VINÍCIUS CAMARGO ANDRADE

TRANSFORMAÇÃO DE MODELOS DE DIAGRAMA DE SEQUÊNCIA UML CONTEMPLANDO RESTRIÇÕES DE TEMPO E ENERGIA PARA REDE DE PETRI TEMPORAL

Dissertação apresentada como requisito parcial à obtenção do grau de Mestre. Programa de Pós-Graduação em Informática, Setor de Ciências Exatas, Universidade Federal do Paraná.
Orientadora: Profª. Drª. Letícia Mara Peres

CURITIBA

2013

VINÍCIUS CAMARGO ANDRADE

TRANSFORMAÇÃO DE MODELOS DE DIAGRAMA DE SEQUÊNCIA UML CONTEMPLANDO RESTRIÇÕES DE TEMPO E ENERGIA PARA REDE DE PETRI TEMPORAL

Dissertação apresentada como requisito parcial à obtenção do grau de Mestre. Programa de Pós-Graduação em Informática, Setor de Ciências Exatas, Universidade Federal do Paraná.
Orientadora: Profª. Drª. Letícia Mara Peres

CURITIBA

2013

VINÍCIUS CAMARGO ANDRADE

TRANSFORMAÇÃO DE MODELOS DE DIAGRAMA DE SEQUÊNCIA UML CONTEMPLANDO RESTRIÇÕES DE TEMPO E ENERGIA PARA REDE DE PETRI TEMPORAL

Dissertação aprovada como requisito parcial à obtenção do grau de Mestre no Programa de Pós-Graduação em Informática da Universidade Federal do Paraná, pela Comisso formada pelos professores:

Orientadora: Profª. Drª. Letícia Mara Peres
Departamento de Informática, UFPR

Prof. Dr. Marcos Didonet Del Fabro
Departamento de Informática, UFPR

Prof. Dr. Andrey Ricardo Pimentel
Departamento de Informática, UFPR

Profª. Drª. Simone Nasser Matos
Departamento de Informática, UTFPR

Curitiba, 09 de Abril de 2013

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

PARECER

Nós, abaixo assinados, membros da Banca Examinadora da defesa de Dissertação de Mestrado em Informática, do aluno Vinicius de Camargo Andrade, avaliamos o trabalho intitulado, *"Transformação de modelos de diagrama de sequência uml contemplando restrições de tempo e energia para rede de petri temporal"*, cuja defesa foi realizada no dia 09 de abril de 2013, às 14:00 horas, no Departamento de Informática do Setor de Ciências Exatas da Universidade Federal do Paraná. Após a avaliação, decidimos pela aprovação do candidato.

Curitiba, 09 de abril de 2013.

Profa. Dra. Leticia Mara Prestes
DINF/UFPR – Orientadora

Prof. Dr. Marcos Didonet Del Fabro
DINF/UFPR – Coorientador

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

Profa. Dra. Simone Nasser Matos
UTFPR – Membro Externo

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

Hoje são dias de guerra,

amanhã serão de glória.

AGRADECIMENTOS

Agradeço primeiramente a Deus pela saúde e sabedoria concedidas para a realização deste trabalho.

A meus pais, Paulo Roberto de Andrade e Milze de Fátima Camargo Andrade, pela vida e pelo apoio que dispuseram em meu favor nas horas difíceis.

Especialmente a Profª. Drª. Letícia Mara Peres, que me orientou com extremo profissionalismo e dedicação em todas as etapas desde grande desafio.

Ao Prof. Dr. Marcos Didonet Del Fabro, pelas valiosas contribuições por meio de sugestões que enriqueceram o trabalho e por sanar dúvidas no decorrer da implementação.

Ao Prof. Dr. Andrey Ricardo Pimentel e a Profª. Drª. Simone Nasser Matos pela composição da banca da Defesa de Dissertação.

Aos meus colegas de laboratório: Alex Mateus Porn, Luis Renato dos Santos e Rafael dos Passos Canteri, não só pelas contribuições através das discussões técnicas relevantes ao desenvolvimento deste trabalho, mas também pelos momentos de descontrações.

Em especial, a minha prima Bruna Camargo Hass e novamente ao meu amigo Luis Renato dos Santos, que estavam ao meu lado me apoiando sempre com uma palavra amiga e conselhos quando mais precisei.

A minha prima Tânia Mara Camargo, pela sua espiritualidade, muitas vezes acalmando-me quando me sentia preocupado com assuntos tanto no âmbito profissional quanto pessoal.

Ao meu amigo Guilherme Della Vechia e Tafael Mangoni pelas horas de descontração.

A secretaria do Programa de Pós-Graduação da UFPR, pelo apoio prestado durante estes dois anos de curso.

A CAPES pelo apoio financeiro através da bolsa de estudos.

A Universidade Federal do Paraná, especialmente ao Programa de Pós-Graduação em Informática – PPGInf, representado por sua diretoria, por me proporcionar este curso de tão alto nível.

A todos os demais familiares, amigos e professores, que sempre mentalizaram positivamente em meu favor e acreditaram no meu sucesso pessoal e profissional.

SUMÁRIO

LISTA DE FIGURAS

RESUMO

Linguagem Unificada de Modelagem (UML) é amplamente adotada para o desenvolvimento de aplicações. Entretanto, ela não foi projetada com estrutura formal que permite sua aplicação direta na verificação de sistemas embarcados de tempo real. Uma abordagem para preencher essa lacuna é transformar modelos UML em representações formais, como redes de Petri. No entanto, os trabalhos existentes não resolvem esta questão quando são utilizadas restrições de energia e de tempo. Este trabalho apresenta como novidade a transformação de diagramas de sequência UML com energia e tempo em modelos de rede de Petri temporal. Estes modelos de rede de Petri são então utilizados como entrada para a análise de tempo e do consumo de energia por ferramentas de verificação de software como Tina e GTT.

ABSTRACT

Unified Modeling Language (UML) is widely adopted for developing applications. However, it was not designed with formal structure suitable for verification of real-time embedded systems. An approach to fill this gap is to transform UML models into formal representations such as Petri nets. Yet, existing works do not address the issue when used both energy and time constraints. This paper presents as novelty a technique for transforming UML sequence diagrams with energy and time to Time Petri net models. These Petri net models are then used as input for the analysis of time and energy consumption by software verification tools like Tina and GTT.

CAPÍTULO 1

INTRODUÇÃO

Sistemas embarcados de tempo real são sistemas em que são considerados, além dos resultados de computação, o tempo em que esses resultados são obtidos. Muitas vezes, este tempo de execução ou de espera presente nos sistemas pode ser previsto ainda em fase de modelagem e recebe o nome de restrição temporal [50].

Além de restrições temporais, restrições de energia também estão presentes em diversos sistemas utilizados no cotidiano, desafiando o desenvolvedor de hardware a criar produtos cada vez mais complexos com um menor consumo de energia. Exemplos de elementos de hardware com este tipo de restrição são os dispositivos móveis, que dependem de um alto nível de processamento e um baixo consumo de energia, pois utilizam baterias que são uma fonte de energia restrita [4].

Existem diversas linguagens na literatura para a modelagem de sistemas. Destas, duas podem ser citadas: *Unified Modeling Language* (UML) e rede de Petri (RdP). Atualmente UML é a mais utilizada, porém não possui em sua especificação um método que permita representar restrições de tempo e energia em seu modelo. Para isso, foi criado um perfil UML [37] com a finalidade de definir os fluxos comuns do projeto para sistemas de tempo real/embarcados. Um desses fluxos de projeto permite definir, em diferentes pontos de vista ou modelos de aplicação, incluindo características funcionais e não funcionais, a arquitetura de hardware e a distribuição do aplicativo para a arquitetura. Este perfil recebeu o nome de *Modeling and Analysis of Real Time and Embedded systems* (MARTE) [37] e é atualmente um padrão para a modelagem em tempo real e aplicações embarcadas com a versão 2.0 da UML [35].

Por outro lado, rede de Petri é uma linguagem formal de modelagem. É um modelo que possui um embasamento matemático capaz de representar graficamente sistemas com diversas características e permitem análises e verificações formais posteriores no modelo.

Além disso, possibilita representações de restrições em suas extensões, como por exemplo, restrições temporais em rede de Petri temporal. Estas características permitem aumentar a qualidade e confiabilidade do software modelado.

Apesar das características de redes de Petri temporal sobre diagramas de sequência UML, como por exemplo a possibilidade de análises e verificações formais, a modelagem de sistemas utilizando UML é mais simples. Neste contexto, a transformação de modelos torna-se útil, pois possibilita modelar um sistema contendo restrições de tempo e energia com uma linguagem usual e amplamente conhecida e verificar estas restrições com uma linguagem formal de modelagem através de uma conversão automática [40].

Atualmente são encontrados na literatura trabalhos envolvendo transformações de modelos de diagramas UML em redes de Petri. Porém muitos destes propõem métodos manuais baseados em estruturas da UML e de RdP, como por exemplo Staines [49]. Estes métodos de transformação demandam um tempo maior em seu processo além de exigir atenção do projetista no momento da conversão.

Há trabalhos que realizam a transformação baseada no conceito de modelagem especificado pelo OMG, como Du et. al. [17]. Este conceito proporciona suporte a diferentes linguagens de modelagem, garante que os modelos originados estejam semanticamente corretos e completos e possibilita a definição e execução de transformações genéricas. Esta arquitetura possui três camadas: modelo, metamodelo e metametamodelo. Onde o modelo é considerado entidade de primeira classe e deve estar em conformidade com seu metamodelo. Um metamodelo por sua vez é uma abstração de um determinado modelo e sua semântica deve obedecer a um metametamodelo, que possui sua semântica de acordo com seus próprios conceitos [5, 18].

Neste trabalho é apresentada e implementada uma técnica de transformação de diagrama de sequência UML contendo restrições de tempo e energia para rede de Petri temporal. Com os modelos de rede de Petri temporal gerados a partir da transformação é possível realizar a análise de tempo e do consumo de energia pela ferramenta GTT [41, 40].

1.1 Objetivos

O objetivo geral deste trabalho é **realizar a transformação de modelos de diagrama de sequência UML contendo restrições de tempo e energia em rede de Petri temporal para permitir verificações desses dois tipos de restrições.** Como objetivos específicos, tem-se:

1. Adequar os metamodelos de diagrama de sequência UML e rede de Petri temporal para que contemplem restrições de tempo e energia.

2. Implementar a transformação de diagramas de sequência UML em redes de Petri temporal.

3. Permitir a realização da análise intervalar, de tempo e energia, em Redes de Petri geradas a partir da transformação.

1.2 Organização do Trabalho

Este trabalho está dividido em cinco capítulos. O capítulo 2 apresenta a revisão bibliográfica do trabalho. Também relata o estudo sobre o desenvolvimento orientado a modelos e análise temporal. O capítulo 3 exibe as adaptações nos metamodelos, as definições das transformações e implementações do projeto. O capítulo 4 aborda três estudos de casos e compara os resultados das conversões e análises dos modelos obtidos por este trabalho. E, por fim, o capítulo 5 apresenta as conclusões do trabalho e algumas propostas para trabalhos futuros.

