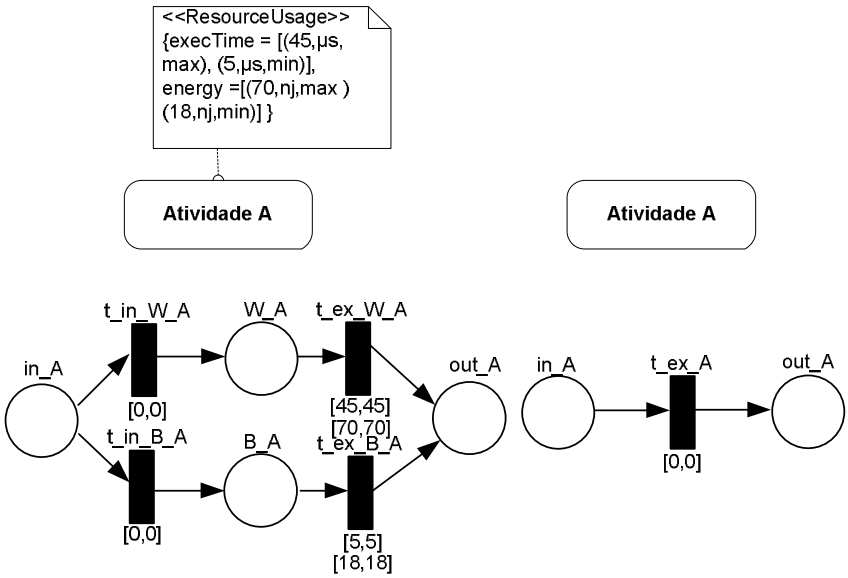


Figura 4.16: Mapeamento das atividades.

Definição 4.8. (Atividade com restrições) O modelo da atividade com restrições é uma ETPN definida como AR= $(P^{AR}, T^{AR}, F^{AR}, W^{AR}, {m_0}^{AR}, I^{AR}, \mathcal{E}^{AR})$, onde:

- $P^{AR} = \{in\_A, W\_A, B\_A, out\_A\}$;
- $T^{AR} = \{t\_in\_W\_A, t\_in\_B\_A, t\_ex\_W\_A, t\_ex\_B\_A\}$;
- $F^{AR} = \{(in\_A, t\_in\_W\_A), (in\_A, t\_in\_B\_A), (t\_in\_W\_A, W\_A), (t\_in\_B\_A, B\_A), (W\_A, t\_ex\_W\_A), (B\_A, t\_ex\_B\_A), (t\_ex\_W\_A, out\_A), (t\_ex\_B\_A, out\_A)\}$;

- $W^{AR}(f) = 1, \forall\ f \in F^{ER}$;
- ${m_0}^{AR}(p) = 0, \forall\ p \in P^{ER}$;
- $I^{AR}(t) = (EFT^{AR}(t), LFT^{AR}(t)), \forall t \in T^{ER}$, onde:
  $EFT^{AR}(t) = LFT^{AR}(t) = \begin{cases} X_E^{AR}, & \text{se } t = t\_ex\_W\_A; \\ X_L^{AR}, & \text{se } t = t\_ex\_B\_A; \\ 0, & \text{caso contrário.} \end{cases}$
- $\mathcal{E}^{AR}\ (t) = \begin{cases} Y_E^{AR}, & \text{se } t = t\_ex\_W\_A; \\ Y_L^{AR}, & \text{se } t = t\_ex\_B\_A; \\ 0, & \text{caso contário.} \end{cases}$

onde X_E^{AR} e X_L^{AR} são os tempos de execução no melhor e pior caso associados às transições $t\_ex\_B\_A$ e $t\_ex\_W\_A$, respectivamente, e Y_E^{AR} e Y_L^{AR} são os consumos de energia no melhor e pior caso associados às transições $t\_ex\_B\_A$ e $t\_ex\_W\_A$, respectivamente.

Definição 4.9. (Atividade sem restrições) O modelo da atividade sem restrições é uma ETPN definida como $ASR = (P^{ASR}, T^{ASR}, F^{ASR}, W^{ASR}, {m_0}^{ASR}, I^{ASR}, \mathcal{E}^{ASR})$, onde:

- $P^{ASR} = \{in\_A, out\_A\}$;
- $T^{ASR} = \{t\_ex\_A\}$;
- $F^{ASR} = \{(in\_A, t\_ex\_A), (t\_ex\_A, out\_A)\}$;
- $W^{ASR}(f) = 1, \forall\ f \in F^{ASR}$;
- ${m_0}^{ASR}(p) = 0, \forall\ p \in P^{ASR}$;
- $I^{ASR}(t) = (EFT^{ASR}(t), LFT^{ASR}(t)), \forall t \in T^{ASR}$, onde:
  $EFT^{ASR}(t) = LFT^{ASR}(t) = 0 \forall t \in T^{ASR}$
- $\mathcal{E}^{ASR}(t) = 0 \forall t \in T^{ASR}$

4.2.2 Barra de Sincronia

A Barra de Sincronização é utilizada para determinar o momento em que o processo passou a ser executado em paralelo e em quantos sub-processos se dividiu *(fork)*. Essa barra também é utilizada para determinar o momento em que dois ou mais sub-processos se uniram em um único processo (*join*). A Figura 4.17 apresenta um exemplo.

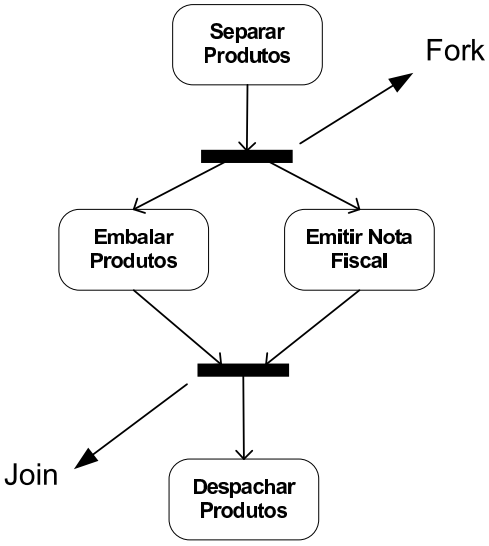


Figura 4.17: Barra de sincronia.

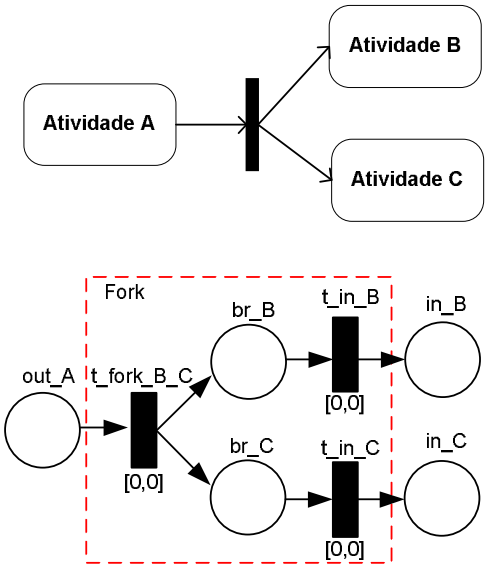


Figura 4.18: Mapeamento do *fork*.

4.2.2.1 Mapeamento do Fork A Figura 4.18 apresenta um exemplo, no qual a atividade A é subdividida em 2 atividades (B e C). Nesse mapeamento, a transição *t_fork_A_B* representa a subdivisão do estado A e os lugares *brc_B* e *brc_C* representam o início das ramificações, ou seja, o ponto de partida para a execução das atividades B e C concorrentemente.

Definição 4.10. (Fork) O modelo do *fork* é uma ETPN definida como $FK = (P^{FK}, T^{FK}, F^{FK}, W^{FK}, {m_0}^{FK}, I^{FK}, \mathcal{E}^{FK})$, onde:

- $P^{FK} = \{br\_B, br\_C\}$;
- $T^{FK} = \{t\_fork\_B\_C, t\_in\_B, t\_in\_C\}$;
- $F^{FK} = \{(t\_fork\_B\_C, br\_B), (t\_fork\_B\_C, br\_C), (br\_B, t\_in\_B), (br\_C, t\_in\_C)\}$;
- $W^{FK}(f) = 1, \forall\ f \in F^{FK_i}$;
- ${m_0}^{FK}(p) = 0, \forall\ p \in P^{FK}$;
- $I^{FK}(t) = (EFT^{FK}(t), LFT^{FK}(t)), \forall t \in T^{FK}$, onde:
  $EFT^{FK}(t) = LFT^{FK}(t) = 0 \forall t \in T^{FK}$
- $\mathcal{E}^{FK}(t) = 0\ \forall t \in T^{FK}$

4.2.2.2 Mapeamento do Join A Figura 4.19 apresenta o processo de sincronização, isto é, a sincronização entre as atividades A e B. No processo de mapeamento, a transição-PN *t_join_A_B* é usada para representar esse processo de sincronização, e o lugar *join_A_B* representa o ponto de sincronização. A transição-PN *t_join_A_B* só pode ser disparada se os lugares *out_A* e *out_B* contiverem *token*.

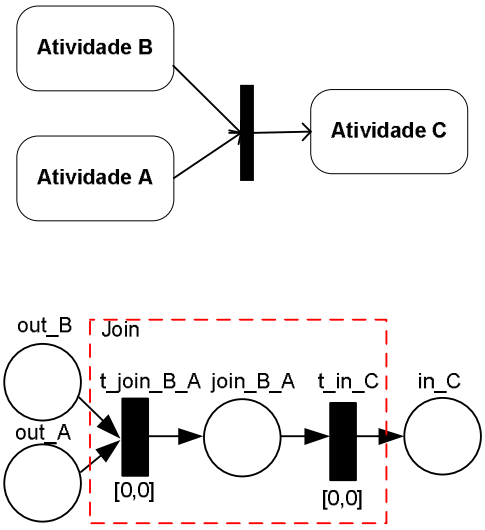


Figura 4.19: Mapeamento do *join*.

Definição 4.11. (Join) O modelo do *join* é uma ETPN definida como $JN = (P^{JN}, T^{JN}, F^{JN}, W^{JN}, {m_0}^{JN}, I^{JN}, \mathcal{E}^{JN})$, onde:

- $P^{JN} = \{join\_A\_B\}$;
- $T^{JN} = \{t\_join\_A\_B, t\_in\_C\}$;
- $F^{JN} = \{(t\_join\_A\_B, join\_A\_B), (join\_A\_B, t\_in\_C)\}$;

- $W^{JN}(f) = 1, \forall\ f \in F^{JN}$;
- ${m_0}^{JN}(p) = 0, \forall\ p \in P^{JN}$;
- $I^{JN}(t) = (EFT^{JN}(t), LFT^{JN}(t)), \forall t \in T^{JN}$, onde:
  $EFT^{JN}(t) = LFT^{JN}(t) = 0\ \forall t \in T^{JN}$
- $\mathcal{E}^{JN}(t) = 0\ \forall t \in T^{JN}$

4.2.3 Mapeamento da Decisão

A decisão representa um ponto do fluxo de controle, onde deve ser realizada uma tomada de decisão. Normalmente é representado por um losango com uma entrada e várias saídas. As saídas possuem condições de guarda que controlam qual transição (de um conjunto de transições alternativas) sucede a atividade a ser concluída. A Figura 4.20 apresenta um exemplo.

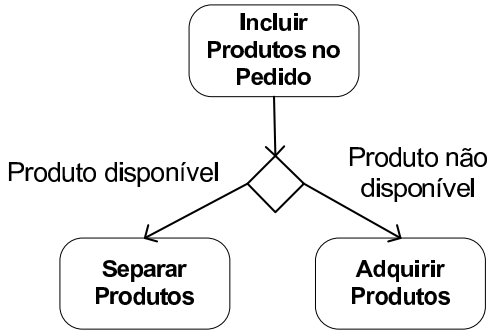


Figura 4.20: Decisão.

No processo de mapeamento da decisão (ver Figura 4.21), a transição-PN *t_in_deci* representa a entrada na decisão e o lugar *brc_B* representa o ponto de decisão entre as atividades B e C.

Definição 4.12. (Decisão) O modelo da decisão é uma ETPN definida como $DE = (P^{DE}, T^{DE}, F^{DE}, W^{DE}, {m_0}^{DE}, I^{DE}, \mathcal{E}^{DE})$, onde:

- $P^{DE} = \{deci\_B\_C\}$;
- $T^{DE} = \{t\_in\_deci, t\_in\_B, t\_in\_C\}$;
- $F^{DE} = \{(t\_in\_deci, deci\_B\_C), (deci\_B\_C, t\_in\_B), (deci\_B\_C, t\_in\_C)\}$;
- $W^{DE}(f) = 1, \forall\ f \in F^{DE_i}$;
- ${m_0}^{DE}(p) = 0, \forall\ p \in P^{DE}$;
- $I^{DE}(t) = (EFT^{DE}(t), LFT^{DE}(t)), \forall t \in T^{DE}$, onde:
  $EFT^{DE}(t) = LFT^{DE}(t) = 0\ \forall t \in T^{DE}$

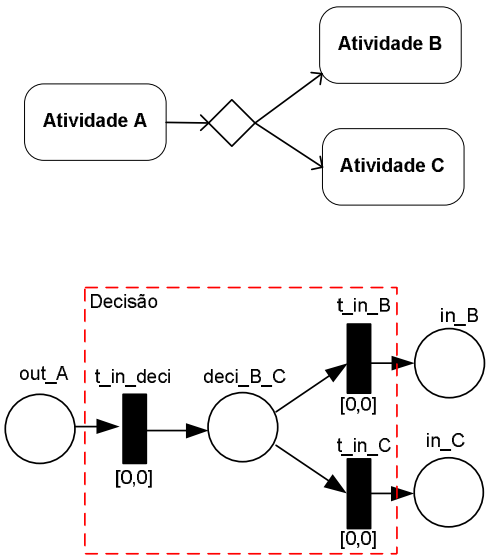


Figura 4.21: Mapeamento da decisão.

- $\mathcal{E}^{DE}(t) = 0 \; \forall t \in T^{DE}$

4.3 MAPEAMENTO DO DIAGRAMA DE SEQÜÊNCIA EM UMA ETPN

Esta seção apresentará os principais elementos que compõem o Diagrama de Seqüência (DS), assim como as regras de mapeamento em um modelo ETPN [AMCNb]. O DS enfatiza a seqüência temporal da troca de mensagens entre as entidades.

4.3.1 Mapeamento da Linha de Vida e Mensagens

Uma linha de vida representa o envolvimento de um dado participante em uma determinada interação. As linhas de vida são representadas por linhas finas verticais tracejadas partindo do retângulo que representa a entidade. Os retângulos brancos na linha de vida são chamados de ativações (ver Figura 4.22) e indicam que uma entidade está participando ativamente de um processo, ou seja, está respondendo/recebendo uma mensagem/chamada. A comunicação entre as linhas de vida é realizada através de mensagens ou chamadas, na mesma ordem em que os eventos ocorrem. Entende-se por mensagens os serviços solicitados de uma entidade a outra, e as respostas desenvolvidas para as solicitações. Neste trabalho as mensagens de retorno não são consideradas por dois motivos: (i) sua utilização leva a um grande número de setas e (ii) atrapalham o entendimento do diagrama como um todo.

A seguir, na Figura 4.22, é mostrado um pequeno exemplo, no qual uma mensagem é disparada entre duas entidades (A e B). Como pode ser observado na figura, a *Mensagem 1* contém restrições de tempo e anotações de energia. Em outras palavras a *Mensagem 1*

possui o tempo de execução no pior caso de 35 μs e no melhor caso 20 μs. De maneira similar, a *Mensagem 1* também possui um consumo de energia no pior caso de 100 ηj e no melhor caso 50 ηj. No entanto, a mensagem subseqüente do diagrama de seqüência não possui restrições de tempo ou anotações energia.