CAPÍTULO 2

REVISÃO BIBLIOGRÁFICA

Este capítulo aborda os tópicos referentes à revisão bibliográfica necessária para este trabalho. A seção 2.1 discorre sobre Desenvolvimento Orientado a Modelos. A UML (Linguagem Unificada de Modelagem) é descrita na seção 2.2. A seção 2.3 apresenta as definições necessárias ao entendimento de rede de Petri e rede de Petri temporal. Os trabalho relacionados se encontram na seção 2.4. A seção 2.5 conceitua sistemas embarcados. A seção 2.6 detalha o método de verificação de software. E, por fim, a seção 2.7 apresenta as considerações finais do capítulo.

2.1 Desenvolvimento Orientado a Modelos

O Desenvolvimento Orientado a Modelos (MDD, em inglês *Model-Driven Development*) é uma abordagem integrada para a arquitetura, desenvolvimento, testes e implantação de sistemas complexos e de alto desempenho [45].

A proposta do MDD é fazer com que o engenheiro se concentre em modelos de alto nível, diminuindo a necessidade que o mesmo interaja manualmente com o código-fonte dando-lhe um amparo contra as complexidades exigidas nas diferentes plataformas.

As vantagens desta abordagem são destacadas por Kleppe et al. [26], van Deursen e Klint [54], Bhanot et al. [8] e Mernik et al. [32]. São elas: produtividade, portabilidade, interoperabilidade, manutenção, documentação, comunicação, reutilização, verificações, otimizações e correções. Entre as desvantagens, Ambler [1] cita: rigidez, complexidade, desempenho, curva de aprendizado e alto investimento inicial.

As principais abordagens existentes na literatura para MDD são baseadas no conceito de metamodelagem especificado pelo *Object Management Group*[1] (OMG) [38], o que proporciona suporte a diferentes linguagens de modelagem, garantindo que os mo-

[1]Object Management Group (OMG) – Associação sem fins lucrativos dedicada a manter especificações para ser usadas pela indústria de computadores [42].

delos gerados estejam semanticamente corretos, completos e que possibilita a definição e execução de transformações genéricas. Este conceito é ilustrado na figura 2.1.

Um modelo é, segundo Bézivin e Gerbé [7], uma especificação ou descrição de um sistema e seu ambiente com um determinado propósito, representado na figura 2.1 no nível M1. Um modelo é frequentemente apresentado com uma combinação de gráficos e textos, onde este texto pode estar em uma linguagem de modelagem ou linguagem natural. A partir da abstração das propriedades de um determinado modelo é possível obter um metamodelo, ilustrado pelo nível M2. Um metamodelo "é a análise, construção e desenvolvimento de *frames*, regras, restrições, modelos e teorias aplicáveis e úteis para a modelagem de uma classe pré-definida de problemas" [10].

Neste contexto, os modelos são considerados entidades de primeira classe. Um modelo deve ser definido de acordo com a estrutura, fornecida pelo seu metamodelo, ou seja, um modelo deve estar em conformidade com o seu metamodelo. Da mesma forma, um metamodelo deve estar em conformidade com um metametamodelo [5]. Este último representado na figura 2.1 pelo nível M3.

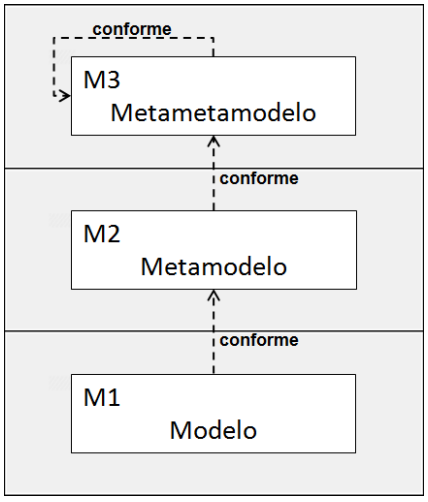

Figura 2.1: Arquitetura clássica de modelagem. Adaptado de Estrella et al. [18].

Nesta arquitetura de três camadas (modelo, metamodelo, metametamodelo), o metametamodelo geralmente está conforme com sua própria estrutura, ou seja, pode ser definido de acordo com seus próprios conceitos. Exemplos de metametamodelos são *Meta-Object Facility* (MOF) [38], definido pelo OMG, e *Ecore* [12], introduzido com o *Eclipse*

Modeling Framework [51].

2.1.1 Linguagem de Transformação

Surhone et al. [52] definem linguagem de transformação como uma linguagem de programação encarregada de transformar um modelo de origem em um modelo alvo, atendendo assim um objetivo previamente estabelecido. A transformação do modelo visa proporcionar facilidades para a geração de um modelo alvo, que está em conformidade com um metamodelo alvo, a partir de um modelo de origem que deve estar em conformidade com um metamodelo de origem.

Uma característica importante na engenharia de modelos é considerar que todos os itens são tratados como modelos. A transformação do modelo em si é definida como um modelo que deve estar em conformidade com um metamodelo de transformação que define, por sua vez, a estrutura de transformação do modelo. Como outros metamodelos, o metamodelo de transformação deve estar em conformidade com o metametamodelo considerado.

A figura 2.2 resume o processo de transformação de modelo. Um modelo M_a, conforme um metamodelo MM_a, está sendo transformado em um modelo M_b que corresponde ao metamodelo MM_b. Esta transformação é definida pelo modelo M_t, que satisfaz o metamodelo de transformação MM_t. Este último metamodelo, juntamente com os metamo-delos MM_a e MM_b, deve estar em conformidade com um metametamodelo, tais como *Meta-Object Facility (MOF)* [38] ou *Ecore* [12].

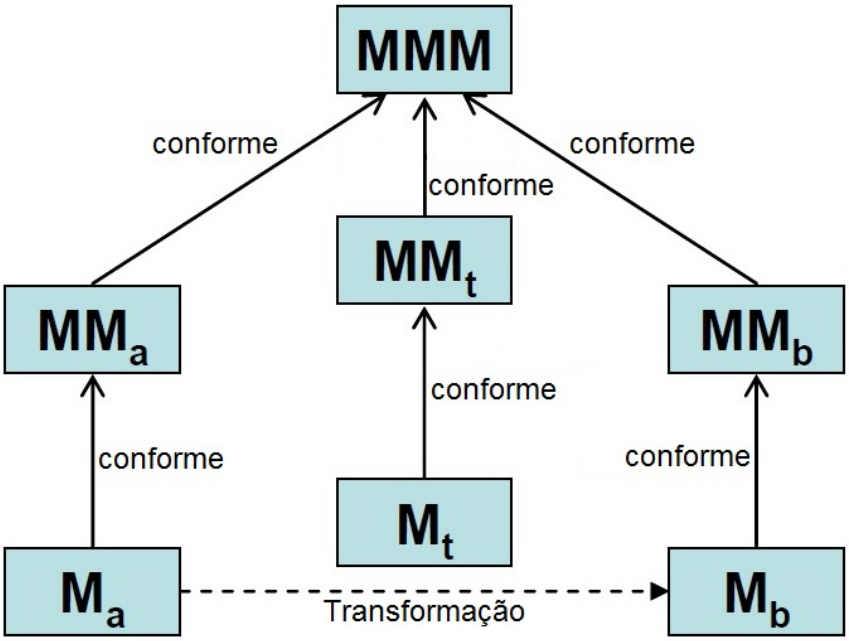

Figura 2.2: Visão global da transformação de modelo utilizando ATL[5].

Para a realização deste trabalho, foram estudadas linguagens de transformações existentes na literatura envolvendo maneiras distintas de transformação, como por exemplo, textuais e gráficas com o intuito de compará-las e optar pela utilização da linguagem que melhor se adequar ao contexto do trabalho. Dentre elas, as mais utilizadas são: *Query / View / Transformation Relations (QVT-R)* [36], *Query / View / Transformation Operational (QVT-O)* [36], e *Atlas Transformation Language (ATL)* [5].

Após o estudo das linguagens, realizou-se uma análise qualitativa entre elas considerando algumas características tidas como desejáveis para a realização da transformação de modelos. São elas:

- linguagem híbrida: incorpora tanto o paradigma imperativo quanto o paradigma declarativo;
- separação sintática: em algumas abordagens podem existir uma separação sintática das regras. Por exemplo, lado esquerdo e direito;
- pré-condições: possibilidade de definição de pré-condições para que a regra seja aplicada;

- parametrização de regras: consiste em tornar as regras mais reusáveis;
- estruturas intermediárias: a linguagem permite a definição de estruturas intermediárias que podem ser utilizadas na execução de uma regra. Estas estruturas não fazem parte dos modelos de origem ou alvo;
- reflexão: a linguagem permite reflexão, que por sua vez possibilita uma maior navegabilidade das propriedades dos elementos em questão e das regras de transformação;
- mecanismos de modularização: regras são agrupadas em módulos que podem ser importados para outros módulos para acessar seu conteúdo;
- mecanismos de reúso: definição de uma regra baseada em outras regras, utilizando herança entre regras e composições lógica das mesmas;
- rastreabilidade: refere-se a mecanismos para gravar a execução da transformação. A maneira mais comum de realizá-lo é através de *links* de rastreamento, que conectam elementos dos modelos de origem aos elementos alvos. Esta informação é útil tanto para realizar uma análise de impacto, quanto para realizar depuração.

A tabela 2.1, relaciona as linguagens de transformação citadas e as características. O "X" significa que a linguagem referente contempla a característica indicada na coluna, caso contrário, não.

Tabela 2.1: Características contempladas pelas linguagens de transformações da literatura

Características	QVT-R [36]	QVT-O [36]	ATL [5]
Linguagem híbrida			X
Separação sintática	X		X
Pré-condições	X	X	X
Parametrização de regras	X		X
Estruturas intermediárias		X	X
Reflexão			X
Mecanismos de modularização	X		X
Mecanismos para reúso	X	X	X
Rastreabilidade	X	X	X

Observa-se que as três linguagens de transformação analisadas contemplam pré-condições, mecanismos de reúso e rastreabilidade. As características separação sintática, parametrização de regras, estruturas intermediárias e mecanismos de modularização estão presentes na ATL e QVT. A estrutura da linguagem QVT é dividida em arquiteturas, por este motivo, QVT-R possui algumas características não cobertas por QVT-O, como separação sintática, parametrização de regras e os mecanismos de modularização. Em contrapartida, QVT-O contempla estruturas intermediárias, característica esta não contempladas por QVT-R.

Como apresentado na tabela 2.1, ATL é a única linguagem que possui as características linguagem híbrida e reflexão. A escolha de ATL no presente trabalho se deu pela mesma possuir várias ferramentas, disponibilidade de documentação e ser de fácil uso. Sua organização de regras possui estruturas e mecanismos de modulação e de reúso, é orientada ao modelo a modelo de origem, além de suportar qualquer modelo que possa ser definido utilizando metamodelo *Meta-Object Facility* (MOF) [5].

2.2 Linguagem Unificada de Modelagem – UML

Segundo Booch et al. [9], a Linguagem Unificada de Modelagem (UML, em inglês *Unified Modeling Language*) "é uma linguagem gráfica padrão para especificar, visualizar, documentar e construir artefatos de sistemas de software (...)". Ou seja, UML é um sistema de notação focada na modelagem de sistemas, proposto inicialmente para a aplicação de conceitos de orientação a objetos no desenvolvimento de software.

O OMG (em inglês, *Object Management Group*) é uma entidade sem fins lucrativos de padronização da indústria de computação que adotou a UML como linguagem padrão de desenvolvimento de projetos orientados a objetos [29, 19].

Dentre os diagramas da UML, o diagrama de sequência destaca-se por possuir um alto grau de detalhamento na transição de mensagens, permitindo identificar de forma explícita a sequência em que elas ocorrem e os objetos (remetente e receptor) envolvidos nesta transição. Por este motivo, optou-se pelo diagrama de sequência UML para a realização da transformação de modelos.

As restrições de tempo e energia necessárias na modelagem de sistemas embarcados não podem ser representados utilizando apenas o diagrama de sequência original. Para atender essa necessidade, a UML possui um perfil chamado "Modelagem e Análise de sistemas Embarcados de Tempo Real" (MARTE) [37].

A subseção 2.2.1 descreve o diagrama de sequência e a subseção 2.2.2 apresenta a modelagem MARTE.

2.2.1 Diagrama de Sequência

O Diagrama de Sequência (DS) UML possui uma notação que pode ilustrar as interações entre atores, classes e as operações iniciadas por eles [29]. A seguir são definidos, de acordo com [35], os elementos presentes no diagrama de sequência abordados neste trabalho.

- *interaction:* é uma unidade de comportamento que foca na troca de informações entre os elementos conectados;
- *interaction fragment:* é uma noção abstrata de uma unidade de interação mais geral;

- *interaction operator:* é o operador indicado na criação de um *combined fragment*. Um *interaction operator* pode ser: *alt, opt, par, loop, critical, neg, assert, strict, seq, ignore* e *consider*;

- *interaction operand:* representa um operador de um *combined fragment*;

- *combined fragment:* define um expressão do *interaction fragments*. Um *combined fragment* é definido por *interaction operator* e *interaction operands* correspondentes;

- *interaction operator kind:* é uma enumeração designada de diferentes tipos de operadores do *combined fragment*. Uma *interaction operand* define o tipo de operador de um *combined fragment*;

- *execution occurrence specification:* representa o momento no qual ações ou comportamentos iniciam ou terminam;

- *lifeline:* representa um participante individual na *interaction*;

- *message:* define uma comunicação particular entre *lifelines* e uma *interaction*;

- *execution specification:* é uma especificação da execução de uma unidade de comportamento ou ação interna da *lifeline.* A duração de uma *execution specification* é representada por duas *execution occurrence specifications*, uma inicial e outra final;

- *occurrence specification:* é uma unidade de semântica básica de interações. *Occurrence specification* são apenas pontos sintáticos nas extremidades das mensagens ou no início/fim de uma especificação de execução.

A figura 2.3 ilustra um diagrama de sequência. Neste exemplo o operador recebe várias chamadas pararelas e certifica a procedência das mesmas. Se a autenticidade da ligação for comprovada, esta é repassada à emergência.

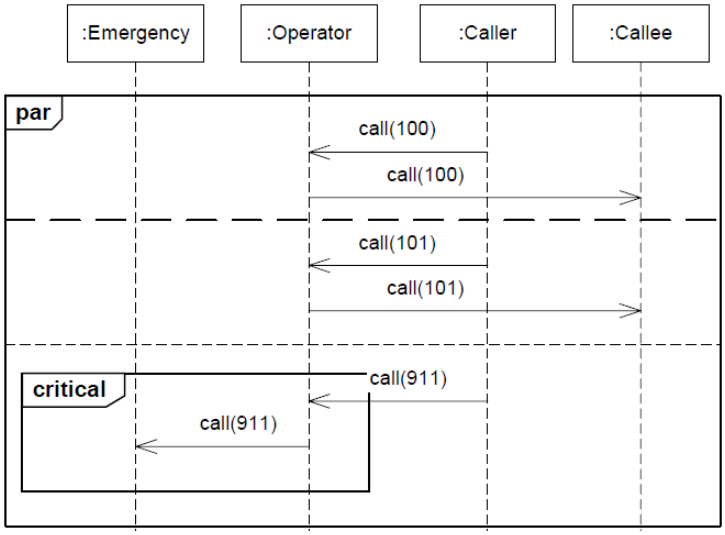

Figura 2.3: Exemplo de diagrama de sequência [35].

2.2.2 MARTE

A UML não contempla um modelo de desenvolvimento orientado a tempo real e aplicações embarcadas, por este motivo foi desenvolvido um perfil[2] para esta modelagem chamado de Modelagem e Análise de Tempo Real e sistemas Embarcados (MARTE, em inglês *Modeling and Analysis of Real-Time Embedded systems*). MARTE atualmente é um padrão para a modelagem em tempo real e aplicações embarcadas com UML.

O perfil MARTE define a semântica precisa de tempo e de modelagem de recursos. Esta semântica permite transformações automáticas de modelos para modelos de menor nível de abstração [37].

A figura 2.4 ilustra a aplicação de MARTE em um diagrama de sequência UML. Neste exemplo a mensagem *"Transmit_Command"* referente ao objeto *" :CAN_Bus"* possui uma restrição de tamanho de 80 Bytes, ou seja, esta mensagem não pode exceder este valor.

[2]Perfil – É um mecanismo de extensão do padrão UML que possibilita a adaptação de um metamodelo existente adicionando construções específicas de um domínio, plataforma ou método em particular [39].

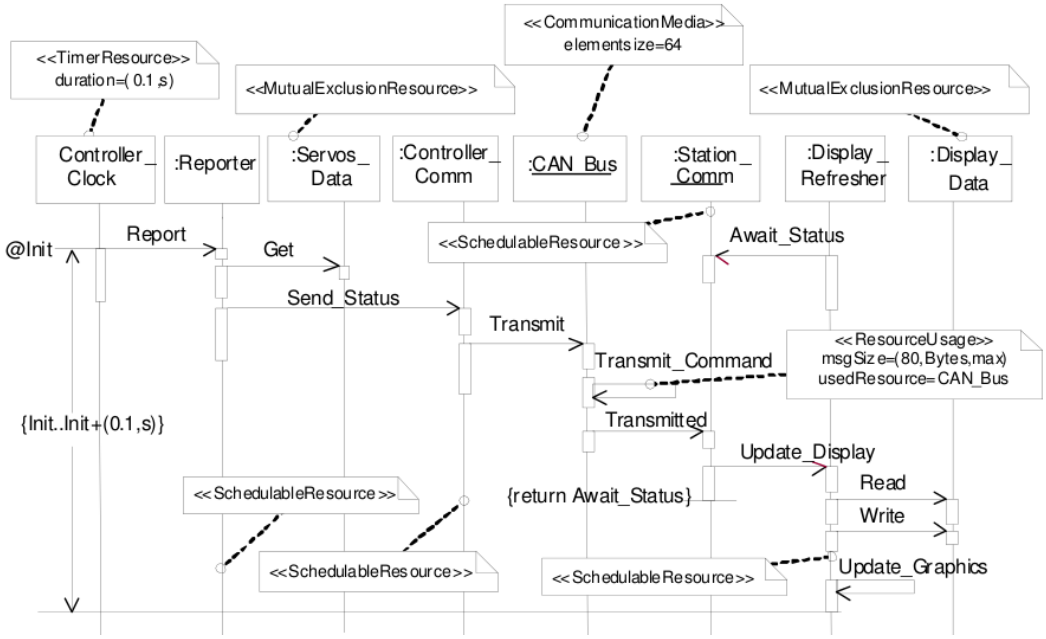

Figura 2.4: Aplicação de MARTE para uma restrição de tamanho da mensagem [37].

2.3 Rede de Petri

Segundo Murata [33] e Maciel et al. [30], uma Rede de Petri (RdP) é um modelo gráfico e matemático utilizado para representar sistemas com características concorrentes, assíncronas, distribuídas, paralelas, não determinísticas e estocásticas.

Redes de Petri aplicadas no desenvolvimento de sistemas computacionais são de grande importância, pois através de sua modelagem podem ser identificados diversos problemas. Também é possível aplicar RdP na análise e verificação de propriedades, não apenas limitando sua aplicação na área de modelagem.

Segundo Cardoso e Valette [13], uma rede de Petri é uma tupla, dada por:
$R =< P, T, Pre, Pos, M_0 >$, onde:

- P é um conjunto finito de lugares;
- T é um conjunto finito de transições;
- $Pre : (P \text{ x } T) \rightarrow \mathbb{N}$ é uma função de entrada em transições, que representa o peso dos arcos de entrada da transição;

- $Pos : (P \text{ x } T) \rightarrow \mathbb{N}$ é uma função de incidência de saída de transições que representa o peso dos arcos de saída de transições.

- $M_0 : P \rightarrow \mathbb{N}$ é a marcação inicial.

Uma rede de Petri $R =< P, T, Pre, Pos, M_0 >$ é apresentada na figura 2.5. O conjunto de lugares é $P = \{P1, P2, P3\}$. Cada lugar $p \in P$ é ilustrado por um círculo e um lugar p pode representar uma condição, um procedimento ou uma variável de estado. Nesta figura, lugares são: *"P1", "P2"* e *"P3"*. Para o mesmo exemplo, o conjunto de transições é $T = \{a, b, c, d\}$ e cada *transição* $t \in T$, é ilustrada por uma barra ou retângulo e representa um evento que ocorre no sistema. As transições são: *a, b, c* e *d*. Os arcos de entrada Pre e saída Pos são: $Pre(P2, c) = 3, Pre(P1, b) = Pre(P2, a) = Pre(P3, d) = 1, Pos(P2, d) = 3$ e $Pos(P1, a) = Pos(P2, b) = Pos(P3, c) = 1$. M_0, a marcação inicial, é dada por $M_0 = [(P1, 0), (P2, 3), (P3, 0)]$. Uma *marca* é ilustrada por um círculo negro em um *lugar*. A existência de marca ou não pode representar um objeto em um determinado estado, o uso de uma estrutura de dados ou a verdade de um predicado. Se não existe marcas neste lugar, o predicado é falso. Um conjunto de atribuições de marcas a lugares da rede é chamado de marcação. Com esse conhecimento pode-se dizer que $M(p)$ representa a marcação de um lugar $p \in P$, representar a marcação R. Se $M(p) > 0$, então este lugar específico é dito como marcado.

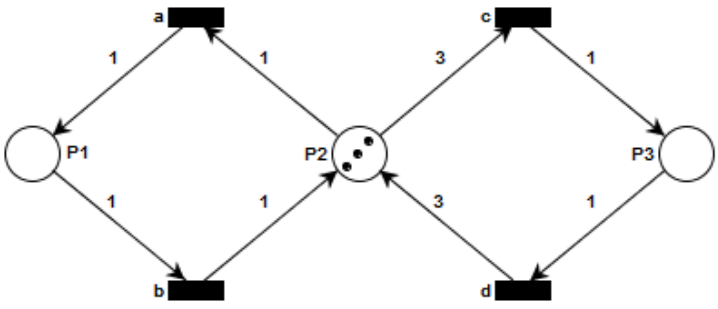

Figura 2.5: Exemplo de uma rede de Petri [13].