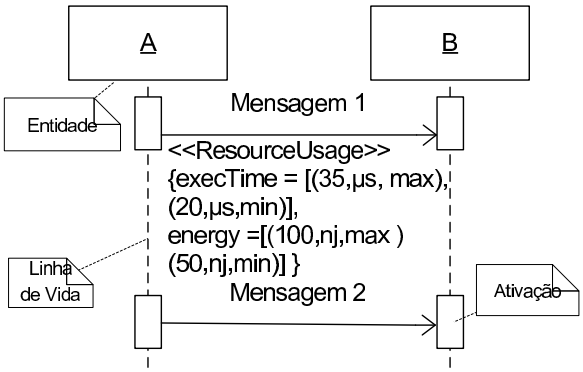


Figura 4.22: Exemplo de diagrama de seqüência.

A Figura 4.23 apresenta o mapeamento do DS ilustrado na Figura 4.22 em um modelo ETPN. Os lugares (*start_A*, *D_1* e *end_A*) e as transições-PN (*t_s_M1* e *t_s_M2*) representam a linha de vida da entidade *A*. Nesse modelo, os lugares representam os estados da entidade ao longo do tempo e as transições-PN representam a transição dos estados. O lugar *start_A* é usado para representar o início da linha de vida da entidade *A*, onde esse lugar possui a marcação inicial de um *token*. O lugar seguinte *D_1* é um lugar *dummy* usado para conectar as transições, e o último lugar *end_A* representa o fim da linha de vida. Por fim, as transições *t_s_M1* e *t_s_M2* representam o envio das mensagens. De maneira similar, o participante B também foi mapeado, a diferença, nesse caso, é que as transições-PN (*t_r_M1* e *t_r_M2*) representam o recebimento das mensagens.

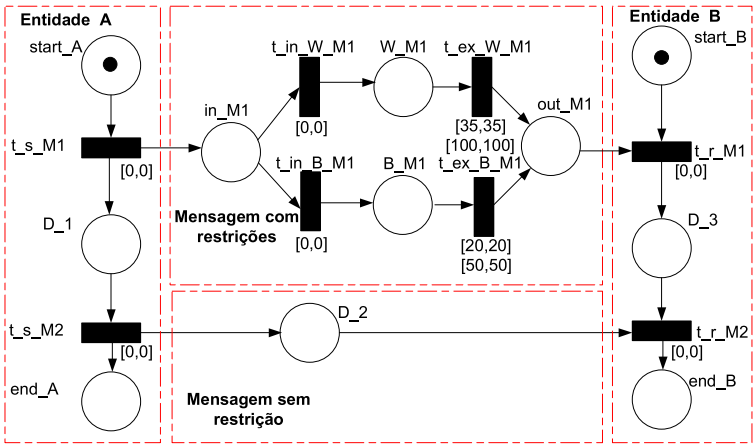


Figura 4.23: Mapeamento do diagrama de seqüência.

Na Figura 4.23 também é apresentado o mapeamento das mensagens. As regras de mapeamento para as mensagens são similares às outras mencionadas neste capítulo. A

Mensagem 1 é mapeada, basicamente, nas transições-PN *t_ex_W_t1* e *t_ex_B_t1*, onde intervalos estreitos, os quais representam o tempo de execução no pior e melhor caso, iguais a [35,35] e [20,20] segundos são atribuídos a essas transições-PN. De forma similar, o consumo de energia foi considerado e incluso no modelo ETPN. O consumo de energia no melhor e pior caso atribuídos às transições-PN foram [100,100] e [50,50]. Porém, se as restrições de tempo forem omitidas das mensagens, então o mapeamento ocorre como descrito na Figura 4.23, na qual o lugar *dummy D_2* é usado para interligar as transições-PN *t_s_M2* e *t_r_M2*. Os lugares *in_M1* e *out_M1*, representam, respectivamente, a entrada e saída da *Mensagem 1*.

Definição 4.13. (Linha de Vida e Mensagens) O modelo da linha de vida e das mensagens é uma ETPN definida como $LVM = (P^{LVM}, T^{LVM}, F^{LVM}, W^{LVM}, {m_0}^{LVM}, I^{LVM}, \mathcal{E}^{LVM})$, onde:

- $P^{LVM} = \{start\_A, in\_M1, W\_M1, B\_M1, out\_M1, start\_B, D\_3, D\_1, end\_A, D\_2, end\_B\}$;
- $T^{LVM} = \{t\_s\_M1, t\_in\_W\_M1, t\_in\_B\_M1, t\_ex\_W\_M1, t\_ex\_B\_M1, t\_r\_M1, t\_s\_M2, t\_r\_M2\}$;
- $F^{LVM} = \{(start\_A, t\_s\_M1), (t\_s\_M1, in\_M1), (in\_M1, t\_in\_W\_M1), (in\_M1, t\_in\_B\_M1), (t\_in\_W\_M1, W\_M1), (t\_in\_B\_M1, B\_M1), (W\_M1, t\_ex\_W\_M1), (B\_M1, t\_ex\_B\_M1), (t\_ex\_W\_M1, out\_M1), (t\_ex\_B\_M1, out\_M1)(out\_M1, t\_r\_M1), (start\_B, t\_r\_M1), (t\_r\_M1, D\_3), (D\_3, t\_r\_M2), (t\_s\_M1, D\_1), (D\_1, t\_s\_M2), (t\_s\_M2, end\_A), (t\_s\_M2, D\_2), (D\_2, t\_r\_M2), (t\_r\_M2, end\_B)\}$;
- $W^{LVM}(f) = 1, \forall\ f \in F^{LVM}$;
- ${m_0}^{LVM}(p) = \begin{cases} 1, & \text{se } p = start\_A; \\ 1, & \text{se } p = start\_B; \\ 0, & \text{caso contrário.} \end{cases}$
- $I^{LVM}(t) = (EFT^{LVM}(t), LFT^{LVM}(t)), \forall t \in T^{LVM}$, onde:
  $EFT^{LVM}(t) = LFT^{LVM}(t) = \begin{cases} X_E^{TR}, & \text{se } t = t\_ex\_W\_M1; \\ X_L^{TR}, & \text{se } t = t\_ex\_B\_M1; \\ 0, & \text{caso contrário.} \end{cases}$
- $\mathcal{E}^{LVM}\ (t) = \begin{cases} Y_E^{LVM}, & \text{se } t = t\_ex\_W\_M1; \\ Y_L^{LVM}, & \text{se } t = t\_ex\_B\_M1; \\ 0, & \text{caso contário.} \end{cases}$

onde X_E^{LVM} e X_L^{LVM} são os tempos de execução no melhor e pior caso associados às transições *t_ex_B_M1* e *t_ex_W_M1*, respectivamente, e Y_E^{LVM} e Y_L^{LVM} são os consumos de energia no melhor e pior caso associados às transições *t_ex_B_M1* e *t_ex_W_M1*, respectivamente.

4.3.2 Mapeamento das Auto-Mensagens

No exemplo da Figura 4.24, é apresentada uma auto-mensagem,ou seja, uma mensagem que a entidade envia para ela mesma.

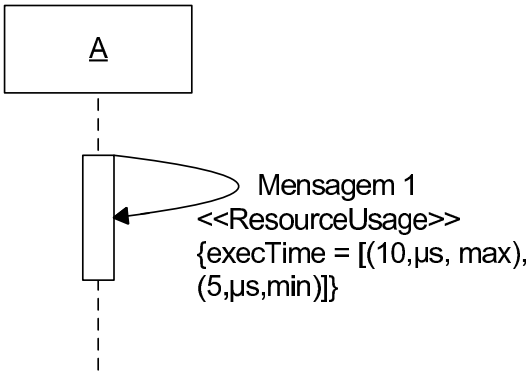


Figura 4.24: Exemplo de uma auto-mensagem.

O mapeamento de uma auto-mensagem é ilustrado na Figura 4.8. As regras de mapeamento para esse tipo de mensagem são similares às mencionadas na Seção 4.3.1, no entanto o modelo ETPN da auto-mensagem retorna para a mesma entidade através da transição-PN *t_r_M1*. É importante ressaltar que a auto-mensagem só executa uma única vez.

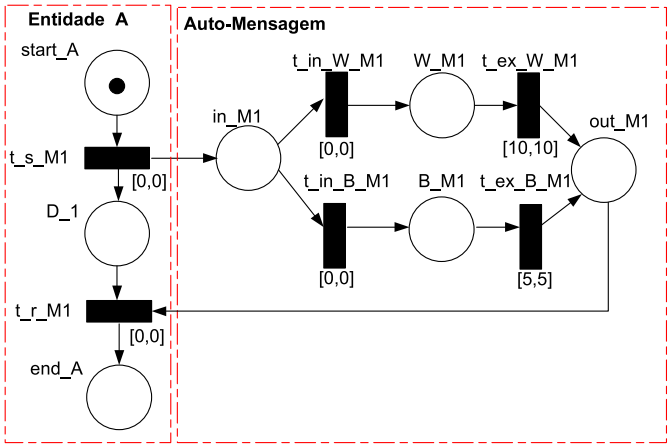


Figura 4.25: Mapeamento da auto-mensagem.

Definição 4.14. (Auto-mensagens com restrições) O modelo da auto-mensagem com restrições é uma ETPN definida como AMR= ($P^{AMR}, T^{AMR}, F^{AMR}, W^{AMR}, {m_0}^{AMR}$, $I^{AMR}, \mathcal{E}^{AMR}$), onde:

- $P^{AMR} = \{start\_A, in\_M1, W\_M1, B\_M1, out\_M1, D\_1, end\_A\}$;
- $T^{AMR} = \{t\_s\_M1, t\_in\_W\_M1, t\_in\_B\_M1, t\_ex\_W\_M1, t\_ex\_B\_M1, t\_r\_M1\}$;

- $F^{AMR} = \{(start\_A, t\_s\_M1), (t\_s\_M1, in\_M1), (in\_M1, t\_in\_W\_M1), (in\_M1, t\_in\_B\_M1), (t\_in\_W\_M1, W\_M1), (t\_in\_B\_M1, B\_M1), (W\_M1, t\_ex\_W\_M1), (B\_M1, t\_ex\_B\_M1), (t\_ex\_W\_M1, out\_M1), (t\_ex\_B\_M1, out\_M1), (out\_M1, t\_r\_M1), (t\_s\_M1, D\_1), (D\_1, t\_r\_M1), (t\_r\_M1, end\_A)\}$;
- $W^{AMR}(f) = 1, \forall\ f \in F^{AMR}$;
- ${m_0}^{AMR}(p) = \begin{cases} 1, & \text{se } p = start\_A; \\ 0, & \text{caso contrário.} \end{cases}$
- $I^{AMR}(t) = (EFT^{AMR}(t), LFT^{AMR}(t)), \forall t \in T^{AMR}$, onde:
  $EFT^{AMR}(t) = LFT^{AMR}(t) = \begin{cases} X_E^{AMR}, & \text{se } t = t\_ex\_W\_M1; \\ X_L^{AMR}, & \text{se } t = t\_ex\_B\_M1; \\ 0, & \text{caso contrário.} \end{cases}$
- $\mathcal{E}^{AMR}(t) = 0\ \forall t \in T^{AMR}$

onde X_E^{AMR} e X_L^{AMR} são os tempos de execução no melhor e pior casos associados às transições *t_ex_B_M1* e *t_ex_W_M1*, respectivamente.

4.3.3 Mapeamento dos Fragmentos Combinados

Os fragmentos combinados são usados para combinar um conjunto de mensagens a fim de mostrar as iterações complexas dos DS. Eles são representados por um retângulo que determina a área de abrangência do fragmento no diagrama. Além disso, o fragmento combinado possui uma subdivisão em sua extremidade superior esquerda para identificar o operador de interação (ver Figura 4.26). Os operadores de interação são usados para definir o tipo de fragmento que está sendo modelado.

4.3.3.1 Mapeamento das Alternativas O operador de iteração *alt* (alternativas) é usado para representar escolhas mutuamente exclusivas de duas ou mais seqüências de mensagens. A Figura 4.26 apresenta um exemplo, onde esse operador indica que somente uma das mensagens será executada.

A Figura 4.27 ilustra o modelo ETPN gerado pelo processo de mapeamento do fragmento combinado *alt* presente na Figura 4.26. As regras para mapear esse fragmento combinado são similares às outras mencionadas neste capítulo, a diferença, é que nesse caso, o modelo ETPN inclui um lugar (alt_M1_M2) e duas transições (t_in_M1 e t_in_M2). O lugar representa o ponto de escolha entre os operandos (*IF* e *ELSE*), como também a entrada no *alt*. Uma transição é usada para representar a entrada na *Mensagem 1* (*IF*). Por fim, a outra transição é usada para representar a entrada na *Mensagem 2* (*ELSE*).

Definição 4.15. (Alternativas com restrições) O modelo do operador *alt* é uma ETPN definida como $FCA = (P^{FCA}, T^{FCA}, F^{FCA}, W^{FCA}, {m_0}^{FCA}, I^{FCA}, \mathcal{E}^{FCA})$, onde:

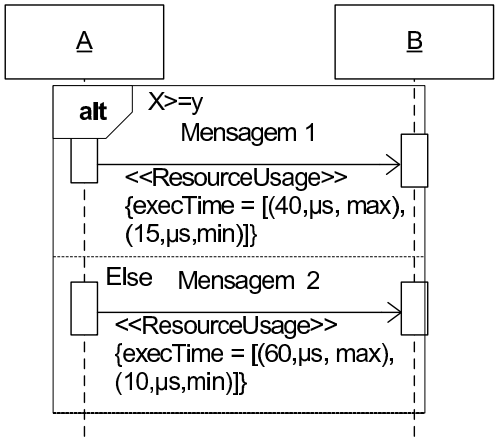


Figura 4.26: Fragmento combinado com interação alternativa.

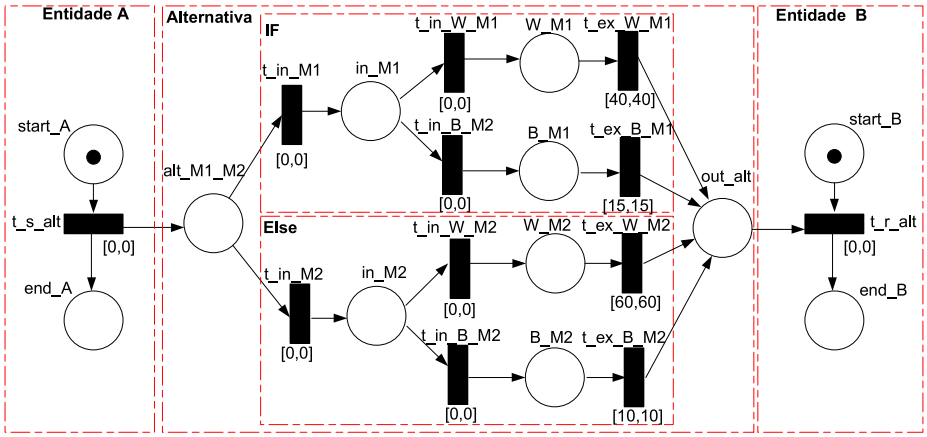


Figura 4.27: Mapeamento do fragmento combinado com interação alternativa.