2.3.1 Rede de Petri Temporal

Segundo Berthomieu e Diaz [6], uma rede de Petri temporal (RdP-T) R é dada por uma tupla $R = (P, T, Pre, Pos, M_0, I)$, onde:

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

Cada transição $t \in T$ em uma rede de Petri temporal, é associada a um intervalo estático de disparo, dado por $e \in I$, denotado por $e(t)$ onde $e(t) = [a, b], a \leq b$. O menor e o maior valores do intervalo de disparo da transição t contados a partir de sua habilitação são representados respectivamente por a e b, ou seja, o disparo deve ocorrer entre estes valores [40]. Neste trabalho, um intervalo de energia que é gasta em um processamento é mapeado a um intervalo estático de valores, $e \in I$. Para efeito de modelagem e análises sobre a rede de Petri temporal, os intervalos podem ser referentes a tempo ou energia.

A figura 2.6 ilustra uma rede de Petri temporal. Neste exemplo, a transição *"T"* possui um intervalo $[1, 4]$ associados, representando respectivamente o valor mínimo e o valor máximo, de tempo ou energia, de disparo.

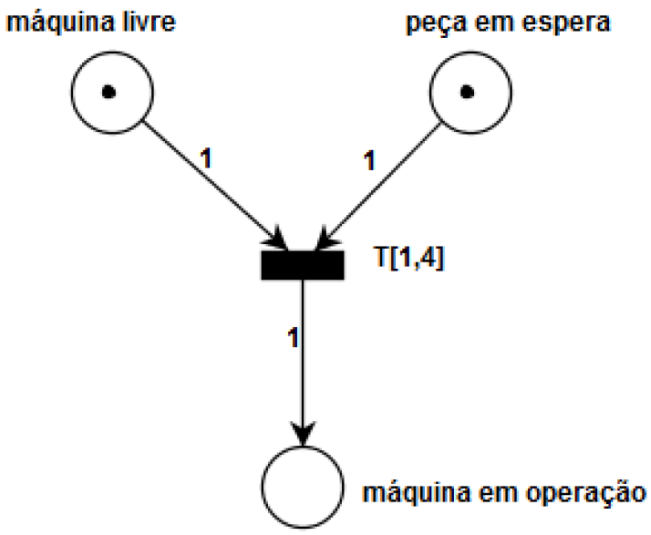

Figura 2.6: Exemplo de uma rede de Petri temporal [13].

A representação da análise intervalar proposta por Peres [40] em uma RdP-T é feita sobre um grafo de classes, que possui os seguintes elementos:

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

- nós – representam as classes;

- arcos – representam as transições disparadas para alcançar uma nova classe ou nó;

- classe de estados – uma *classe de estados* c_k de uma RdP-T é uma tupla $c_k = (M_k, W_k)$, onde M_k é uma marcação da rede e W_k é o conjunto das informações dos intervalos da classe. O conjunto dessas informações associadas às classes de estados tem por finalidade representar o comportamento temporal ou energético da rede em cada um de seus estados.

As informações dos intervalos associadas à classe de estados definidas em Peres [40] e usadas este trabalho são:

- tempo global – é a soma de intervalos de tempo relativo de cada transição e representa o tempo mínimo e máximo total de uma sequência de disparos;
- coeficiente de ajuste – utilizado para garantir que não haja imprecisão pela simples soma dos intervalos desde o início da execução da rede;
- tempo relativo – intervalo acumulado desde o momento em que a transição que leva a classe é habilitada até seu disparo.

Neste trabalho, um intervalo é composto por um valor mínimo e um valor máximo. Para efeito de modelagem e análises sobre a rede de Petri temporal, os intervalos podem ser referentes a tempo ou energia. Considera-se que um intervalo de energia são os valores mínimo e máximo de energia gasta em um processamento que são mapeados a um intervalo estático.

2.4 Trabalhos Relacionados

Staines [49] realiza uma transformação manual do diagrama de atividades UML em rede de Petri e rede de Petri colorida através do *Fundamental Modeling Concept Petri Net diagram* (FMC-PND) [27]. Em outro trabalho [48], este autor aborda a transformação de modelos pela técnica *Triple Graph Grammar* (TGG) [25] para conversão do diagrama de atividades UML em rede de Petri. Estes trabalhos propõem apenas a transformação manual

Kerkouche et al. [24] propõem uma transformação de modelos de diagrama de estados e de colaboração em rede de Petri colorida através de grafos. Este trabalho tem o objetivo de fazer uma conexão entre notação informal e formal para fins de análise e técnicas de simulação. Du et al. [17] abordam em seu trabalho um mapeamento de diagrama de classe e diagrama de sequência para rede de Petri colorida baseado na técnica *Edged Graph Grammar* (EGG) [57]. Destes, nenhum contém restrições em seus modelos.

O trabalho de Andrade et al. [4] transformam um modelo de origem DS em um modelo alvo RdP-T e contemplam restrições de tempo e energia em seus modelos. Neste trabalho a transformação é realizada envolvendo mapeamento manual.

2.5 Sistemas Embarcados

Segundo Noergaard [34], sistema embarcado é um sistema de computação com uma finalidade específica e limitado a recursos e funcionalidades de hardware e software, ou seja, é um sistema ao mesmo tempo completo e independente preparado para desempenhar uma determinada tarefa. Este tipo de sistema exige domínio absoluto do processador e periféricos, para a execução de requisitos específicos; exige facilidade para interação com o hardware; deve ser altamente eficiente, com a implementação voltada para melhor desempenho e exige controle de concorrência de tarefas [20, 34, 42, 46].

Sistemas embarcados na maioria das vezes são complexos em seu desenvolvimento, pois são sistemas que têm um grau de criticidade elevado e não são flexíveis, isto é, não possuem a capacidade de realizar outras tarefas a não ser as pré-determinadas.

O desenvolvimento de um sistema embarcado inicia-se com o estudo de viabilidade técnica, econômica e comercial do sistema. Constatada a viabilidade, procede-se às fases de desenvolvimento: engenharia de requisitos, projeto de software, implementação, verificação e validação, prototipação e padrões de projeto [47]. Nesta seção é abordada as fases de engenharia de requisitos. É na fase de engenharia de requisitos que ocorre a especificação do sistema embarcado dado pelo entendimento dos requisitos e seu refinamento. As subfases envolvidas na engenharia de requisitos são: análise de domínio, análise da solução, modelagem e validação de requisitos. Como saída, produz a especificação que descreve as propriedades funcionais e não funcionais do sistema a ser desenvolvido. A especificação de requisitos pode ser descrita em linguagens textuais, gráficas ou formais.

Este trabalho encontra-se no contexto de linguagens gráficas e formais para modelagem e validação de requisitos, que consiste em uma transformação de modelos de uma linguagem gráfica e fracamente formal para uma notação com sustentação matemática.

2.6 Método de Verificação de Software

Segundo Murata [33], a análise e verificação de propriedades comportamentais e estruturais de sistemas dinâmicos são permitidas através do uso de redes de Petri como estrutura

formal de representação de modelagem. Também é possível através de algumas extensões das redes de Petri, a representação de características temporais dos sistemas, como é o caso das redes de Petri temporais, que permitem a associação de cada evento do sistema a um intervalo de ocorrência e representado por uma transição da rede.

A análise intervalar de redes de Petri tem como principal objetivo verificar se um determinado estado será alcançado, dentro das restrições intervalares especificadas na rede [40]. Análises estruturais, comportamentais e temporais, são os modos de avaliação da representação do software embarcado por RdP.

A figura 2.7 ilustra uma visão geral das etapas propostas para a verificação de software embarcado usando RdP-T e análise intervalar [40]. Este método admite como entradas a especificação da execução de processos juntamente com suas restrições e sua arquitetura de software. Como saída gera os resultados da análise intervalar, com a verificação dos piores e melhores tempos de execução de acordo com certas políticas de escalonamento, como por exemplo prioridade fixa e primeiro *deadline*, e ainda um modelo em rede de Petri da interação entre os processos.

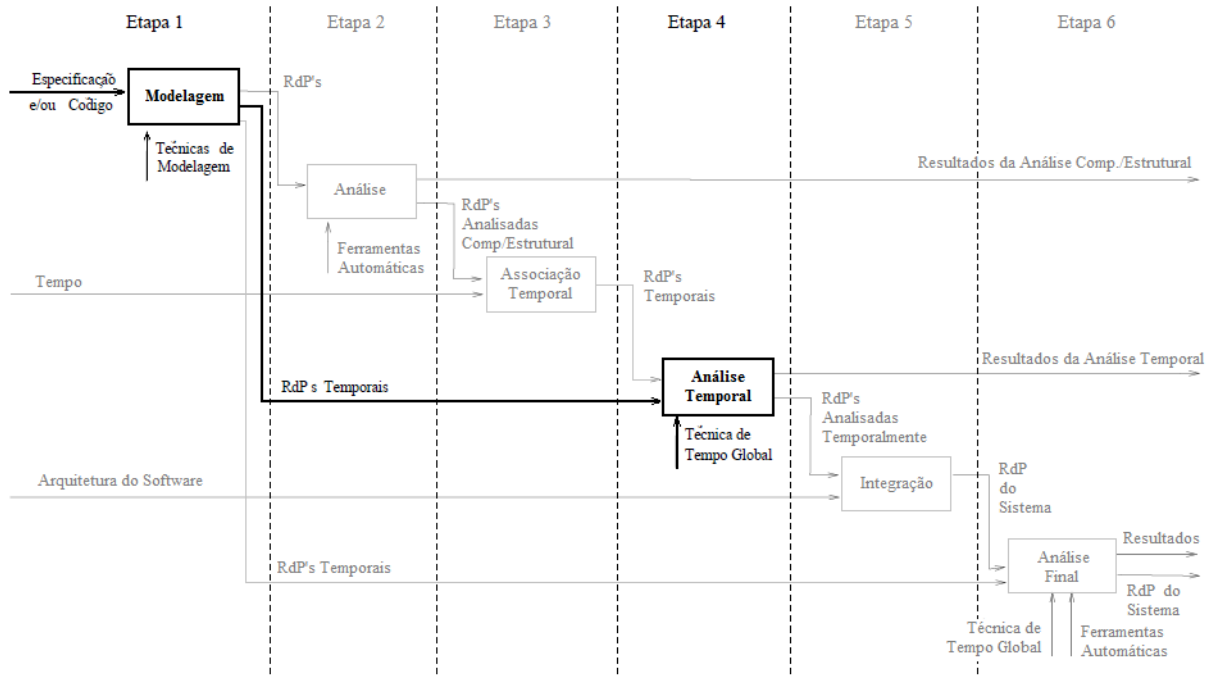

Figura 2.7: Visão Geral do método de verificação por tempo global [40].

Na figura 2.7, a etapa 1 corresponde à modelagem do software embarcado em RdP.

Nesta modelagem são considerados condições e eventos, modelados por lugares e transições, respectivamente. As análises de bloqueio e alcançabilidade para a verificação da estrutura da RdP são realizadas na etapa 2 usando a ferramenta TINA [28]. É possível associar tempo à RdP apenas na etapa 3 ou, da etapa 2, passar diretamente para a etapa 4. Na etapa 4 é aplicada a ferramenta GTT [41] para a análise de tempo global [40] através da análise de cenários e roteiros com classes equivalentes. As RdP's analisadas nesta fase representam módulos do sistemas, funções ou métodos do código. Para realizar uma análise intervalar sobre uma RdP que representa inteiramente um sistema é necessária a integração destas redes. Esta integração é realizada na etapa 5. Por fim na etapa 6, o método de Peres [40] propõe que seja feita a verificação intervalar sobre essa única RdP.

Considerando o algoritmo de Peres [40, 41], este trabalho é baseado na modelagem de software embarcado e sua análise intervalar, representadas nas etapas 1 e 4 respectivamente (figura 2.7).

2.7 Considerações do Capítulo

O Desenvolvimento Orientado a Modelos (MDD) é uma abordagem integrada para a arquitetura, desenvolvimento, testes e implantação de sistemas complexos e de alto desempenho [45]. Segundo Kleppe et al. [26], van Deursen e Klint [54], Bhanot et al. [8] e Mernik et al. [32], esta abordagem possui algumas vantagens, tais como: produtividade, portabilidade, interoperabilidade, manutenção, documentação, comunicação, reutilização, verificações, otimizações e correções.

A Linguagem Unificada de Modelagem, conhecida como UML, é uma linguagem gráfica padrão para especificar, visualizar, documentar e construir artefatos de sistemas de software [9]. Sua aplicação consiste na modelagem de sistemas orientado a objeto.

Rede de Petri é um modelo gráfico e matemático utilizado para representar sistemas com características concorrentes, assíncronas, distribuídas, paralelas, não determinísticas e estocásticas [33, 30]. Identificação prematura de problemas e a possibilidade de análise e verificação de propriedades são algumas das vantagens de modelar sistemas em redes de Petri.

Um sistema de computação com uma finalidade específica e limitado a recursos e funcionalidades de hardware e software é chamado de sistema embarcado [34].

Segundo Murata [33], a análise e verificação de propriedades comportamentais e estruturais de sistemas dinâmicos são permitidas através do uso de redes de Petri como estrutura formal de representação de modelagem. Também é possível através de algumas extensões das redes de Petri, a representação de características temporais dos sistemas.

A análise intervalar de redes de Petri tem como principal objetivo verificar se um determinado estado será alcançado, dentro das restrições intervalares especificadas na rede [40]. Análises estruturais, comportamentais e temporais, são os modos de avaliação da representação do software embarcado por rede de Petri.

CAPÍTULO 3

CONTRIBUIÇÃO

Apesar das vantagens em modelar um software em RdP, este tipo de modelagem não é amplamente utilizado pela indústria como o diagrama de sequência UML. Por este motivo, tornou-se necessária a implementação de uma transformação de modelos, onde o projetista tem a comodidade de modelar o sistema em uma linguagem conhecida pela indústria de computação para um linguagem formal, que possibilita verificações e consequentemente a detecção precoce de erros em toda a modelagem.

A transformação de modelos realizada neste trabalho é representada na figura 3.1 pela fase "Transformação de modelos". Nesta fase transforma-se um diagrama de sequência UML com restrições de tempo e energia em uma rede de Petri temporal e realiza-se três atividades: adaptações dos metamodelos, definição e implementação das transformações. A primeira atividade teve como objetivo realizar adaptações nos metamodelos de diagrama de sequência e de rede de Petri temporal para que ambos contenham restrições de tempo e energia. Esta atividade é apresentada na seção 3.1. A segunda atividade define as transformações dos elementos do diagrama de sequência em rede de Petri temporal e é descrita na seção 3.2. Na terceira atividade foram implementadas as regras de transformação de modelos seguindo as transformações definidas na segunda etapa e descrita na seção 3.3.1. Ainda na figura 3.1, é ilustrado o contexto da conversão de formatos *.xmi* para *.net* (da ferramenta TINA), realizado neste trabalho e apresentado na seção 3.3.2.

A figura 3.2 ilustra de forma simplificada como é realizada a transformação do modelo que baseia-se na arquitetura de três camadas da transformação de modelos, apresentado na seção 2.1.1. Na parte inferior da figura, localizam-se os modelos que se pretende transformar, neste caso "Diagrama de sequência UML/MARTE" (SD) e "Rede de Petri temporal" (RdP-T) que estão em conformidade com os metamodelos respectivos a cada modelo, representados pelos elementos "Metamodelo de diagrama de sequência UML/-

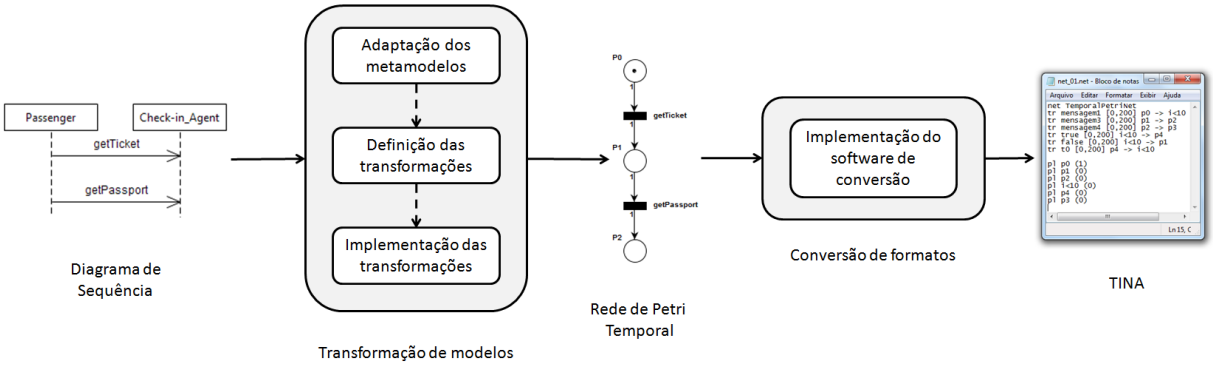

Figura 3.1: Visão geral do trabalho realizado

MARTE" e "Metamodelo de rede de Petri temporal". As regras de transformação são representadas pelo elemento "UMLSD2TPN.atl" que deve estar em conformidade com uma linguagem de transformação, neste caso "ATL". Esta linguagem juntamente com "Metamodelo de diagrama de sequência UML/MARTE" e "Metamodelo de rede de Petri temporal" devem estar em conformidade com um metametamodelo, neste caso o *Ecore* [12].

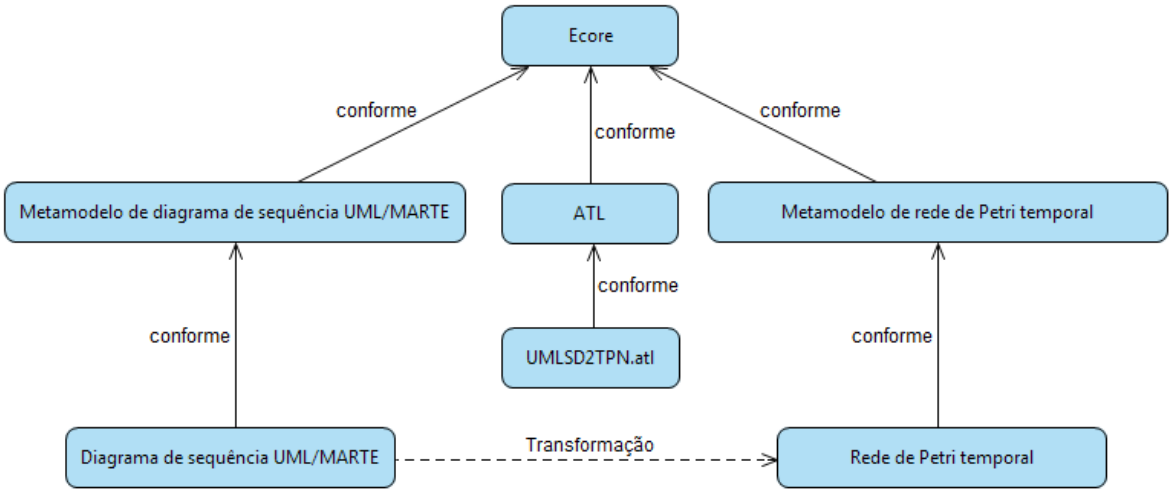

Figura 3.2: Visão global da transformação de DS para RdP-T utilizando ATL. Adaptado de LINA & INRIA ATLAS group [5].

3.1 Adaptações nos Metamodelos

Para a realização deste trabalho, foram levantados metamodelos de diagrama de sequência UML e rede de Petri temporal. Os metamodelos de diagrama de sequência estudados foram: Garousi et al. [21], Thongmak e Muenchaisri [53], Shen et al. [44], Ameedeen

et al. [2] e *Object Management Group (OMG)* [35]. Os metamodelos de rede de Petri temporal analisados foram: Breton e Bézivin [11], Ameedeen et al. [2] e Combemale et al. [14, 15].

Foi identificado, após o estudo dos metamodelos DS e RdP-T, que nenhum dos metamodelos de DS contemplam os dois tipos de restrições, de tempo e de energia. Optou-se então pela adaptação do metamodelo UML proposto pelo OMG [35]. Esta escolha se deu pelo fato do OMG ser o responsável pelo desenvolvimento da própria UML.

Em nenhum dos metamodelos de RdP-T estudados existe definição de restrição de energia. Apenas o metamodelo proposto em [15] contém restrição de tempo, por este motivo foi o escolhido para a adaptação e utilização no projeto.

As seções 3.1.1 e 3.1.2 abordam, respectivamente, os metamodelos de diagrama de sequência e de rede de Petri temporal selecionados.

3.1.1 Metamodelo de Diagrama de Sequência

As restrições de tempo e energia são realizadas devido a utilização do perfil MARTE. Neste trabalho optou-se por representar estas restrições de sete maneiras distintas no DS UML [35]. São elas: na troca de uma mensagem específica, representada no elemento *message*; no tempo total de execução de uma determinada ação, representada no elemento *execution specification*; na execução dos elementos internos de um *combined fragment* dos tipos *Option, Alternatives, Parallel, Loop* e *Resource Usage*, representados nos elementos *interaction operands* e *combined fragments*. Para a representação destas restrições, realizou-se adaptações em alguns pontos do metamodelo proposto pelo OMG. Estas adaptações são apresentadas neste trabalho nas figuras 3.3, 3.4 e 3.5. Os elementos originais da especificação OMG estão representados nas figuras em cinza claro e os novos elementos em cinza escuro.