- $P^{FCA} = \{start\_A, end\_A, alt\_M1\_M2, in\_M1, W\_M1, B\_M1, out\_alt, in\_M2, W\_M2, B\_M2, start\_B, end\_B\}$;
- $T^{FCA} = \{t\_s\_alt, t\_in\_M1, t\_in\_W\_M1, t\_in\_B\_M1, t\_ex\_W\_M1, t\_ex\_B\_M1, t\_in\_M2, t\_in\_W\_M2, t\_in\_B\_M2, t\_ex\_W\_M2, t\_ex\_B\_M2, t\_r\_alt\}$;
- $F^{FCA} = \{(start\_A, t\_s\_alt), (t\_s\_alt, end\_A)(t\_s\_alt, alt\_M1\_M2), (alt\_M1\_M2, t\_in\_M1), (t\_in\_M1, in\_M1), (in\_M1, t\_in\_W\_M1), (in\_M1, t\_in\_B\_M1), (t\_in\_W\_M1, W\_M1), (t\_in\_B\_M1, B\_M1), (W\_M1, t\_ex\_W\_M1), (B\_M1, t\_ex\_B\_M1), (t\_ex\_W\_M1, out\_alt), (t\_ex\_B\_M1, out\_alt), (alt\_M1\_M2, t\_in\_M2), (t\_in\_M2, in\_M2), (in\_M2, t\_in\_W\_M2), (in\_M2, t\_in\_B\_M2), (t\_in\_W\_M2, W\_M2), (t\_in\_B\_M2, B\_M2), (W\_M2, t\_ex\_W\_M2), (B\_M2, t\_ex\_B\_M2), (t\_ex\_W\_M2, out\_alt), (t\_ex\_B\_M2, out\_alt), (out\_alt, t\_r\_alt), (start\_B, t\_r\_alt), (t\_r\_alt, end\_B)\}$;
- $W^{FCA}(f) = 1, \forall\ f \in F^{FCA}$;

- ${m_0}^{FCA}(p) = \begin{cases} 1, & \text{se } p = start\_A; \\ 1, & \text{se } p = start\_B; \\ 0, & \text{caso contrário.} \end{cases}$
- $I^{FCA}(t) = (EFT^{FCA}(t), LFT^{FCA}(t)), \forall t \in T^{FCA}, onde:$

$$EFT^{FCA}(t) = LFT^{FCA}(t) = \begin{cases} X_{E_1}^{FCA}, & \text{se } t = t\_ex\_W\_M1; \\ X_{L_1}^{FCA}, & \text{se } t = t\_ex\_B\_M1; \\ X_{E_2}^{FCA}, & \text{se } t = t\_ex\_W\_M2; \\ X_{L_2}^{FCA}, & \text{se } t = t\_ex\_B\_M2; \\ 0, & \text{caso contrário.} \end{cases}$$

- $\mathcal{E}^{FCA}(t) = 0 \ \forall t \in T^{FCA}$

onde $X_{E_1}^{FCA}$ e $X_{L_1}^{FCA}$ são os tempos de execução no melhor e pior caso associados às transições *t_ex_B_M1* e *t_ex_W_M1*, respectivamente, e $X_{E_1}^{FCA}$ e $X_{L_1}^{FCA}$ também são os tempos de execução no melhor e pior caso, no entanto, associados às transições *t_ex_B_M2* e *t_ex_W_M2*, respectivamente.

4.3.3.2 Mapeamento da paralela O operador de iteração *par* (paralela) é usado para definir que duas ou mais mensagens podem ser executas paralelamente. A Figura 4.28 apresenta um exemplo, onde as *Mensagens 1 e 2* podem ser executadas ao mesmo tempo.

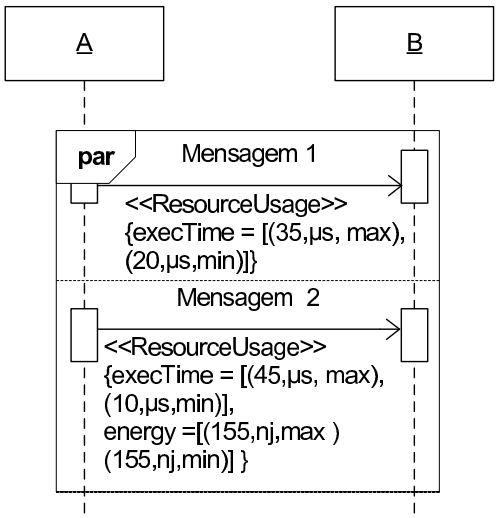


Figura 4.28: Fragmento combinado com interações paralelas.

A seguir, na Figura 4.29, é apresentado o modelo ETPN gerado pelo processo de mapeamento do fragmento combinado com interações paralelas presente na Figura 4.28. As transições *t_in_par* e *t_syn_par* representam, respectivamente, o início do paralelismo e a sincronização das mensagens paralelas. Cada uma das regiões contém mensagens distintas e são mapeadas de acordo com as regras apresentadas anteriormente neste capítulo.

Os lugares *start_par* e *out_par* representam, respectivamente, a entrada no paralelismo e a saída do paralelismo.

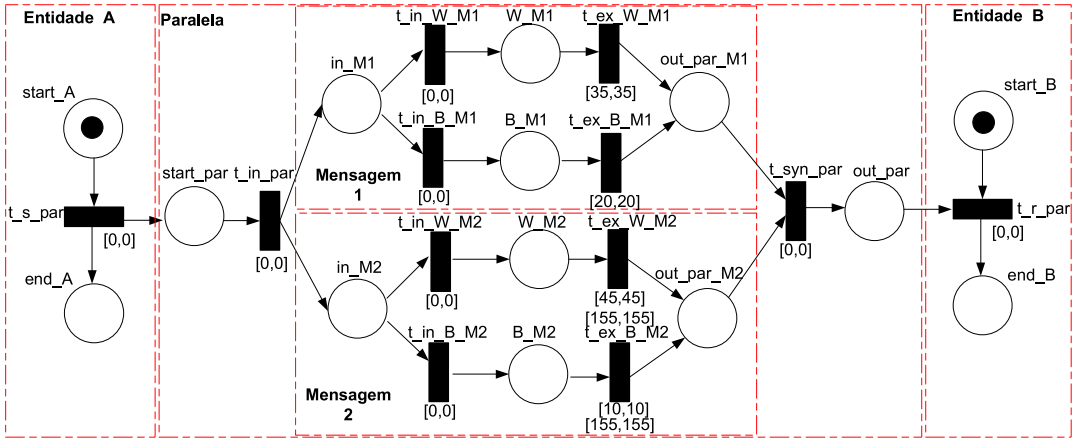


Figura 4.29: Mapeamento do fragmento combinado com interações paralelas.

Definição 4.16. (Paralelas com restrições) O modelo do fragmento combinado com interações paralelas é uma ETPN definida como $FCP = (P^{FCP}, T^{FCP}, F^{FCP}, W^{FCP}, {m_0}^{FCP}, I^{FCP}, \mathcal{E}^{FCP})$, onde:

- $P^{FCP} = \{start\_A, end\_A, start\_par, in\_M1, W\_M1, B\_M1, out\_par_M1, in\_M2, W\_M2, B\_M2, out\_par\_M2, out\_par, start\_B, end\_B\}$;
- $T^{FCP} = \{t\_s\_par, t\_in\_par, t\_in\_W\_M1, t\_in\_B\_M1, t\_ex\_W\_M1, t\_ex\_B\_M1, t\_in\_W\_M2, t\_in\_B\_M2, t\_ex\_W\_M2, t\_ex\_B\_M2, t\_syn\_par, t\_r\_par\}$;
- $F^{FCP} = \{(start\_A, t\_s\_par), (t\_s\_par, end\_A)(t\_s\_par, start\_par), (start\_par, t\_in\_par), (t\_in\_par, in\_M1), (in\_M1, t\_in\_W\_M1), (in\_M1, t\_in\_B\_M1), (t\_in\_W\_M1, W\_M1), (t\_in\_B\_M1, B\_M1), (W\_M1, t\_ex\_W\_M1), (B\_M1, t\_ex\_B\_M1), (t\_ex\_W\_M1, out\_par_M1), (t\_ex\_B\_M1, out\_par\_M1), (out\_par\_M1, t\_syn\_par), (t\_in\_par, in\_M2), (in\_M2, t\_in\_W\_M2), (in\_M2, t\_in\_B\_M2), (t\_in\_W\_M2, W\_M2), (t\_in\_B\_M2, B\_M2), (W\_M2, t\_ex\_W\_M2), (B\_M2, t\_ex\_B\_M2), (t\_ex\_W\_M2, out\_par_M2), (t\_ex\_B\_M2, out\_par\_M2), (out\_par\_M2, t\_syn\_par), (t\_syn\_par, out\_par), (out\_par, t\_r\_par)(start\_B, t\_r\_par), (t\_r\_par, end\_B)\}$;
- $W^{FCP}(f) = 1, \forall\ f \in F^{FCP}$;
- ${m_0}^{FCP}(p) = \begin{cases} 1, & \text{se } p = start\_A; \\ 1, & \text{se } p = start\_B; \\ 0, & \text{caso contrário.} \end{cases}$

- $I^{FCP}(t) = (EFT^{FCP}(t), LFT^{FCP}(t)), \forall t \in T^{FCP}, onde:$
  $$EFT^{FCP}(t) = LFT^{FCP}(t) = \begin{cases} X_{E_1}^{FCP}, & \text{se } t = t\_ex\_W\_M1; \\ X_{L_1}^{FCP}, & \text{se } t = t\_ex\_B\_M1; \\ X_{E_2}^{FCP}, & \text{se } t = t\_ex\_W\_M2; \\ X_{L_2}^{FCP}, & \text{se } t = t\_ex\_B\_M2; \\ 0, & \text{caso contrário.} \end{cases}$$
- $\mathcal{E}^{FCP}(t) = \begin{cases} Y_E^{FCP}, & \text{se } t = t\_ex\_W\_M2; \\ Y_L^{FCP}, & \text{se } t = t\_ex\_B\_M2; \\ 0, & \text{caso contário.} \end{cases}$

onde $X_{E_1}^{FCP}$ e $X_{L_1}^{FCP}$ são os tempos de execução no melhor e pior caso associados às transições *t_ex_B_M1* e *t_ex_W_M1*, respectivamente, e $X_{E_1}^{FCP}$ e $X_{L_1}^{FCP}$ também são os tempos de execução no melhor e pior casos, no entanto, associados às transições *t_ex_B_M2* e *t_ex_W_M2*, respectivamente. Por outro lado, Y_E^{FCP} e Y_L^{FCP} são os consumos de energia no melhor e pior caso associados às transições *t_ex_W_M2* e *t_ex_B_M2*, respectivamente.

4.4 CONSIDERAÇÕES FINAIS

Este capítulo apresentou o mapeamento dos diagramas comportamentais da SysML, incluindo as restrições tempo e anotações energia, em modelos ETPN. Dentre os elementos mapeados, pode-se citar: estados, transições, *lifelines*, fragmentos combinados, etc. Além disso, este capítulo introduziu o formalismo dos modelos gerados pelo processo de mapeamento, que é, antes de mais nada, uma RdP com restrições de tempo e anotações de consumo de energia.

CAPÍTULO 5

ESTUDO DE CASO

Neste capítulo são apresentados os estudos de caso realizados de forma a validar, tanto os modelos ETPN gerados pelo processo de mapeamento, quanto a metodologia de avaliação. Para tanto, todas as etapas da metodologia foram executadas, bem como algumas métricas relacionadas ao consumo de energia e ao tempo de execução foram estimadas. A fim de demonstrar a aplicabilidade da metodologia proposta, as estimativas dos modelos são comparadas com os valores medidos do *hardware*. Além disso, são apresentados os resultados da análise qualitativa.

5.1 INTRODUÇÃO

Os exemplos utilizados neste capítulo são sistemas embarcados reais, que possuem restrições de tempo e energia. Neste sentido, procura-se demonstrar a aplicabilidade do modelo ETPN gerado pelo processo de mapeamento e da metodologia desenvolvida para realizar análises e verificações dos ERTS. Dois estudos de casos são mostrados neste capítulo. O primeiro deles é um oxímetro de pulso, que é responsável por medir a saturação de oxigênio no sangue através de um técnica não-invasiva [JÓ98]. O segundo estudo de caso é uma impressora térmica. Esse tipo de impressora produz uma imagem quando a cabeça de impressão térmica passa sobre o papel.

5.2 OXÍMETRO DE PULSO

O oxímetro de pulso implementa uma técnica não-invasiva para a monitorização *in vivo* da saturação arterial de oxigênio. A saturação arterial de oxigênio é um parâmetro médico que mensura o quanto da capacidade que o sangue tem de transportar oxigênio está sendo efetivamente utilizada. Esse dispositivo pode ser aplicado principalmente em: unidades de tratamento intensivo, monitorização de pacientes portadores de doenças pulmonares crônicas, avaliação da resposta à administração de oxigênio em casos de coma, estudo do sono, avaliação de atletas, procedimentos anestesiológicos, entre outros [JÓ98].

5.2.1 Projeto de Requisitos

Esta etapa tem o objetivo de investigar e definir os requisitos do sistema. O oxímetro de pulso é dividido em três partes distintas, denominadas processos (ver Figure 5.1). Os processos são programas que executam independentes um dos outros. Cada um desses

processos é dividido em sub-atividades. Seguindo esse conceito, nomeiam-se os processos de acordo com a atividade que desempenham. Os processos são:

- Processo 1 - Rotina de Excitação;
- Processo 2 - Rotina de Amostragem e Controle; e
- Processo 3 - Programa Gerenciador.

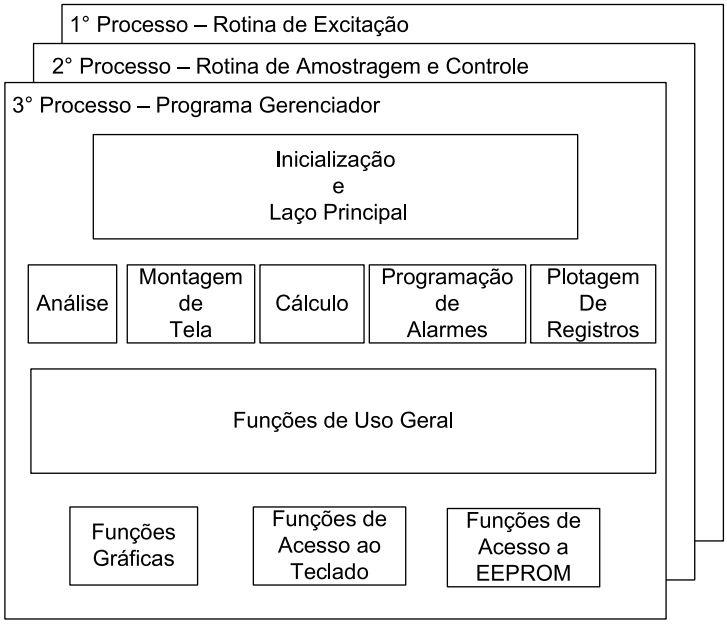


Figura 5.1: Estrutura do oxímetro.

Este trabalho adotou o processo de excitação como estudo de caso, pois esse processo é o que possui maior prioridade entre os demais [JÓ8]. O processo de excitação é responsável por produzir pulsos de corrente não simultâneos aos *leds* a fim de gerar os pulsos de radiação (ver Figure 5.2). É importante ressaltar que algumas propriedades são importantes serem encontradas nos modelos gerados pelo mapeamento. Entre as propriedades de interesse podemos ressaltar a existência ou ausência de *deadlock*, *liveness* e *reversibilidade*. Além disso, é importante verificar a comunicação e passagem de controle entre as entidades ao longo do tempo.

5.2.2 Criação dos Diagramas Comportamentais da SysML

Nesta etapa os diagramas comportamentais da SysML são criados. Os diagramas adotados para representar o processo de excitação são os de atividades e seqüência. Embora esses diagramas estejam modelando o mesmo cenário, eles são de extrema importância quando se trabalha com sistemas críticos, pois permitem avaliar os diferentes aspectos

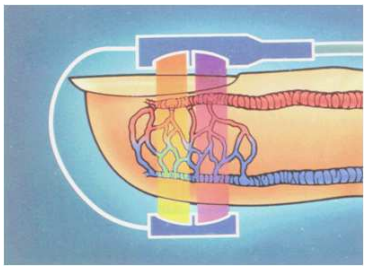


Figura 5.2: Processo de excitação.

da dinâmica do sistema, pois o diagrama atividades mostra a seqüência de etapas que compõem o processo de excitação. O diagrama seqüência enfatiza a comunicação e passagem de controle entre as entidades ao longo do tempo. Esses diagramas são apresentados, respectivamente, nas Figuras 5.3 e 5.4. Todos esses diagramas foram criados segundo os passos descritos na Seção 3.3.