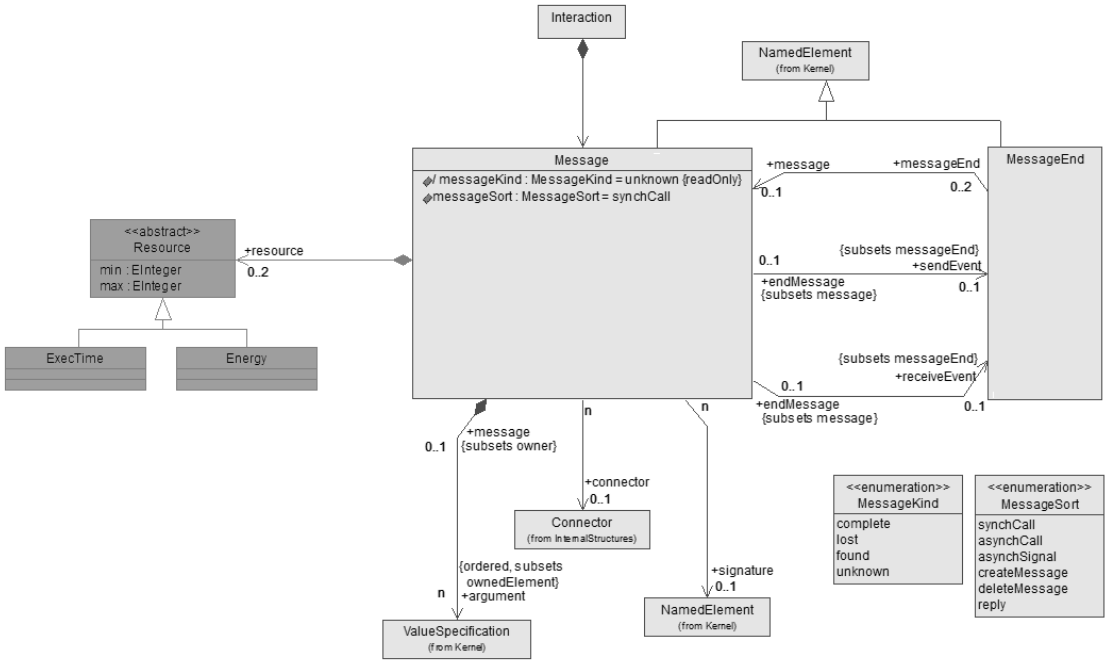

Figura 3.3: Adaptação realizada no metamodelo para representar as restrições contidas em uma mensagem. Adaptado de OMG [35].

As classes *ExecTime* e *Energy* são responsáveis respectivamente por conter as restrições de tempo e energia. Ambas as classes herdam da classe abstrata *Resource* seus valores mínimo e máximo. Esta classe *Resource* é vinculada às classes *Message, ExecutionSpecification* e *InteractionOperand* para que seja possível associar restrições às mensagens, linhas de vida de um determinado objeto e a um bloco de execução. As figuras 3.4 ilustra a adaptação realizada na classe *ExecutionSpecification*.

A figura 3.5 apresenta também as adaptações realizadas no metamodelo para que seja possível a representação de restrições associadas ao elemento *combined fragment*. Para a criação de um novo *combined fragment*, a opção *resourceUsage* foi adicionada à classe *InteractionOperatorKind*.

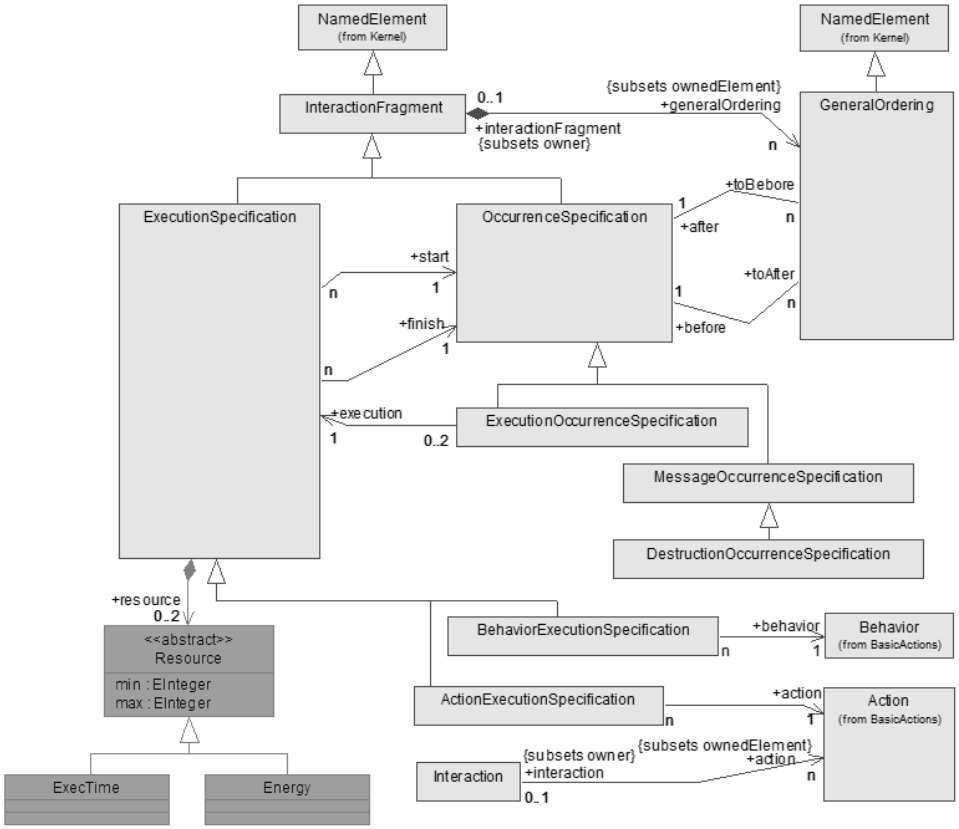

Figura 3.4: Adaptação realizada no metamodelo para representar as restrições contidas em um tempo de execução. Adaptado de OMG [35].

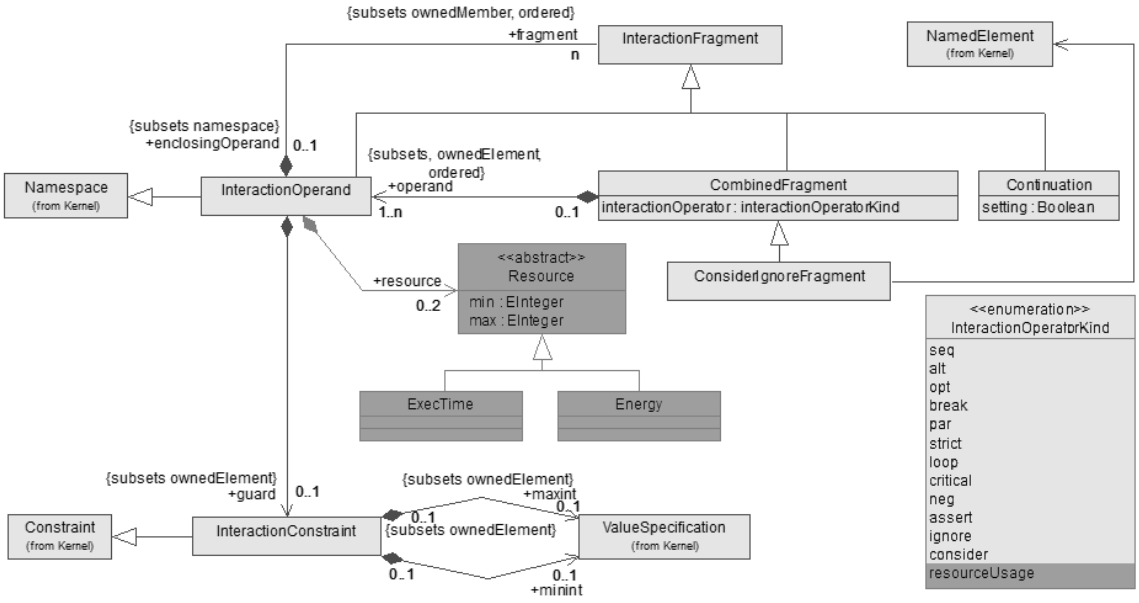

Figura 3.5: Adaptação realizada no metamodelo para representar as restrições contidas em um conjunto de atividades. Adaptado de OMG [35].

3.1.2 Metamodelo de rede de Petri Temporal

O metamodelo correspondente à RdP-T é apresentado na figura 3.6. O elemento *Transition* representado na cor cinza escuro foi o único que sofreu alteração. O mesmo já continha os atributos *min_time* e *max_time*. Foi necessário adicionar a restrição de energia, representado pelos atributos *min_energy* e *max_energy*.

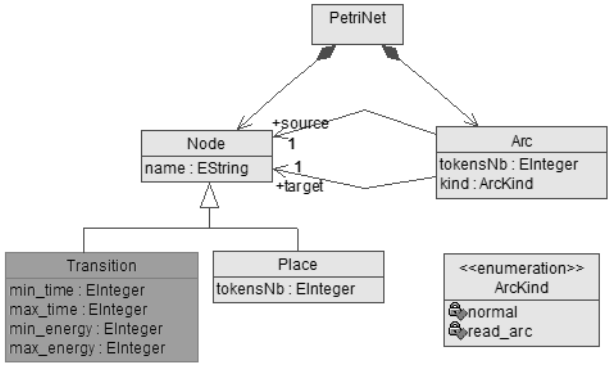

Figura 3.6: Adaptação realizada no metamodelo para representar as restrições de tempo e energia em rede de Petri temporal. Adaptado de OMG [15].

3.2 Definição das Transformações

Os elementos do DS UML [35]: *message, execution specification* e *combined fragment* sofreram adaptações a fim de contemplarem restrições de tempo e energia. As seções 3.2.1, 3.2.2 e 3.2.3 apresentam respectivamente cada uma destas adaptações de restrições e definições de transformações.

3.2.1 *Message* (Mensagem)

O primeiro exemplo de transformação é ilustrado na figura 3.7 (a). Nesta figura tem-se o envio de uma mensagem, representada por uma seta horizontal, entre um objeto emissor, *Objeto1*, a um objeto receptor, *Objeto2*. Na figura 3.7 (b), observa-se uma sub-rede de Petri básica, onde tem-se dois lugares, *P0* e *P1*, uma transição, *mensagem1* e dois arcos: um que liga *P0* a *mensagem1*, e outro que liga a *mensagem1* a *P1*.

Neste exemplo, a transformação ocorre da seguinte maneira: existindo uma troca de

mensagem, a mesma é convertida em uma sub-rede de Petri contendo respectivamente, *lugar*, *arco*, *transição*, *arco* e *lugar*. A transição recebe o mesmo nome da mensagem trocada entre os objetos e os lugares recebem nomes indicadores de ordem: *P0, P1, P2, .., Pn*.

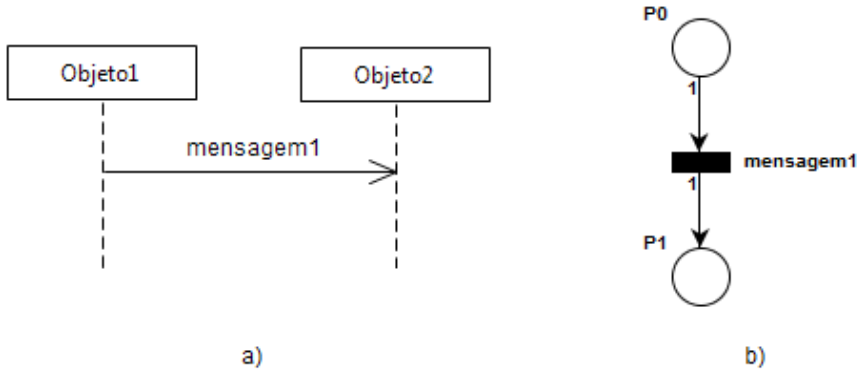

Figura 3.7: Transformação aplicada à mensagem.