A Figura 5.3 descreve o diagrama de atividades do processo de excitação. A atividade referente ao processo de excitação se inicia com o recebimento de uma interrupção no *Timer 1*. Os *timers* são usados para determinar qual processo será executado primeiro. Nas etapas subseqüentes as atividades *Set Led Infra*, *Set Led Guad* e *Set Led Red* são realizadas. Essas atividades são responsáveis por enviarem pulsos de corrente aos *leds* cuja luminosidade atravessa parte do corpo do paciente translúcida (ex.: a ponta dos dedos ou lóbulo da orelha). Subseqüentemente uma decisão é realizada, caso seja necessário modificar a intensidade da corrente que gera os pulsos vermelhos e infravermelhos então será executada a atividade *Run*, caso contrário a atividade *NotRun* será executada. Como pode ser observado na Figura 5.3, o fluxo do diagrama sempre retorna para a atividade *Timer 1*.

Na Figura 5.4, é apresentado o diagrama de seqüência do processo de excitação. Inicialmente a entidade *Timer* executa a chamada *SetLedRed()* para a entidade *LedRed* enviar pulsos de corrente para o *led* vermelho. Subseqüentemente, a entidade *LedRed* executa a chamada *SetGuard()* para a entidade *Guard*. Essa chamada *SetGuard()* inclui um atraso no processo de excitação para garantir a distinção entre os pulsos vermelhos e infravermelhos. Em seguida a entidade *Guard* executa a chamada *SetLedInfra()* para a entidade *LedInfra* enviar pulsos de corrente para o *led* infravermelho. Após isso, como pode ser observado na figura, o operador de iteração *alt* (alternativas) é usado para representar a escolha mutuamente exclusiva de duas chamadas, um acima (*CalibrationNotRun()*) e um abaixo (*CalibrationRun()*) da linha tracejada. No entanto, apenas uma das alternativas oferecida será executada em qualquer passagem pela interação. Em outras palavras, caso seja necessário modificar a intensidade da corrente que gera os pulsos vermelhos e infravermelhos então será executada a chamada *CalibrationRun()*, caso contrário a chamada *CalibrationNotRun()* será executada.

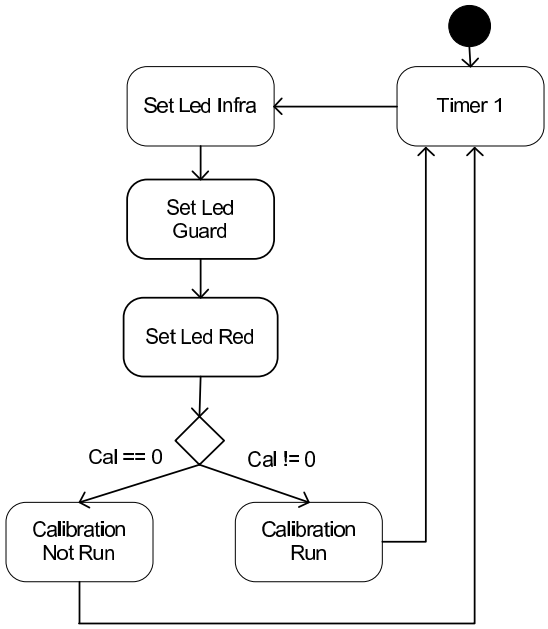


Figura 5.3: Diagramas de atividades do processo de excitação.

5.2.3 Atribuição de Restrições aos Diagramas usando MARTE

Uma vez criados os diagramas de atividades e seqüência, as restrições de tempo e anotações de energia são especificadas através de MARTE.

A Figura 5.5 apresenta o diagrama de atividades do processo de excitação com suas respectivas restrições de tempo e anotações relativas ao consumo de energia. Essas restrições e anotações são relacionadas à plataforma de *hardware* (processador philips LPC2106, um microcontrolador 32-bit com núcleo ARM7). Por exemplo, a atividade *Set Led Infra* possui restrições de tempo e anotações de energia, isto é, essa atividade é especificada com um *<<ResourceUsage>>* de *execTime = [(14,69,μs, max), (14,11,μs,min)]* e *energy = [(835,99,ηJ,max), (795,21,ηJ,min)]*. As restrições dos tempos de execução significam que a atividade *Set Led Infra* possui o tempo de execução no pior caso de 14,69 μs e no melhor caso de 14,11 μs, respectivamente. A atividade *Set Led Infra* também possui um consumo de energia no pior caso de 835,99 ηJ e no melhor caso de 795,21 ηJ.

De forma semelhante, a Figura 5.6 apresenta o diagrama de seqüência do processo de excitação com suas respectivas restrições de tempo e anotações de energia. Essas restrições e anotações também são relacionadas à plataforma de *hardware* (processador philips LPC2106). Por exemplo, a chamada *SetGuard()* executada pela entidade *LedRed* possui restrições de tempo e anotações de consumo de energia. Neste caso, *SetGuard()* é especificado com um *<<ResourceUsage>>* de *execTime = [(7,75, μs, min), (7,45, μs, max)]* e *energy = [(437,88 ηJ,min), (416,52,ηJ,max)]*.

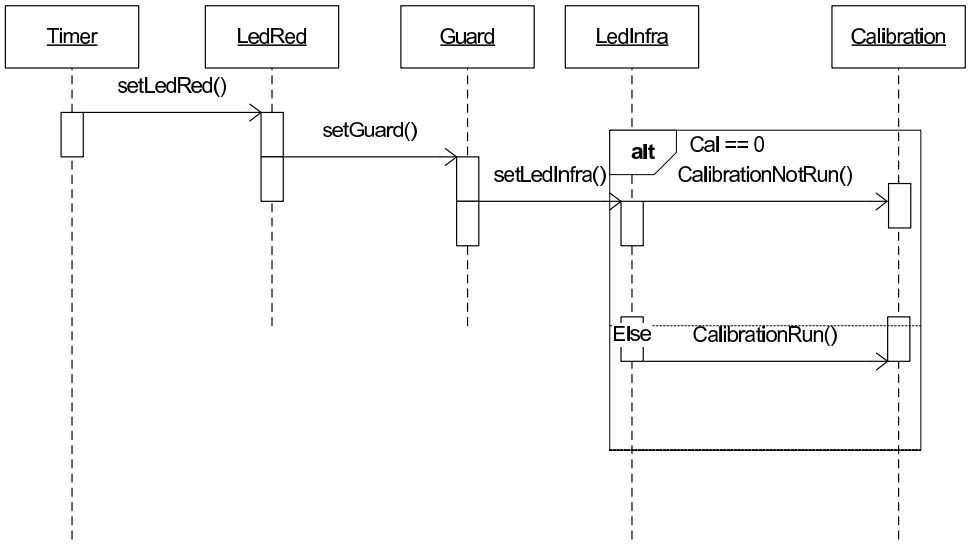


Figura 5.4: Diagrama de seqüência do processo de excitação.

5.2.4 Mapeamento dos Diagramas Comportamentais da SysML em uma ETPN

Para possibilitar a integração dos modelos semiformais e formais, o mapeamento tanto do diagrama de atividades quanto do diagrama de seqüência em um modelo ETPN é realizado. A seguir, o processo de mapeamento é brevemente detalhado.

Este procedimento foi adotado para a obtenção do modelo ETPN da Figura 5.7. Cada atividade é representada, basicamente, por dois lugares e duas transições. Um lugar é usado para representar a entrada na atividade, e o outro lugar é usado para representar a saída da atividade. As transições, por sua vez, são usadas para representar as restrições de tempo e anotações de energia relacionadas às atividades. Adicionalmente, duas transições-PN são adotadas no modelo ETPN devido à semântica das ETPN (*Strong Firing Semantics*). O momento a partir do qual as atividades serão analisadas é representado por um estado inicial. O modelo ETPN correspondente ao estado inicial é um lugar com a marcação inicial igual a um *token*. Após todos os modelos individuais terem sido construídos, então as transições do diagrama de atividade são mapeadas em transições-PN no modelo ETPN, levando em consideração as restrições de tempo e anotações de energia. No entanto, se as restrições de tempo forem omitidas das atividades, então essas atividades são mapeados em transições-PN, nas quais o tempo máximo e mínimo é zero.

De modo semelhante, a fim de obter o modelo ETPN (ver Figura 5.8) que representa o cenário presente na Figura 5.6, o seguinte procedimento foi adotado. Cada entidade é representado por pelo menos dois lugares e uma transição no modelo ETPN. Um lugar com a marcação inicial igual a um *token* é usado para representar o começo da linha de vida, e o outro lugar representa o fim da linha de vida. A transição representa o envio/recebimento de uma mensagem/chamada. Após todos os modelos ETPN, que representam as entidades terem sido construídos, as mensagens/chamadas são mapeadas em transições-PN no modelo ETPN, levando em consideração as restrições de tempo e

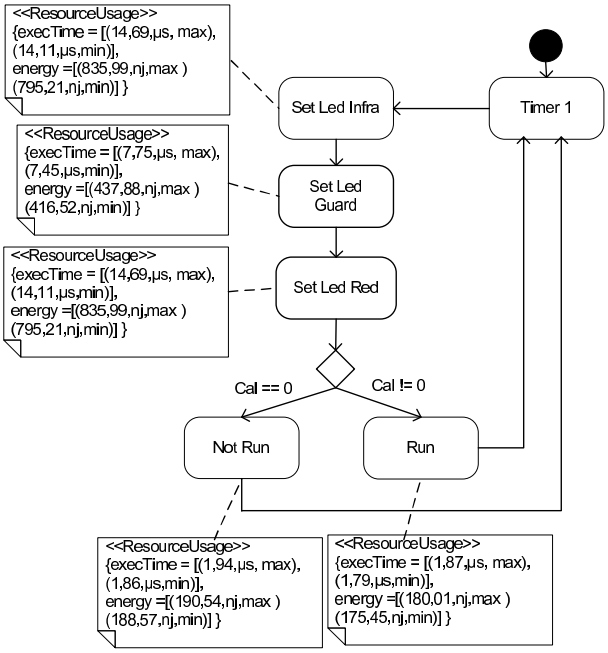


Figura 5.5: Diagramas de atividades com restrições de tempo e anotações de energia.

anotações de energia. Mensagens/chamadas que não possuem restrições de tempo são mapeadas em lugares *dummy*, os quais são usados para interligar as transições-PN.

5.2.5 Análises e Verificações

A quinta etapa da metodologia consiste na realização de análises e verificações dos modelos ETPN. Após analisar o modelo ETPN da Figura 5.7, conclui-se que o processo de excitação possui as seguintes propriedades:

- **Deadlock Freedom.** A análise do processo de excitação indica que o sistema não possui *deadlock*. Em outras palavras, as atividades do processo de excitação não ficam impedidas de continuar suas execuções à espera de um evento que uma outra atividade possa executar.

- **Reversibilidade.** Em sistemas computacionais muitas aplicações têm a propriedade do retorno ao estado inicial ou mesmo a algum grupo de estados. O processo de excitação não é reversível para sua marcação inicial, porém possui *home state*, pois esse processo não pode retornar a seu estado inicial após sua inicialização. O estado inicial só poderá ser alcançado quando o processo excitação for inicializado ou reiniciado. No entanto, existe um *home state*, ou seja, uma marcação (atividade) que pode ser alcançada novamente pelo disparo de uma seqüência de transições (eventos).

- **Liveness.** Uma rede é dita *live* se, não importa quais marcações sejam alcançáveis

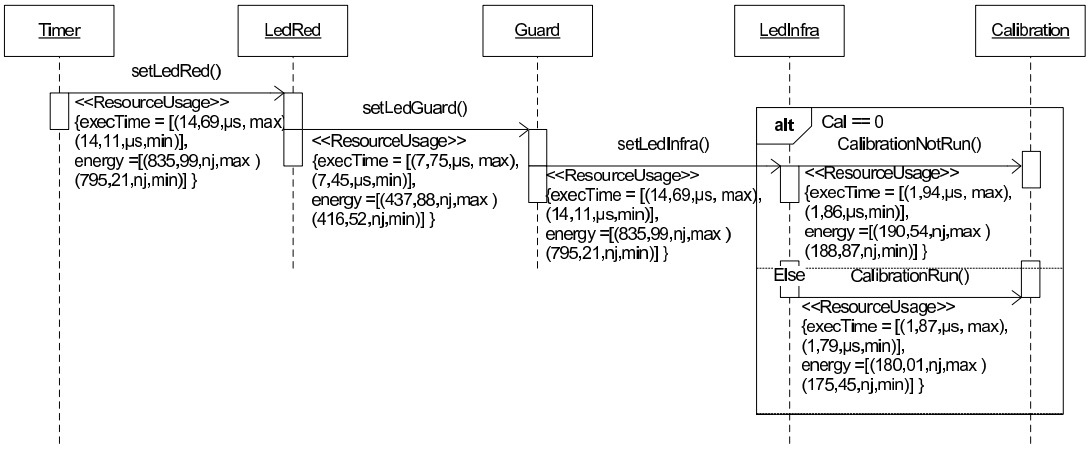


Figura 5.6: Diagrama de seqüência com restrições de tempo e anotações de energia.

a partir de um marcação inicial m_0, for possível disparar qualquer transição através do disparo de alguma seqüência de transições $L(m_0)$. O modelo do processo de excitação não é *live*, pois existem situações, onde as transições são executadas apenas uma vez, isto é, a especificação do processo de excitação só irá executar todos seus eventos quando for inicializado ou reiniciado. No entanto, o modelo pode ser considerado do tipo *L1-live* (ver Seção 2.5).

O modelo ETPN da Figura 5.8, que representa o diagrama de seqüencia, foi utilizado para simular a comunicação e passagem de controle entre as entidades ao longo do tempo. A avaliação das transições habilitadas e seus respectivos disparos (simulação da rede) é também chamada de *token game*. A simulação é uma ferramenta importante para estudo dos sistemas pois, através dela, é possível compreender os vários aspectos do comportamento dos ERTS. Dessa forma, foi realizado o *token game* para verificar se o processo modelado respeita a ordem temporal da execução das chamadas.

Além disso, os modelos ETPN também foram adotados para análises quantitativas. O tempo de execução no melhor caso (BCET) calculado foi 37,46 μs e no pior caso (WCET) foi 39,07 μs. É importante salientar que esses valores calculados são para uma única execução. Além disso, os *traces* encontrados para o pior e melhor tempo de execução, foram adotados para calcular o consumo de energia. Os resultados obtidos foram 2182,39 ηJ e 2300,4 ηJ, respectivamente. A ferramenta INA [SR99] foi adotada neste trabalho, como já mencionado anteriormente, para calcular o melhor e pior tempo de execução. Uma vez encontrados os *traces* dos tempos de execução (BCET e WCET), então o consumo de energia é calculado. A Tabela 5.2 resume as estimativas encontradas. As estimativas do BCET e WCET são importantíssimas para os sistemas de tempo-real crítico, pois essas estimativas são usadas para fornecer análises de escalonamento e correção temporal.

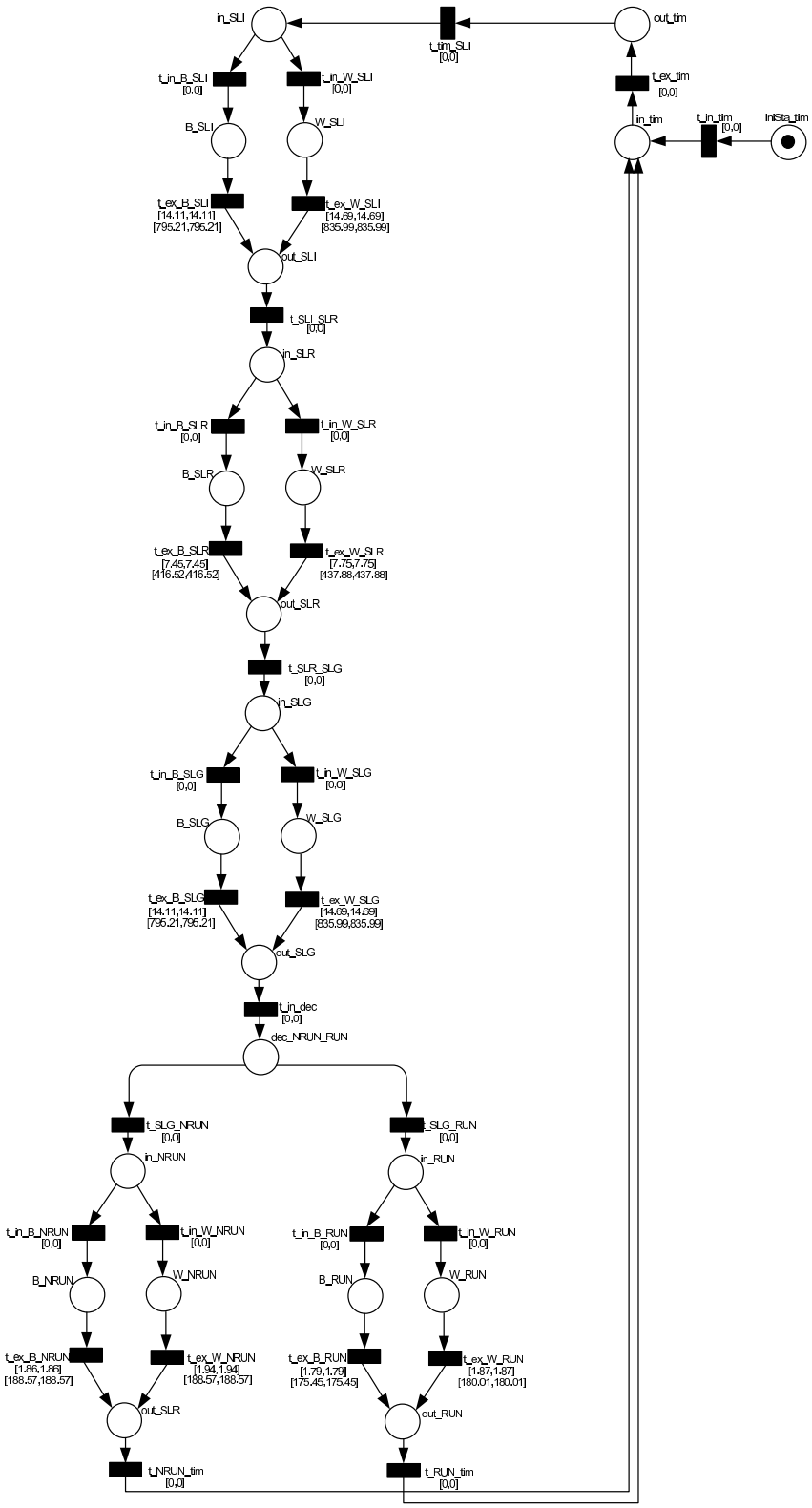


Figura 5.7: Modelo ETPN do diagrama de atividade com restrições de tempo e anotações de energia.

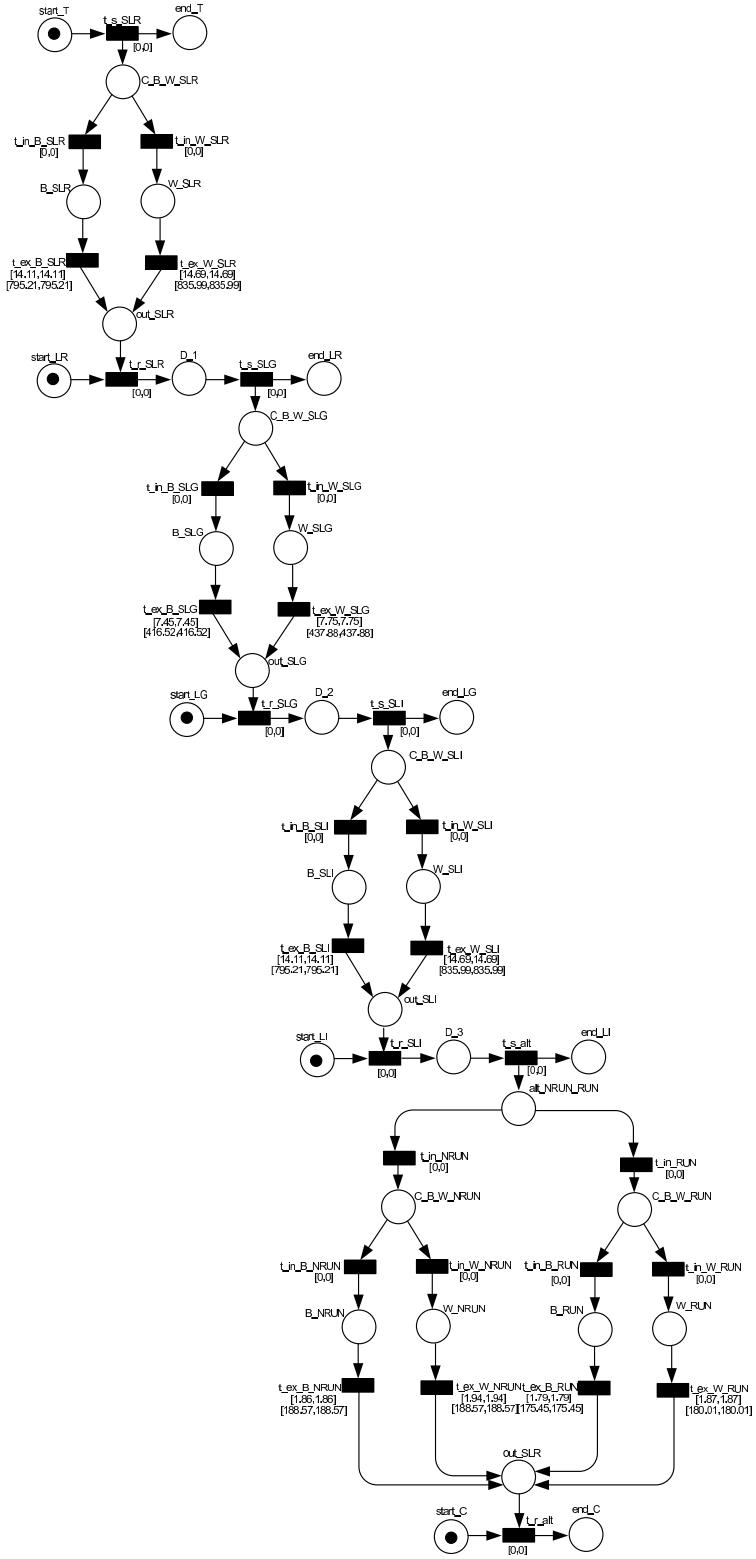


Figura 5.8: Modelo ETPN do diagrama de seqüência com restrições de tempo e anotações de energia.

Tabela 5.1: Estimativas do processo de excitação.

Métrica	Resultado
BCET	37,46 μs
WCET	39,07 μs
ECBC	2182,39 ηJ
ECWC	2300,4 ηJ

5.2.6 Validação

Esta seção, embora não faça parte da metodologia, consiste na validação dos resultados obtidos nas etapas anteriores com o intuito de demonstrar aplicabilidade do método proposto. Neste trabalho os valores estimados foram comparados com os respectivos valores medidos da plataforma de *hardware*.

O processador considerado neste trabalho é o LPC2106, que permite o desenvolvimento de aplicações de alto desempenho e baixo consumo de energia. A escolha desse processador se deu em função de sua grande utilização em aplicações de tempo-real em que o consumo de energia é uma questão central.

A Figure 5.9 apresenta um esquema que ilustra o processo de medição de energia e tempo de execução das aplicações. Este mesmo esquema é utilizado para caracterização dos processadores. A caracterização é realizada por meio da plataforma de medição AMALGHMA (ver Figura 5.10) [TM06]. O AMALGHMA captura os dados adquiridos pelo osciloscópio e faz um tratamento estatístico dos dados para produção de medidas confiáveis. Para medir o consumo de energia de um código, por exemplo, um computador é utilizado para controlar um osciloscópio digital (Agilent DS0302A). São utilizados dois canais do osciloscópio, um para a medição da tensão e o outro como *trigger* na porta de entrada/saída de comando de início e fim da medição. A tensão é medida em resistor colocado em série com o processador. Este processo reproduz basicamente as técnicas apresentadas em [TMW94, RJ98].

Para que utilizemos adequadamente o AMALGHMA, o código a ser medido deve ser inserido no corpo de um laço. No início do laço o nível de tensão de um pino de uma porta de entrada/saída do microprocessador deve ser alterado para o seu valor máximo e ao final do laço o nível de tensão deve ser alterado para o seu valor mínimo. A Figura 5.11 exibe a tela do osciloscópio durante a medição. Quando o sinal do pino da porta de entrada/saída vai para o nível alto, AMALGHMA detecta o início da execução do código de interesse. Quando o nível de tensão do respectivo pino é alterado para o valor mínimo AMALGHMA detecta o fim da execução do código de interesse. Dessa forma, além do consumo de energia, é possível também mensurar o tempo de execução do código.

Para medir o consumo de energia e tempo de execução de uma instrução o mesmo procedimento descrito acima é realizado. Cada instrução é medida individualmente. Para

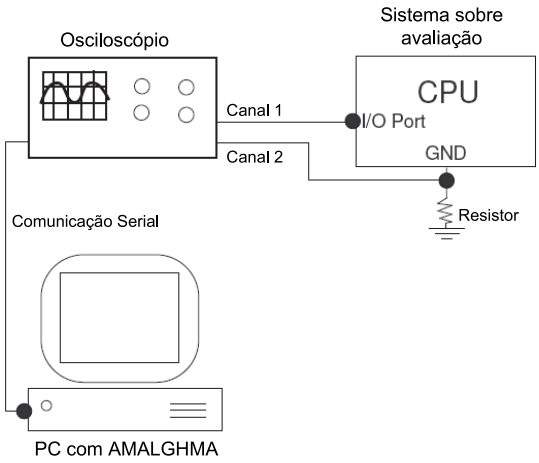


Figura 5.9: Esquema de medição.

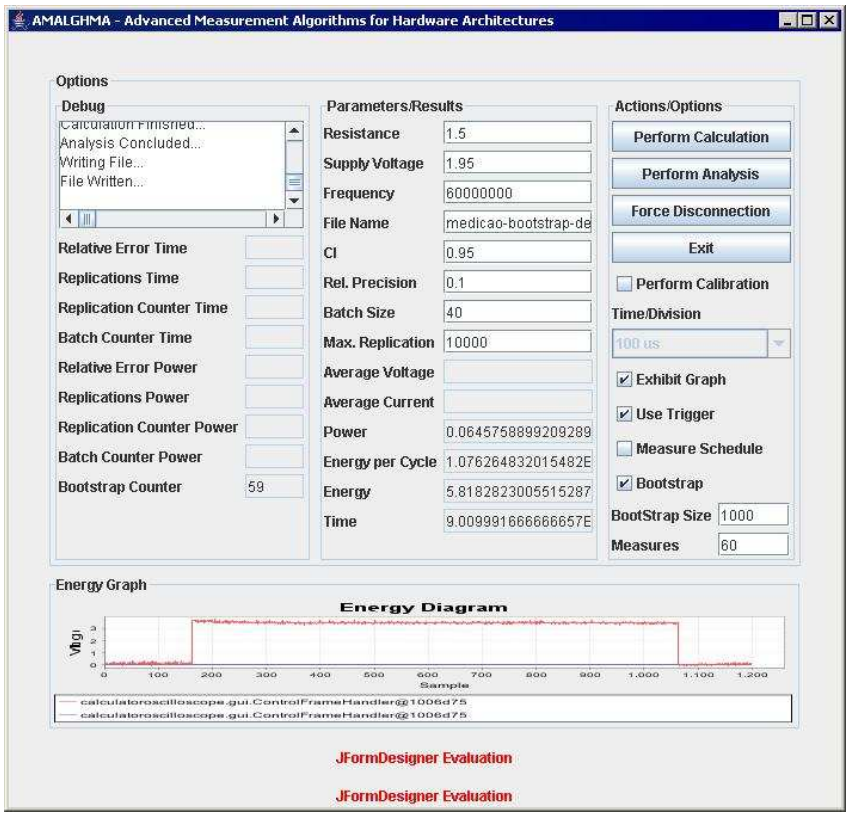


Figura 5.10: AMALGHMA.

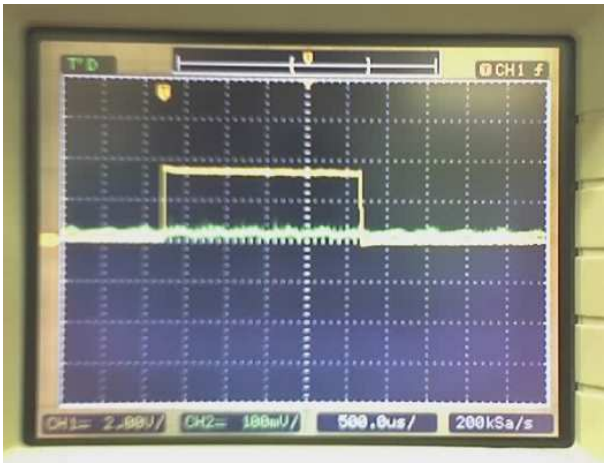


Figura 5.11: Tela do Agilent DS0302A durante o processo de medição.

```
1
2 while(1){
3 int i;
4 IOSET = IOPIN | 0X00000080; /* levanta pino */
5
6 /* início do código */
7 __asm{
8 mov r1, #0
9 mov r1, #0
10 ... (10000 instâncias das instruções)
11 mov r1, #0
12 mov r1, #0
13 }
14 /* fim do código */
15
16 IOCLR = (~IOPIN) | 0X00000080 /* desce pino */
17 for (i=0; i<5300; i++); /* guarda */
18 }
```

Figura 5.12: Exemplo de código para medição.

eliminar o efeito do *pipeline* e do código de controle (que altera o nível de tensão do sinal do pino monitorado) sobre as medidas, o corpo do laço é formado por 10000 instâncias da instrução a ser medida. Ao final do processo de medição as medidas são divididas por 10000 para obter os valores individuais relativos a uma instrução. A Figura 5.12 exibe um exemplo de código de caracterização. É importante ressaltar que os dados relativos ao tempo de execução de cada instrução foram validados através da comparação desses valores com os disponíveis no *datasheet* [Lim01] de especificação do ARM7TDMI-S.

Os valores medidos na plataforma de *hardware* (tempo de execução e consumo de energia) para o processo de excitação foram 38,88 μs e 2251,84 ηJ, respectivamente. A análise dos resultados mostra que o erro calculado para o tempo de execução e o consumo de energia dos modelos foram menores que 5% em relação ao medidos na plataforma de *hardware*. As Figuras 5.13 e 5.14 ilustram esta comparação entre valores medidos e os calculados através dos modelos. Como pode ser observado nas figuras, os resultados experimentais mostram que os valores computados dos modelos são muito próximos dos valores reais medidos do *hardware* para o processo de excitação.

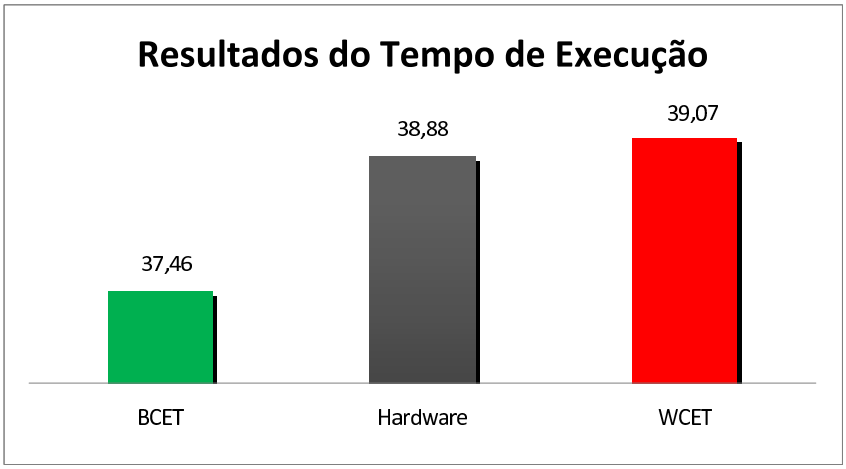


Figura 5.13: Comparação dos resultados do tempo de execução.