O exemplo ilustrado na figura 3.8 (a), é uma troca de mensagem com restrições de tempo e energia. Estas restrições são representadas pela utilização do perfil MARTE e definidas pelo estereótipo $<< ResourceUsage >>$ acompanhado da palavra *execTime* e *energy*, seguidas de seus respectivos valores máximos e mínimos de tempo de execução e energia consumida que devem ser respeitados. Ao disparar uma mensagem, os valores iniciais, tanto de tempo (S_0) quanto de energia (J_0) são atribuídos. Assim que o objeto receptor recebe a mensagem, os valores de tempo final (S) e de energia final (J) são atribuídos. A sub-rede de Petri representada na figura 3.8 (b) é composta por dois lugares, *P0* e *P1*, uma transição, *mensagem1* e dois arcos, um que liga *P0* a *mensagem1* e outro que liga *mensagem1* a *P1*. As restrições de tempo e energia são associadas à transição, seguindo a regra que a primeira tupla representa a limitação de tempo e a segunda de energia.

A transformação de modelos de mensagem com restrição ocorre da seguinte maneira: existindo uma troca de mensagem contendo restrições de tempo e energia associadas à ela, esta é convertida em uma sub-rede de Petri contendo respectivamente, *lugar*, *arco*, *transição*, *arco* e *lugar*. A transição recebe o mesmo nome da mensagem trocada entre os objetos em questão e os valores mínimo e máximo referente às restrições associadas a ela.

Os lugares recebem nomes indicadores de ordem: *P0, P1, P2, .., Pn.*

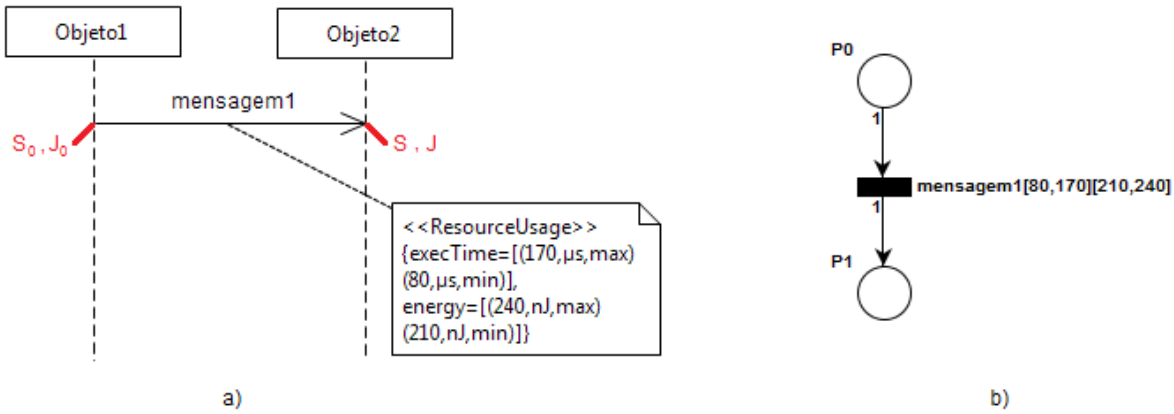

Figura 3.8: Transformação aplicada à mensagem contendo restrições de tempo e energia.

3.2.2 *Execution Specification* (Especificação de Execução)

A modelagem do tempo ou energia necessários para a execução de uma determinada ação em uma linha de vida é possível graças à definição de tempo e energia inicial e final.A figura 3.9 (a) apresenta uma aplicação de MARTE no elemento *execution specification* com o intuito de realizar esta limitação. Neste exemplo, os valores iniciais de tempo (S_0) e energia (J_0) são atribuídos juntamente com o elemento *execution occurrence specification* de ínicio. Após a execução total do elemento *execution specification*, o mesmo finalizará com o elemento *execution occurrence specification* de fim. Este último é responsável por atribuir os valores finais a tempo (S) e energia (J). A sub-rede de Petri representada na figura 3.9 (b) é formada por dois lugares, *P0* e *P1*, uma transição, *T0* e dois arcos: um que liga *P0* a *T0* e outro que liga *T0* a *P1*. As restrições *[80,170]* e *[210,240]* são associadas à.

Para que ocorra a transformação, é necessário que o elemento *execution specification* contenha uma restrição associada, como no exemplo da figura 3.9 (a). Neste exemplo, este elemento é transformado em uma sub-rede de Petri temporal, representado na figura 3.9 (b). Esta sub-rede é formada pelos elementos *lugar, arco, transição, arco* e *lugar*. A transição recebe os nomes indicadores de ordem: *T0, T1, T2, .., Tn* e os valores mínimo e máximo referente às restrições associadas ao elemento *execution specification*. Os lugares

recebem nomes indicadores de ordem: *P0, P1, P2, .., Pn.*

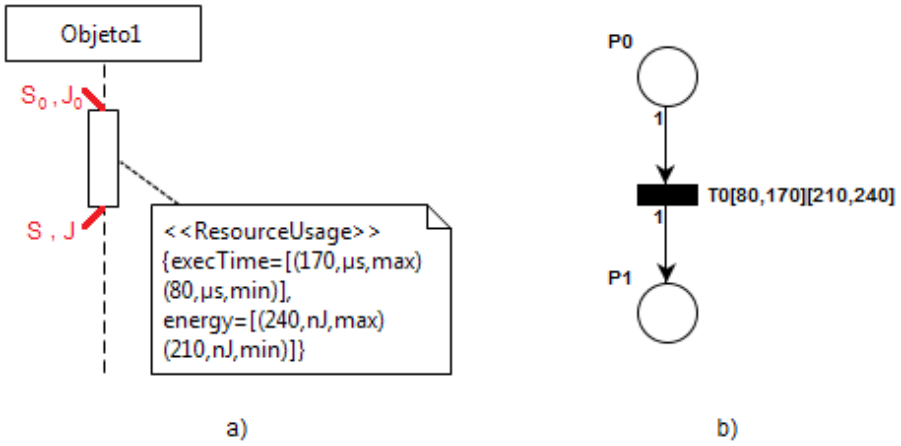

Figura 3.9: Transformação aplicada ao elemento *Action Execution Specification* contendo restrições de tempo e energia.

3.2.3 *Combined Fragment* (Fragmento Combinado)

Neste trabalho são definidos e implementados um total de cinco *combined fragment: Option, Alternatives, Parallel, Loop* e *Resource Usage*. Destes, quatro são especificados pelo OMG [35], escolhidos devido ao elevado número de utilização dos mesmos nos diagramas de sequência, e um proposto por este trabalho. Ressalta-se que os *combined fragments* existentes não contém restrições associadas. Explorou-se a adaptação destes elementos para que seja possível tal representação. Os valores iniciais de tempo e energia são representados nas figuras por S_0 e J_0 respectivamente. Do mesmo modo, os valores finais são dados por S e J.

Neste trabalho a representação dos elementos internos aos *combined fragments* são agrupados em apenas um lugar na rede de Petri temporal.

***Option* (Opção)** Denomina que o *combined fragment* representa uma escolha de comportamento definida em um *interaction operand "Opt_fragment"* na figura 3.10 (a), que é executado no caso de uma condição *Arg* ser verdadeira. No caso da condição ser falsa, nada é executado [35]. *Option* pode ser comparado em linguagem de programação, à um IF simples, sem o fluxo ELSE. A sub-rede de Petri temporal respectiva é apresentada na figura 3.10 (b) e é formada por três lugares: *Arg, Opt_fragment* e *P0*, três transições: *True,*

False e *T0* e seis arcos: o primeiro liga *Arg* a *True*, o segundo liga *True* a *Opt_fragment*, o terceiro *Opt_fragment* a *T0*, o quarto *T0* a *P0*, o quinto *Arg* a *False* e o último liga *False* a *P0*.

O lugar de início na sub-rede de Petri recebe como nome a condição especificada no diagrama de sequência seguido de duas transições: *True* e *False*, as quais definem os dois possíveis caminhos a serem seguidos. Os elementos no interior do bloco *opt*, representados na figura 3.10 (a) pelo elemento *Opt_fragment*, são representados pelo lugar *Opt_fragment* na sub-rede de Petri, e as restrições de tempo e energia associadas ao *combined fragment* são representadas na transição *T0*.

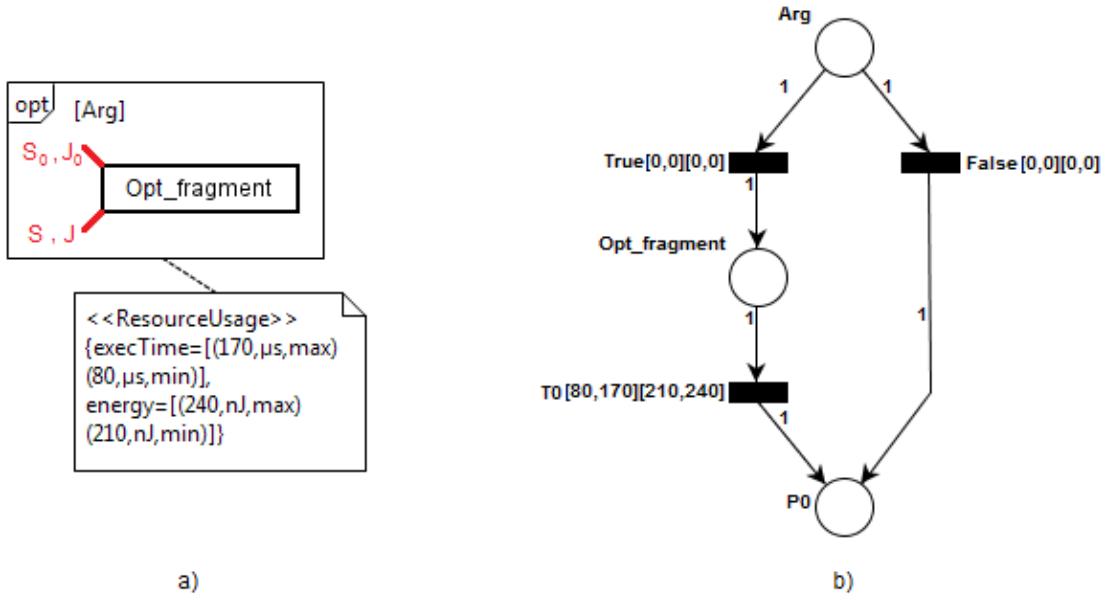

Figura 3.10: Transformação aplicada à estrutura de decisão simples.

***Alternatives* (Alternativas)** Esta estrutura representa dois ou mais fluxos alternativos e é ilustrada na figura 3.11 (a). É necessário que haja um *interaction fragment* chamado ELSE, para que, se nenhuma das condições anteriores forem atendidas, algo seja executado [35]. Este *combined fragment* possui uma restrição para cada *interaction operand* pertencente a ele. A sub-rede de Petri temporal respectiva é apresentada na figura 3.11 (b) e é iniciada por um lugar *P0* conectado a N transições, onde N é definido pelo número de *interaction operands*. Cada transição recebe o nome da condição e restrições zeradas, pois representam apenas uma escolha de fluxo. Estas transições são associadas

a um lugar que representa as execuções internas desses operandos. Estes conectados a uma transição respectiva, contento as restrições de tempo e energia. Para finalizar, estas transições são conectadas a um único lugar final.