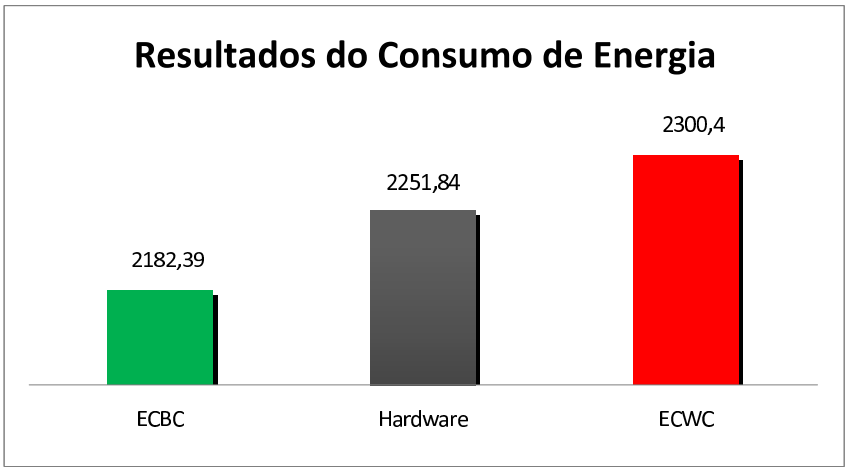


Figura 5.14: Comparação dos resultados do consumo de energia.

5.3 IMPRESSORA TÉRMICA

A impressora térmica produz uma imagem impressa aquecendo seletivamente o papel por meio de uma cabeça de impressão térmica controlada por um microprocessador. Na impressão térmica direta o papel utilizado é um papel termo-sensível ao calor. Essas impressoras são bastante usadas atualmente, pois são rápidas, silenciosas, econômicas e compactas. Além disso, têm sido utilizadas com as mais diversas finalidades, desde imprimir recibos de caixas eletrônicos ou máquinas de cartão de crédito até criar etiquetas ou códigos de barras. No entanto, a grande vantagem dessa impressora é que o único material de consumo é o papel. A Figura 5.15 apresenta dois exemplos de impressoras térmicas.

Uma impressora térmica é constituída por diversas tarefas encarregadas de monitorar sensores e botões, controlar os *leds*, a cabeça de impressão e o motor de passo, entre outras atividades. A Figura 5.16 apresenta um esquemático simplificado dos processos

Figura 5.15: Impressoras térmicas

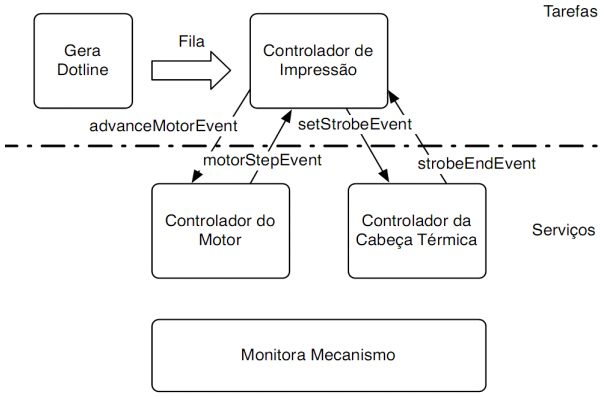


Figura 5.16: Componentes básicos de uma impressora.

de uma impressora e suas interações. No entanto, a tarefa do controlador de impressão será adotada neste estudo de caso. Essa tarefa é a principal dentre as demais, pois é encarregada de disparar o processo que controla o motor de avanço do papel e, ao seu tempo, liberar o processo que controla o acionamento da cabeça térmica, caso existam *dotlines*[1] para serem impressos. Adicionalmente, análise de um conjunto de propriedades qualitativas é de essencial importância para o desenvolvimento destes produtos dado que estes não podem violar algumas características primordiais. Dentre estas propriedades, podemos ressaltar a existência ou ausência de *deadlocks*, *liveness* e *reversibilidade*.

O diagrama de estados do controlador de impressão é apresentada na Figura 5.17. Caso existam dados para serem impressos, o controlador de impressão envia esses *dotlines* para o controlador da SPI (*Serial Peripheral Interface*), aguardando uma sinalização de retorno quando todos os dados forem recebidos. Concluída esta etapa, o controlador do motor será acionado. Logo que o motor avançar a quantidade de passos programados, seu controlador informará ao controlador de impressão esta ocorrência. Neste momento, a cabeça térmica está pronta para imprimir os dados no papel. Para que isso aconteça,

[1]São as linhas de impressão que juntas formam as frases ou imagens, ou seja, são as unidades básicas de impressão.

um sinal de *strobe*[2] é enviado ao controlador da cabeça, o qual informará ao controlador de impressão quando o este sinal for desativado. Caso ainda existam *dotlines* prontos, o processo será repetido, caso contrário, o motor será desligado aguardando que o controlador de impressão receba novamente dados de impressão, recomeçando todo o ciclo. A Figura 5.18 também ilustra o diagrama de estados do controlador de impressão, no entanto, com suas respectivas restrições de tempo e anotações de energia. Essas restrições e anotações são relacionadas à plataforma de *hardware* (processador philips LPC2106).

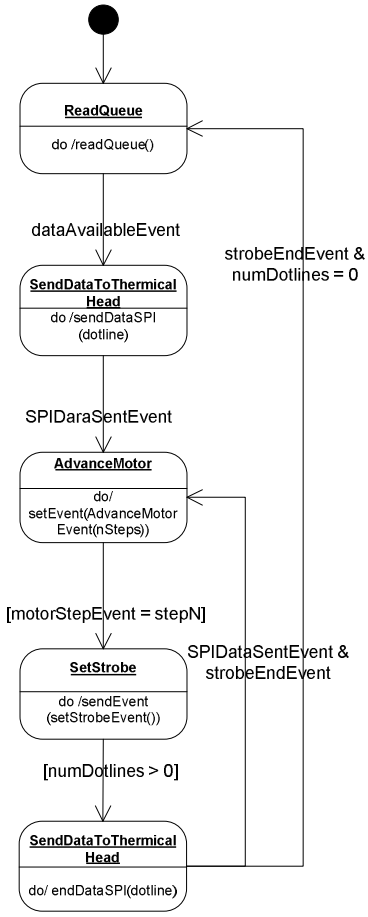


Figura 5.17: Controlador de Impressão.

Uma vez modelado o diagrama e especificado as restrições através de MARTE, então é realizado o mapeamento em um modelo ETPN. O processo para a obtenção do modelo ETPN da Figura 5.19 é semelhante ao apresentado anteriormente para o diagrama de atividades, onde cada estado simples é representado, basicamente, por dois lugares e duas transições. Os lugares são usados para representar tanto a entrada quanto a saída do estado simples e as transições são usadas para representar as restrições de tempo e anotações de energia relacionadas às atividades e transições internas do estado simples. O momento a partir do qual as atividades serão analisadas é representado por um estado inicial. O

[2]Sinal responsável por informar quando as informações podem ser carregadas na cabeça de impressão.

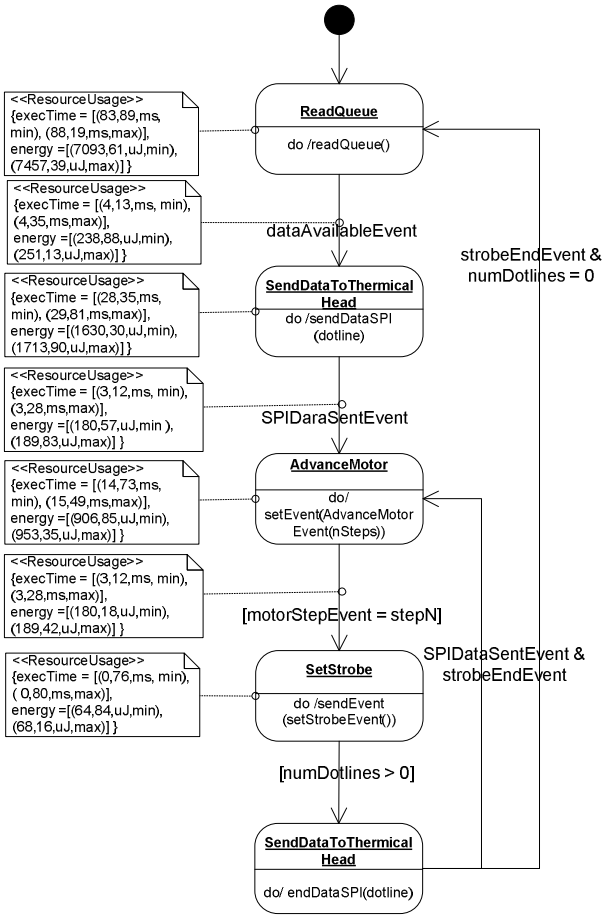


Figura 5.18: Diagramas de estados do controlador de impressão com restrições de tempo e anotações de energia.

modelo ETPN correspondente ao estado inicial é um lugar com a marcação inicial de um *token*. Após todos os modelos individuais terem sido construídos, então as transições-SM são mapeadas em transições-PN no modelo ETPN, levando em consideração as restrições de tempo e energia. Porém, se as restrições de tempo forem omitidas dos estados simples, então esses estados são mapeados em transições-PN, nas quais o tempo máximo e mínimo é zero. Adicionalmente, duas transições-PN são adotadas no modelo ETPN devido à semântica das ETPN (*Strong Firing Semantics*).

Após analisar o modelo ETPN da Figura 5.19, conclui-se que a especificação do controlador de impressão possui as seguintes propriedades:

- **Deadlock Freedom.** A análise do controlador de impressão indica que o diagrama do controlador de impressão não possui *deadlocks*, ou seja, os estados do controlador de impressão não ficam impedidos de continuar suas execuções à espera de um evento que um outro estado possa executar.

- **Reversibilidade.** O controlador de impressão não é reversível para sua marcação inicial, porém possui *home states*. Pois o controlador não pode retornar a seu estado

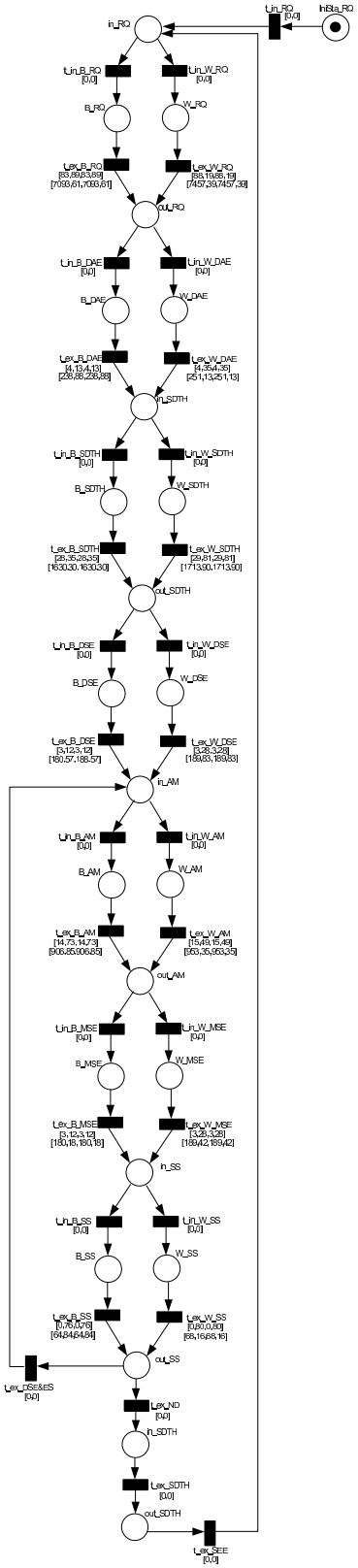


Figura 5.19: Modelo ETPN do controlador de impressão com restrições de tempo e anotações de energia.

inicial após sua inicialização. O estado inicial só poderá ser alcançado quando o controlador for inicializado ou reiniciado.

- **Liveness.** O modelo do controlador de impressão não é *live*, pois existem situações, onde as transições são executadas apenas uma vez, isto é, o controlador de impressão só irá executar todos seus eventos quando for inicializado ou reiniciado. No entanto, o modelo pode ser considerado do tipo *L1-live* (ver Seção 2.5).

O modelo ETPN gerado pelo processo de mapeamento da Figura 5.19 também foi adotado para análises quantitativas. O tempo de execução no melhor caso calculado foi 138,11 ms e o no pior caso foi 145,19 ms. Os resultados obtidos dos *traces* para o consumo de energia foram 10295,22 μJ e 10823,18 μJ, respectivamente. A Tabela 5.2 resume as estimativas encontradas. Além disso, os valores medidos na plataforma de *hardware* (tempo de execução e consumo de energia) para o controlador de impressão foram 141,65 ms e 10559,2 μJ, respectivamente. De forma semelhante ao caso de estudo anterior, a análise dos resultados mostra que o erro calculado para o tempo de execução e o consumo de energia dos modelos foram menores que 5% em relação ao medidos na plataforma de *hardware*. As Figuras 5.20 e 5.21 apresentam esta comparação entre valores medidos e os calculados através dos modelos.

Tabela 5.2: Estimativas do controlador de impressão.

Métrica	Resultado
BCET	138,11 ms
WCET	145,19 ms
ECBC	10295,22 μJ
ECWC	10823,18 μJ

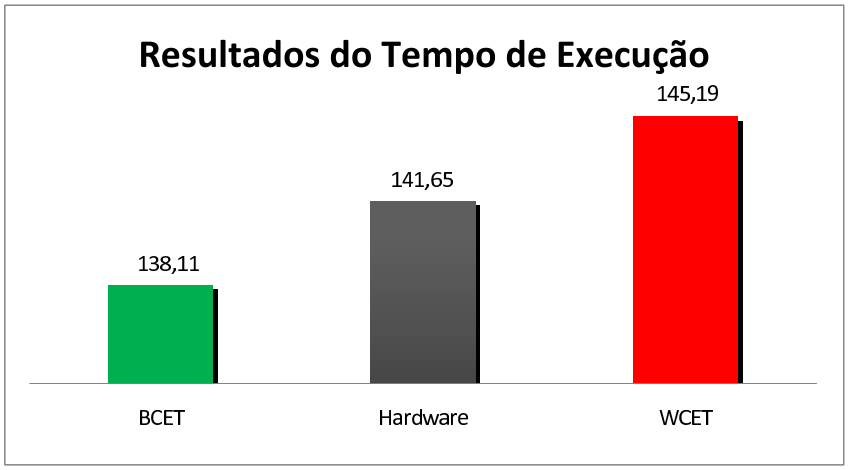


Figura 5.20: Comparação dos resultados do tempo de execução.

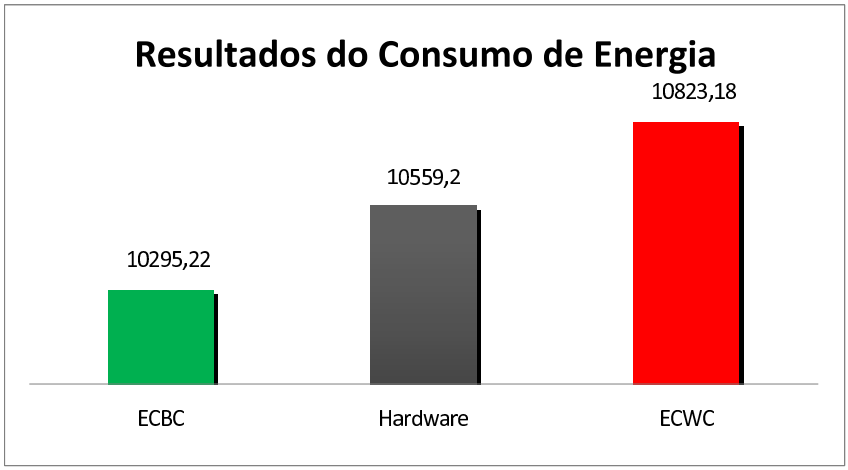


Figura 5.21: Comparação dos resultados do consumo de energia.

SysML é uma linguagem de especificação de fácil utilização que suporta a especificação, análise e *design* de uma grande variedade de sistemas complexos. Assim, se as vantagens da SysML forem aliadas ao poder dos modelos formais, então interpretações equivocadas podem ser evitadas, permitindo tanto a redução dos riscos da propagação dos erros da especificação para o código final, quanto análises e verificações das propriedades dos sistemas. Portanto, a integração dos modelos formais e semiformais pode ser usada para reduzir os ricos, como também recursos financeiros e esforço demandados para na construção de projetos embarcados.

5.4 CONSIDERAÇÕES FINAIS

Este capítulo apresentou os resultados obtidos na realização dos estudos de caso (oxímetro pulso e impressora térmica). Através dos estudos, aqui apresentado, foi possível não somente validar o modelo proposto por este trabalho através da composição dos modelos ETPN básicos, como também foi possível validar a metodologia de avaliação. Os resultados dos experimentos mostraram uma precisão de 95% utilizando a metodologia proposta em comparação aos valores medidos do *hardware*. Além disso, foram apresentados os resultados obtidos relativos a análise qualitativa.

CAPÍTULO 6

CONCLUSÕES

Com o aumento da complexidade e diversidade dos sistemas embarcados de tempo-real críticos, aliado à pressão mercadológica, onde se tem como objetivo o máximo de produtividade ao menor tempo e custo possível, surge cada vez mais a necessidade da análise desses sistemas, incluindo suas restrições ainda nas fases iniciais de projeto, visto que erros podem ser catastróficos e resultar em perdas de vidas ou de grandes quantias de recursos financeiros. Assim, a adoção de métodos formais na análise de requisitos é uma tarefa fundamental nos dias atuais, pois os modelos formais são matematicamente fundamentados, e permitem que verificações e análises tanto qualitativas quanto quantitativas possam ser realizadas. Além disso, modelos formais podem garantir ainda fase de especificação que os aspectos críticos (ex.: tempo ou energia) sejam garantidos pela captura precisa das restrições do sistema através de simulações.

Este trabalho apresentou uma abordagem baseada em ETPN para estimação (tempo de execução e consumo de energia) e verificações de propriedades nas fases iniciais do ciclo de desenvolvimento dos ERTS. O método proposto consiste, primeiramente, na derivação dos elementos básicos de cada diagrama (estados, transições, *lifeline*, mensagens, chamadas, fragmentos combinados, estado inicial, estado final etc.) em modelos ETPN individuais. Após isso, esses modelos ETPN são compostas em um modelo único que representa o respectivo diagrama. As anotações quantitativas do *profile* MARTE, tais como tempo e energia, são levadas em consideração e inclusas no modelo ETPN. Por fim, os modelos obtidos que representam o comportamento dos diagramas são adotados para realizar análises e verificações.

Com o intuito de validar tanto os modelos ETPN gerados pelo processo de mapeamento, quanto a metodologia de avaliação, foram realizado dois estudo de casos. O primeiro deles é um oxímetro de pulso, que é responsável por medir a saturação de oxigênio no sangue através de um técnica não-invasiva. O segundo estudo de caso é uma impressora térmica. Esse tipo de impressora produz uma imagem quando a cabeça de impressão térmica passa sobre o papel. Os resultados dos experimentos mostram uma precisão de 95% utilizando a metodologia proposta em comparação aos valores reais medidos do *hardware*. Esses resultados mostram que a abordagem é muito promissora para modelagem, análise e verificação dos sistemas de tempo-real com restrições de energia.

6.1 CONTRIBUIÇÕES

As contribuições deste trabalho são as seguintes:

- a realização de análises/verificações nas fases iniciais do desenvolvimento dos ERTS

com o intuito de encontrar erros ou inconsistências dos requisitos. É importante ressaltar que o custo da detecção de um erro após a entrega do sistema é, no mínimo, 100 vezes maior do que se ele tivesse sido detectado em tempo de definição do sistema [DRP99a];

- a integração dos modelos formais e semiformais. Essa integração permitiu uma especificação completa, rigorosa (sem ambigüidades nem inconsistências) e verificável dos requisitos dos ERTS. Os modelos formais são importantes porque a análise detalhada e o rigor necessário à sua construção permitem prevenir e detectar erros mais cedo, e conseqüentemente, a correção do erro se torna mais barata. Contudo, os modelos formais não são intuitivos e requerem um considerável esforço por parte dos projetistas para entenderem a notação usada. Por outro lado, as anotações dos modelos semiformais são amigáveis e intuitivas. Assim, é importante adotar o uso colaborativo dos modelos semiformais e formais a fim de encontrar erros ou inconsistências;

- a utilização dos diagramas comportamentais da SysML para modelar os ERTS. Como a equipe de desenvolvimento de um ERTS é interdisciplinar, envolvendo profissionais de áreas distintas (engenharia de *software*, mecânica, elétrica e eletrônica), a SysML mostrou ser adequada para melhorar a comunicação entre as áreas afim, pois a SysML inclui, em uma única especificação, uma visão integrada do sistema, incluindo *hardware*, *software* e partes eletro-mecânicas. Um dos problemas enfrentados na utilização da SysML para modelar os ERTS foi a busca por material de apoio para o estudo da linguagem. Além da especificação da linguagem em versão rascunho, foram encontrados apenas alguns relatórios técnicos e artigos disponibilizados por empresas que estão trabalhando na sua especificação;

- a especificação das restrições dos ERTS através de MARTE, pois através dele foi possível padronizar as notações, e assim, facilitar tanto o processo de modelagem, quanto o processo de mapeamento. De forma semelhante a SysML, um dos problemas enfrentados na utilização de MARTE para especificar as restrições dos ERTS foi a busca por material de apoio para o estudo;

- a criação de uma metodologia para auxiliar o processo de avaliação de desempenho das especificação de sistemas crítico. Essa metodologia é composta por uma série de atividades que envolvem desde o projeto dos requisitos e modelagem dos diagramas comportamentais da SysML até a geração e análise dos modelos ETPN. Com a aplicação da metodologia proposta, vários problemas relacionados ao desenvolvimento dos sistemas embarcados não somente críticos, mas em geral, poderão ser solucionados;

- o desenvolvimento de um método para o mapeamento dos diagramas comportamentais da SysML (estado, atividades e seqüência) em modelos ETPN, levando em consideração as restrições de tempo e anotações de energia. Devido à utilização das ETPN como ferramenta análise, foi possível aferir métricas. Por exemplo, pode-se calcular o tempo de execução para o pior ou melhor caso, e também, o consumo

de energia associado aos *traces* dos tempos de execução. Além disso, é importante ressaltar que através dos modelos ETPN, foi possível obter todas as vantagens dos métodos formais e aplicar na promoção da eficiência, *corretude* e validação dos requisitos. Um dos problemas enfrentados nesse mapeamento foi conversão dos cenários em modelos ETPN de forma manual, pois a medida que o cenário crescia, a modelo ETPN gerado pelo processo de mapeamento também crescia, deixando o modelo resultante muito grande e às vezes confuso;

- o cálculo de estimativas do tempo de execução dos ERTS. Os valores estimados do BCET (*Best Case Execution Time*) e WCET (*Worst Case Execution Time*) são muito importantes no projeto de sistemas de tempo-real para escalonamento de tarefas e, no caso do WCET, para indicar se uma determinada plataforma de *hardware* é capaz de executar o sistema cumprindo as restrições de tempo impostas a ele;

- o cálculo de estimativas do consumo de energia dos ERTS. Essas estimativas são de essencial importância, pois geralmente os ERTS possuem uma fonte de energia restrita, de tal forma que se essa fonte chegar ao fim, o sistema pára de funcionar. Portanto métricas relacionadas ao consumo de energia se tornaram extremamente necessárias nos projetos desses sistemas. Atualmente não existem outros trabalhos na literatura que propõe o mapeamento de modelos semiformais em modelos formais com o intuito de realizar estimativas de consumo de energia.

6.2 LIMITAÇÕES

As limitações deste trabalho são as seguintes:

- o mapeamento dos diagramas comportamentais da SysML é realizado de forma manual. Devido a esse processo manual, falhas podem ocorrer na atividade de conversão, e consequentimente, erros podem ser inclusos no modelo gerado;

- o consumo de energia é calculado de forma manual, pois a ferramenta INA não permite a adição de energia as transições do modelo. Neste caso, para cada *trace* encontrado dos tempos de execução, o consumo de energia era calculado manualmente;

- alguns elementos dos diagramas comportamentais da SysML foram omitidos do processo de mapeamenteo, tais como: estado de história, mensagens assíncronas, operador de interação *break*, entre outros. Isso ocorreu devido aos seguintes fatores: (i) restrições de tempo, (ii) complexidade do modelo gerado pelo processo de mapeamento, nesse caso, para as mensagens síncronas e (iii) alguns desses elementos são muito pouco usados para modelar os ERTS;

- as estimativas do tempo de execução médio (*Average Case Execution Time*) não são consideradas. Em sistemas embarcados de tempo-real esse tipo de estimativa não

é essencial, pois o que se deseja é que as restrições temporais não sejam violadas. No entanto, essas estimativas são importantes em sistemas embarcados não críticos, como, por exemplo, os sistemas móveis.

6.3 TRABALHOS FUTUROS

Como trabalho futuro, pode-se incluir a implementação de uma ferramenta para a geração automática dos modelos ETPN, a partir dos diagramas comportamentais da SysML. Essa ferramenta poderá contribuir em diversos aspectos. O tempo do processo de mapeamento poderá ser reduzido, uma vez que o modelo ETPN será gerado automaticamente. Devido a esse processo automático, também poderá ser garantido que não ocorreram falhas no processo de conversão do modelo de alto-nível para o domínio das ETPN. Essa ferramenta também permitirá um completa abstração com relação à utilização das ETPN. Além disso, as estimativas tanto do tempo de execução quanto do consumo de energia serão realizadas diretamente na ferramenta. Essa ferramenta também será integrada com outras ferramentas já existentes, tais como: INA, PepTool e Snoopy.

Este trabalho também poderá ser estendido para cobrir os diagramas comportamentais da UML, tais como o diagrama de colaboração, tempo e interatividade. Em especial, o diagrama de interatividade é de grande interesse, pois esse diagrama representa a fusão do diagrama de atividades e seqüência. Um outro trabalho futuro será relacionado a estressar mais a capacidade de simulação e análise das ETPN a fim de encontrar informações significantes. Adicionalmente, pode-se incluir, também, a realização de outros estudos de caso.

REFERÊNCIAS BIBLIOGRÁFICAS

[AMC+08] Ermeson Andrade, Paulo Maciel, Gustavo Callou, Eduardo Tavares, and Bruno Nogueira. *Mapping SysML State Machine Diagram to Time Petri Net for Analysis and Verification of Embedded Real-Time Systems with Energy Constraints*. 29 2008-Oct. 4 2008.

[AMCNa] Ermeson Andrade, Paulo Maciel, Gustavo Callou, and Bruno Nogueira. *Mapping UML Interaction Overview Diagram to Time Petri Net for Analysis and Verification of Embedded Real-Time Systems with Energy.*

[AMCNb] Ermeson Andrade, Paulo Maciel, Gustavo Callou, and Bruno Nogueira. *Mapping UML Sequence Diagram to Time Petri Net for Requirement Validation of Embedded Real-Time Systems with Energy Constraints.*

[AMCNc] Ermeson Andrade, Paulo Maciel, Gustavo Callou, and Bruno Nogueira. *A Methodology for Mapping SysML Activity Diagram to Time Petri Net for Requirement Validation of Embedded Real-Time Systems with Energy Constraints.*

[AMN+05] L. Amorim, P. Maciel, M. Nogueira, R. Barreto, and E. Tavares. A Methodology for Mapping Live Sequence Chart to Coloured Petri Net. *Systems, Man and Cybernetics, 2005 IEEE International Conference on*, 2005.

[AMN+06] L. Amorim, P. Maciel, M. Nogueira, R. Barreto, and E. Tavares. Mapping live sequence chart to coloured petri nets for analysis and verification of embedded systems. *ACM SIGSOFT Software Engineering Notes*, 31(3):1–25, 2006.

[ATM+] Julian Araújo, Erica Teixeira, Paulo Maciel, Fábio Chicout, and Ermeson Andrade. Performance modeling for evaluation and planning of electronic funds transfer systems with bursty arrival traffic. *The First International Conference on Intensive Applications and Services - INTENSIVE 2009.*

[Bal01] G. Balbo. Introduction to Stochastic Petri Nets. *Lectures on Formal Methods and Performance Analysis: First EEF/Euro Summer School on Trends in Computer Science, Berg en Dal, The Netherlands, July 3-7, 2000: Revised Lectures*, 2001.

[BDM02] S. Bernardi, S. Donatelli, and J. Merseguer. From UML sequence diagrams and statecharts to analysable petri net models. *Proceedings of the 3rd international workshop on Software and performance*, 2002.

[BGdMT98a] G. Bloch, S. Greiner, H. de Meer, and S. Trivedi. Queueing Networks and Markov Chains: Modeling and Performance Evaluation with Computer Science Applications, 1998.

[BGdMT98b] G. Bolch, S. Greiner, H. de Meer, and K.S. Trivedi. *Queueing networks and Markov chains: modeling and performance evaluation with computer science applications*. Wiley-Interscience New York, NY, USA, 1998.

[BL04] R. Barreto and R. Lima. A novel approach for off-line multiprocessor scheduling in embedded hard real-time systems. *Design Methods And Applications For Distributed Embedded Systems*, 2004.

[BP01a] L. Baresi and M. Pezzè. Improving UML with Petri nets. *Electronic Notes in Theoretical Computer Science*, 44(4):107–119, 2001.

[BP01b] L. Baresi and M. Pezzè. On formalizing uml with high-level petri nets. pages 276–304, 2001.

[CA07] B.H.C. Cheng and J.M. Atlee. Research Directions in Requirements Engineering. *International Conference on Software Engineering*, pages 285–303, 2007.

[Car05] Fernando Ferreirade Carvalho. *Avaliação Estocástica de Consumo de Energia no Projeto de Sistemas Embarcados.* MSc Thesis, Centro de Informática. Universidade Federal de Pernambuco, April 2005.

[CMA+08a] Gustavo Callou, Paulo Maciel, Ermeson Andrade, Bruno Nogueira, and Eduardo Tavares. A coloured petri net based approach for estimating execution time and energy consumption in embedded systems. pages 134–139, New York, NY, USA, 2008. ACM.

[CMA+08b] Gustavo Callou, Paulo Maciel, Ermeson Andrade, Bruno Nogueira, and Eduardo Tavares. Estimation of energy consumption and execution time in early phases of design lifecycle: an application to biomedical systems. 2008.

[CMA+08c] Gustavo Callou, Paulo Maciel, Ermeson Andrade, Bruno Nogueira, and Eduardo Tavares. A formal approach for estimating embedded system execution time and energy consumption. 2008.

[CMT89] G. Ciardo, J. Muppala, and K. Trivedi. SPNP: stochastic Petri net package. *Petri Nets and Performance Models, 1989. PNPM89., Proceedings of the Third International Workshop on*, pages 142–151, 1989.

[CW96] E.M. Clarke and J.M. Wing. Formal methods: state of the art and future directions. *ACM Computing Surveys (CSUR)*, 28(4):626–643, 1996.

[Dav05] R. David. *Discrete, Continuous, And Hybrid Petri Nets.* Springer, 2005.

[dC04] Fernando Ferreira de Carvalho. *Avaliação estocástica de consumo de energia no projeto de sistemas embarcados. (in portuguese)*. MSc Thesis, Centro de Informática, Universidade Federal de Pernambuco, March 2004.

[DdSS] L.M. Döll, J.U.F. de Souza, and P.C. Stadzisz. Verificação e Validação de Sistemas Orientados a Objetos Usando Redes de Petri.

[dOJ06] Meuse Nogueira de Oliveira Júnior. *Estimativa do Consumo Energia devido ao Software: uma Abordagem baseada em Redes de Petri Colorida.* PhD Thesis, Centro de Informática. Universidade Federal de Pernambuco, April 2006.

[DRP99a] E. Dustin, J. Rashka, and J. Paul. *Automated Software Testing: Introduction, Management, and Performance.* Addison-Wesley Professional, 1999.

[DRP99b] E. Dustin, J. Rashka, and J. Paul. *Automated Software Testing: Introduction, Management, and Performance.* Addison-Wesley Professional, 1999.

[dSB05] Raimundo da Silva Barreto. *A Time Petri Net-Based Methodology for Embedded Hard Real-Time Systems Software Synthesis.* PhD Thesis, Centro de Informática. Universidade Federal de Pernambuco, April 2005.

[EBKB02] M. Elkoutbi, M. Bennani, R. K. Keller, and M. Boulmalf. *Real-time system specifications based on UML Scenarios and Timed Petri Nets.* 2002.

[EES+03] J. Engblom, A. Ermedahl, M. Sjoedin, J. Gustafsson, and H. Hansson. Worst-case execution-time analysis for embedded real-time systems. *International Journal on Software Tools for Technology Transfer (STTT)*, 4(4):437–455, 2003.

[FN80] G. Florin and S. Natkin. Evaluation Based upon Stochastic Petri Nets of the Maximum Throughput of a Full Duplex Protocol. *Informatik-Fachberichte; Vol. 52*, pages 280–288, 1980.

[GH85] D. Gross and C.M. Harris. *Fundamentals of queueing theory .* John Wiley & Sons, Inc. New York, NY, USA, 1985.

[Gia02] B. Gianfranco. Introduction to Stochastic Petri Nets. *Lectures on Formal Methods and Performance Analysis: first EEF/Euro summer school on trends in computer science. Springer-Verlag, Berlin Heidelberg New York*, 2002.

[GKZH94] R. German, C. Kelling, A. Zimmermann, and G. Hommel. *TimeNET: A Toolkit for Evaluating Non-Markovian Stochastic Petri Nets.* Technische Universität Berlin, Fachbereich 13, Informatik, 1994.

[Gro95] Standish Group. The standish group chaos report. www.projectsmart.co.uk/docs/chaos_report.pdf, 1995.

[GUE04] G.T.A. GUEDES. UML: Uma abordagem prática. *São Paulo: Novatec*, 2004.

[GV01] C. Girault and R. Valk. *Petri Nets for System Engineering: A Guide to Modeling, Verification, and Applications*. Springer-Verlag New York, Inc. Secaucus, NJ, USA, 2001.

[H+87] D. Harel et al. Statecharts: A Visual Formalism for Complex Systems. 1987.

[Hal07] A. Hall. Realising the Benefits of Formal Methods. *Journal of Universal Computer Science*, 13(5):669–678, 2007.

[Har87] D. Harel. Statecharts: A visual formalism for complex systems. *Science for Computer Programming*, 1987.

[Hau06] M. Hause. The SysML Modelling Language. 2006.

[HoCU93] M. Hennessy, Brighton (GB). School of Cognitive, and Computing Sciences Sussex Univ. Timed Process Algebras: A Tutorial. *Program Design Calculi*, 1993.

[HS04] Z. Hu and S.M. Shatz. Mapping UML Diagrams to a Petri Net Notation for System Simulation. *16th Int. Conf. on Software Engineering & Knowledge Engineering (SEKE 2004)*, pages 213–9, 2004.

[JÓ98] M. Nogueira Oliveira Júnior. *Desenvolvimento de Um Protótipo para a Medida Não Invasiva da Saturação Arterial de Oxigênio em Humanos - Oxímetro de Pulso (in portuguese)*. MSc Thesis, Departamento de Biofísica e Radiobiologia, Universidade Federal de Pernambuco, August 1998.

[Jan] C.V. Janousek. *Modelling Objects by Petri Nets*. PhD thesis, PhD thesis, Department of Computer Science and Engineering, Technical University of Brno, Czech Republic, 1998.(In Czech). CiteSeer. IST-Copyright Penn State and NEC.

[Jen91] K. Jensen. Coloured Petri Nets: A High Level Language for System Design and Analysis. *Advances in Petri Nets 1990*, 1991.

[Jen92] K. Jensen. *Coloured Petri Nets: Basic Concepts, Analysis Methods, and Practical Use*. Springer, 1992.

[JM97] J.M. Jézéquel and B. Meyer. Design by Contract: The Lessons of Ariane. 1997.

[JMBC04] M.N.O. Junior, P.R.M. Maciel, R.S. Barreto, and F.F. Carvalho. Towards a Software Power Cost Analysis Framework Using Colored Petri Net. *LECTURE NOTES IN COMPUTER SCIENCE*, pages 362–371, 2004.

[JML+05] M.N.O. Junior, P. Maciel, R. Lima, A. Ribeiro, C. Oliveira, A. Arcoverde, R. Barreto, E. Tavares, and L. Amorin. A Retargetable Environment for Power-Aware Code Evaluation: An Approach Based on Coloured Petri Net. *LECTURE NOTES IN COMPUTER SCIENCE*, 3728:49, 2005.

[KP99] P. King and R. Pooley. Using UML to Derive Stochastic Petri Net Models. *Proceedings of the Fifteenth Annual UK Performance Engineering Workshop*, pages 45–56, 1999.

[KP00] P. King and R. Pooley. Derivation of Petri Net Performance Models from UML Specifications of Communications Software. *Computer Performance Evaluation: Modelling Techniques and Tools: 11th International Conference, TOOLS 2000, Schaumburg, IL, USA, March 27-31, 2000: Proceedings*, 2000.

[Lim01] ARM Limited. ARM7TDMI technical reference manual. *ARM DDI 0210A*, 2001.

[LPK+00] J. Lee, J.I. Pan, J.Y. Kuo, Y.Y. Fanjiang, and S. Yang. Towards the verification of scenarios with time Petri-nets. *Computer Software and Applications Conference, 2000. COMPSAC 2000. The 24th Annual International*, pages 503–508, 2000.

[LT93] NG Leveson and CS Turner. An investigation of the Therac-25 accidents. *Computer*, 26(7):18–41, 1993.

[Mar89] M.A. Marsan. Stochastic Petri Nets: An Elementary Introduction. *Advances in Petri Nets*, 424:1–29, 1989.

[MAR07] OMG MARTE. Profile for Modeling and Analysis of Real-Time and Embedded systems (MARTE), Beta1. 2007.

[MBC+98] M.A. Marsan, G. Balbo, G. Conte, S. Donatelli, and G. Franceschinis. Modelling with Generalized Stochastic Petri Nets. *ACM SIGMETRICS Performance Evaluation Review*, 26(2), 1998.

[MC87] M.A. Marsan and G. Chiola. On Petri nets with deterministic and exponentially distributed firing times. *Advances in Petri Nets*, 266:132–145, 1987.

[MCB84] M.A. MARSAN, G. CONTE, and G. BALBO. A Class of Generalized Stochastic Petri Nets for the Performance Evaluation of Multiprocessor Systems. *ACM Transactions on Computer Systems*, 2(2):93–122, 1984.

[MCBD02a] J. Merseguer, J. Campos, S. Bernardi, and S. Donatelli. A compositional semantics for UML state machines aimed at performance evaluation. *Discrete Event Systems, 2002. Proceedings. Sixth International Workshop on*, pages 295–302, 2002.

[MCBD02b] J. Merseguer, J. Campos, S. Bernardi, and S. Donatelli. A compositional semantics for UML state machines aimed at performance evaluation. *Discrete Event Systems, 2002. Proceedings. Sixth International Workshop on*, pages 295–302, 2002.

[MCM02] J. Merseguer, J. Campos, and E. Mena. Performance evaluation for the design of agent-based systems: A Petri net approach. *Proceedings of the Workshop on Software Engineering and Petri Nets, within the 21st International Conference on Application and Theory of Petri Nets*, pages 1–20, 2002.

[MF76] P. Merlin and D. J. Faber. Recoverability of communication protocols: Implicatons of a theoretical study. *IEEE Transactions on Communications*, 24(9):1036–1043, Sept. 1976.

[MLC96] P.R.M. Maciel, R.D. Lins, and P.R.F. Cunha. Introdução às Redes de Petri e Aplicações. *X Escola de Computação, Campinas, SP*, 1996.

[Mur89] T. Murata. Petri nets: Properties, analysis and applications. *Proc. IEEE*, 77(4):541–580, April 1989.

[NN73] JD Noe and GJ Nutt. Macro E-Nets for Representation of Parallel Systems. *IEEE Transactions on Computers*, 31(9):718–727, 1973.

[OMG89] The object management group (omg). http://www.omg.org/, 1989.

[Pen03] T. Pender. *UML Bible*. John Wiley & Sons, Inc. New York, NY, USA, 2003.

[Pet62] C. A. Petri. *Kommunikation mit Automaten*. PhD Dissertation, Darmstad University, Germany, 1962.

[Ram74] C. Ramchandani. Analysis of asynchronous concurrent systems by Timed Petri Nets. Project MAC-TR 120. *Massachussets Institute of Technology, feb*, 1974.

[RGG96] E. Rudolph, P. Graubmann, and J. Grabowski. Tutorial on Message Sequence Charts. *Computer Networks and ISDN Systems*, 28(12):1629–1641, 1996.

[RJ98] J.T. Russell and M.F. Jacome. Software power estimation and optimization for high performance, 32-bit embedded processors. In *Proc. Int. Conf. Computer Design*, pages 328–333, 1998.

[Sil06] A. J. Silva. *Aspectos da Modelagem em SysML Ligados a Seleção de Processador para Sistema Embutido*. 2006.

[Sin96] A. Singhal. Real time systems: A survey. Technical report, Computer Science Department. University of Rochester, December 1996.

[Som06] I. Sommerville. *Software Engineering.* Addison-Wesley, 2006.

[SPT03] OMG SPT. Profile for Schedulability, Performance, and Time Specification. *Object Management Group*, 2003.

[SR99] P. Starke and S. Roch. *INA - Integrated Net Analyzer - Version 2.2.* Humbolt Universität zu Berlin - Institut für Informatik, 1999.

[Sys07] OMG SysML. Systems Modeling Language (SysML) Specification final report. *Object Management Group*, 2007.

[Tav06] Eduardo Antonio Guimaraes Tavares. *A Time Petri Net Based Approach for Software Synthesis in Hard Real-Time Embedded Systems with Multiple Processors.* MSc Thesis, Centro de Informática. Universidade Federal de Pernambuco, April 2006.

[TBM+07] Eduardo Tavares, Raimundo Barreto, Paulo Maciel, Jr. Meuse Oliveira, Leonardo Amorim, Fernando Rocha, and Ricardo Lima. Software synthesis for hard real-time embedded systems with multiple processors. *SIGSOFT Softw. Eng. Notes*, 32(2):1–10, 2007.

[TM06] E. Tavares and P. Maciel. Amalghma tool. http://www.cin.ufpe.br/∼eagt/tools/, 2006.

[TMB+05] Eduardo Tavares, Paulo Maciel, Arthur Bessa, Raimundo Barreto, Leonardo Barros, Jr Meuse Oliveira, and Ricardo Lima. A time petri net based approach for embedded hard real-time software synthesis with multiple operational modes. In *SBCCI '05: Proceedings of the 18th annual symposium on Integrated circuits and system design*, pages 98–103, New York, NY, USA, 2005. ACM.

[TMS+07] E. Tavares, P. Maciel, B. Silva, M. Oliveira, and R. Rodrigues. Modelling and scheduling hard real-time biomedical systems with timing and energy constraints. *Electronics Letters*, 43(19):1015–1017, 2007.

[TMSO08] E. Tavares, P. Maciel, B. Silva, and M.N. Oliveira. Hard real-time tasks' scheduling considering voltage scaling, precedence and exclusion relations. *Information Processing Letters*, 2008.

[TMW94] V. Tiwari, S. Malik, and A. Wolfe. Power analysis of embedded software: a first step towards softwarepower minimization. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 2(4):437–445, 1994.

[TZ05] J. Trowitzsch and A. Zimmermann. Real-Time UML State Machines: An Analysis Approach. *Object Oriented Software Design for Real Time and Embedded Computer Systems*, 2005.

[TZ06] J. Trowitzsch and A. Zimmermann. Using UML state machines and petri nets for the quantitative investigation of ETCS. *Proceedings of the 1st international conference on Performance evaluation methodolgies and tools*, 2006.

[TZH05] J. Trowitzsch, A. Zimmermann, and G. Hommel. Towards Quantitative Analysis of Real-Time UML Using Stochastic Petri Nets. *13th Int. Workshop on Parallel and Distributed Real-Time Systems (WPDRTS)*, 2005.

[UML05] OMG UML. 2.0 Superstructure Specification. *Object Management Group*, 2005.

[VCF+06] N.L. Vijaykumar, SV Carvalho, C.R.L. Francês, V. Abdurahiman, and ASM Amaral. Performance Evaluation from Statecharts Representation of Complex Systems: Markov Approach. *IV WPerformance, SBC*, pages 183–202, 2006.

[vdAvHR00] WMP van der Aalst, KM van Hee, and HA Reijers. Analysis of discrete-time stochastic petri nets. *Statistica Neerlandica*, 54(2):237–255, 2000.

[VLM+03] R.A. Vinter, W. Lisa, L.H. Michael, et al. CPN Tools for Editing, Simulating, and Analysing Coloured Petri Net. *Proceedings of Applications and Theory of Petri Nets*, pages 23–27, 2003.

[Wal88] J. Walrand. *An Introduction to Queueing Networks*. Prentice Hall, 1988.

[WE01] Fabian Wolf and Rolf Ernst. Execution cost interval refinement in static software analysis. *J. Syst. Archit.*, 47(3-4):339–356, 2001.

[Wol94] WH Wolf. Hardware-software co-design of embedded systems [and prolog]. *Proceedings of the IEEE*, 82(7):967–989, 1994.

[Zub80] WM Zuberek. Timed Petri nets and preliminary performance evaluation. *Proceedings of the 7th annual symposium on Computer Architecture*, pages 88–96, 1980.

Dissertação de Mestrado apresentada por **Ermeson Carneiro de Andrade** à Pós-Graduação em Ciência da Computação do Centro de Informática da Universidade Federal de Pernambuco, sob o título **"Modelagem e Análise de Especificações de Sistemas Embarcados Críticos com Restrições de Energia"** orientada pelo **Prof. Paulo Romero Martins Maciel** e aprovada pela Banca Examinadora formada pelos professores:

Prof. Paulo Romero Martins Maciel
Centro de Informática / UFPE

Prof. Ricardo José Paiva de Britto Salgueiro
Departamento de Ciência da Computação e Estatística / UFS

Prof. Ricardo Massa Ferreira Lima
Centro de Informática / UFPE

Visto e permitida a impressão.
Recife, 2 de março de 2009.

Prof. FRANCISCO DE ASSIS TENÓRIO DE CARVALHO
Coordenador da Pós-Graduação em Ciência da Computação do
Centro de Informática da Universidade Federal de Pernambuco.

Em tempo: na terceira linha onde se lê Modelagem e Análise de Especificações de Sistemas Embarcados Críticos com Restrições de Energia, leia-se Modelagem e Análise de Especificações de Sistemas Embarcados de Tempo-Real Críticos com Restrições de Energia.