A transformação do elemento *combined fragment* do tipo *Alternatives*, consiste em transformar cada *interaction operand* em uma transição, nomeando-a com o nome da própria condição do *interaction operand* e estabelecendo a elas restrições zeradas, pois o consumo de tempo e energia só irá ocorrer após a definição do fluxo a ser seguido pela marca. Os elementos que serão executados caso a condição seja satisfeita no diagrama de sequência, são convertidos em lugares associados a cada transição respectiva na sub-rede de Petri. Cada um destes lugares são respectivamente associados a uma transição particular, contendo as restrições estabelecidas para cada *interaction operand* do *combined fragment*. Ao fim, estas transições são conectadas a um lugar final comum entre elas, neste caso, chamado de *P1*.

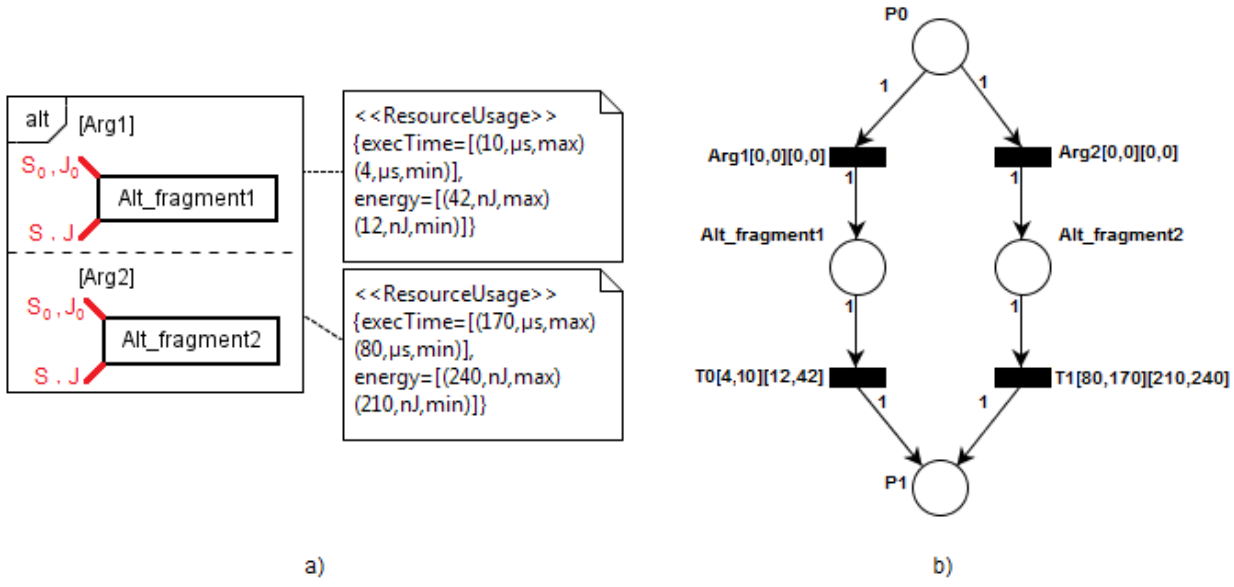

Figura 3.11: Transformação aplicada à estrutura de decisão composta.

***Parallel* (Paralelo)** Define que dois ou mais *interaction operands* sejam executados em paralelo, não necessitando de qualquer condição para que ocorram [35]. A figura 3.12 (a) apresenta uma execução paralela em DS onde o elemento *Par_fragment1* e *Par_fragment2* são executados paralelamente, sendo possível restringir tempo e ener-

gia. A sub-rede de Petri temporal respectiva é apresentada na figura 3.12 (b) e inicia com um lugar *P0* conectado a uma transição *T0*, esta transição recebe restrições zeradas e é conectada a N lugares, onde N é o número de *interaction operands* presentes no *combined fragment*. Estes lugares recebem o nome indicados de ordem: *Par_fragment1, Par_fragment2, Par_fragmentN*, em referência aos elementos internos de cada operando e são associados a uma transição comum entre eles, esta recebe as restrições impostas no DS e são conectadas a um único lugar final.

Ao verificar a existência de um *combined fragment* do tipo *Parallel*, o mesmo é transformado em um lugar e uma transição, interligados por um arco. Esta transição é conectada a N lugares, onde N é definido pelo número de operandos do diagrama de sequência. Para representar o fim da execução paralela, é definida uma transição que recebe as restrições impostas do *combined fragment* e que todos os lugares referentes a cada operando são associados à ela que por sua vez é conectada a um único lugar final.

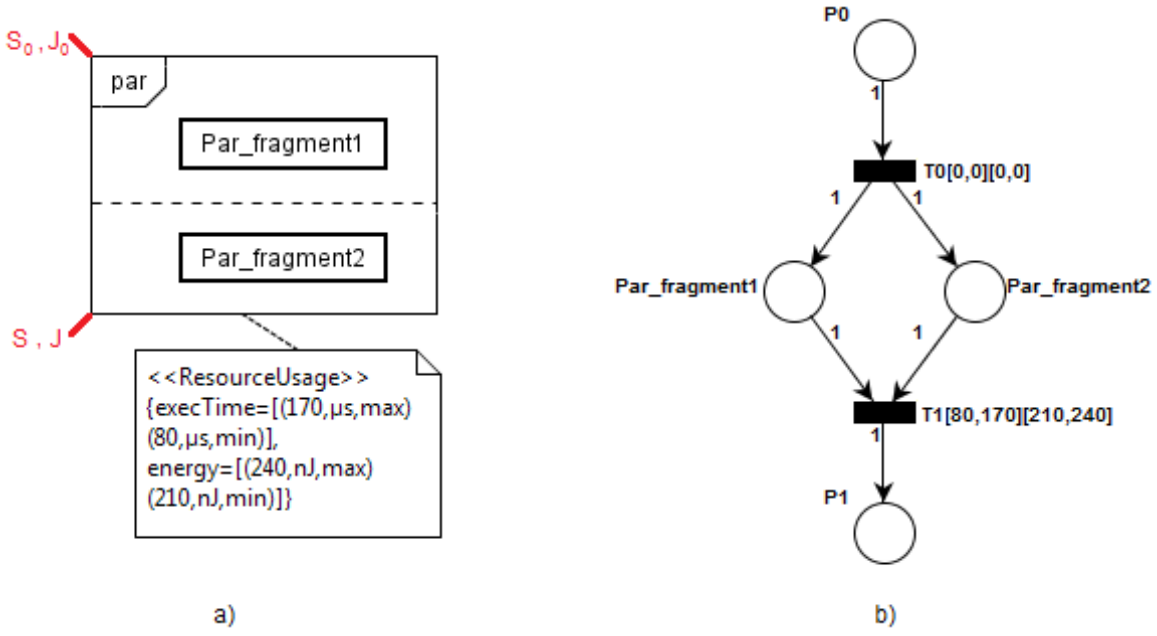

Figura 3.12: Transformação aplicada à execução paralela.

***Loop* (Laço)** Representa um laço, o qual possui uma condição, identificada na figura 3.13 (a) pelo elemento *Arg*. Enquanto essa condição for satisfeita, são executados os elementos presentes no interior do *combined fragment*, identificados pelo elemento

Loop_fragment. Este pode conter restrições associadas à ele. A sub-rede de Petri temporal está ilustrada na figura 3.13 (b). Esta sub-rede é formada por três lugares: *Arg, Loop_Fragment* e *P0*, três transições: *True, False* e *T0* e seis arcos: o primeiro liga *Arg* a *True*, o segundo liga *True* a *Loop_Fragment*, o terceiro *Loop_Fragment* a *T0*, o quarto *T0* a *Arg*, o quinto *Arg* a *False* e o sexto *False* a *P0*.

A transformação inicia no elemento responsável pela representação da condição. O lugar na RdP-T recebe a condição especificada no *combined fragment* do DS. Este lugar é associado a duas transições, *True* e *False*. Os comandos internos do laço de repetição, representados pelo elemento *Loop_fragment* no DS, são convertidos no elemento lugar chamado de *Loop_Fragment* na sub-rede de Petri temporal e suas restrições são atribuídas à transição *False*.

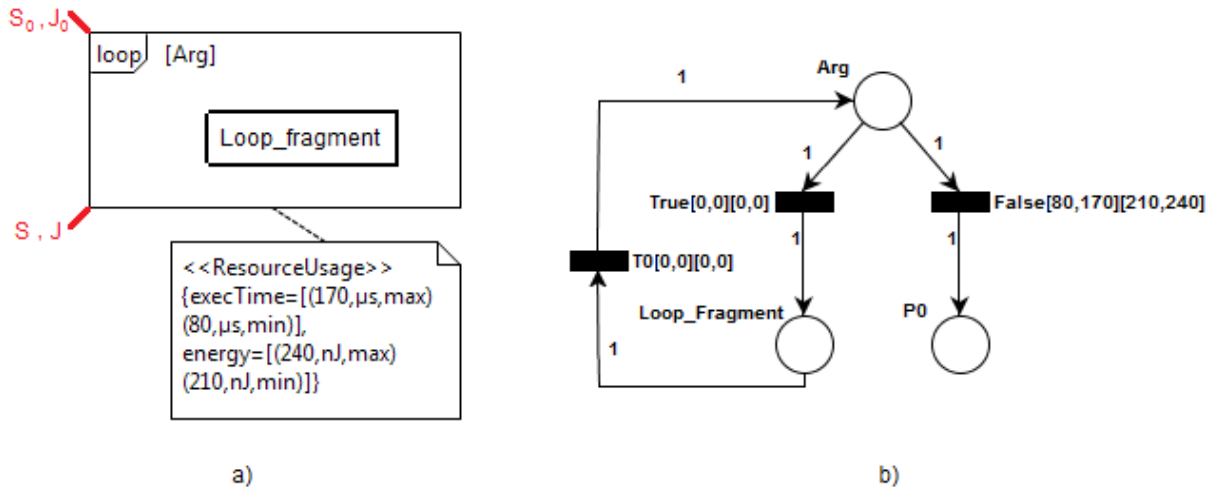

Figura 3.13: Transformação aplicada a laços de repetição.

***Resource Usage* (Uso de Recurso)** Este *combined fragment* é proposto neste trabalho e é identificado com a sentença *resourceUsage* descrita no canto superior esquerdo como representado na figura 3.14 (a). Com este *combined fragment* é possível definir restrições não apenas a um elemento específico, mas sim a um grupo de elementos. Ou seja, todos os comandos internos a este bloco de execução, identificado na figura 3.14 (a) pelo elemento *(ResourceUsage_fragment)*. Ao iniciar a execução do bloco, os valores iniciais de tempo e energia são definidos, ao fim da execução do mesmo, são estabelecidos os valores finais das restrições. A sub-rede de Petri respectiva ao *combined fragment*

Resource Usage é ilustrada na figura 3.14 (b) e é formada apenas por dois lugares, *P0* e *P1*, uma transição, *T0* e dois arcos, o primeiro liga *P0* a *T0* e o segundo liga *T0* a *P1*.

A transformação do elemento *combined fragment* do tipo *Resource Usage* envolve o agrupamento de todos os elementos internos, representado na figura 3.14 (a) pelo elemento *ResourceUsage_fragment*, em apenas uma transição que recebe o nome indicado de ordem: *T0, T1, T2, .., Tn*. Esta sub-rede é ilustrada na figura 3.14 (b).

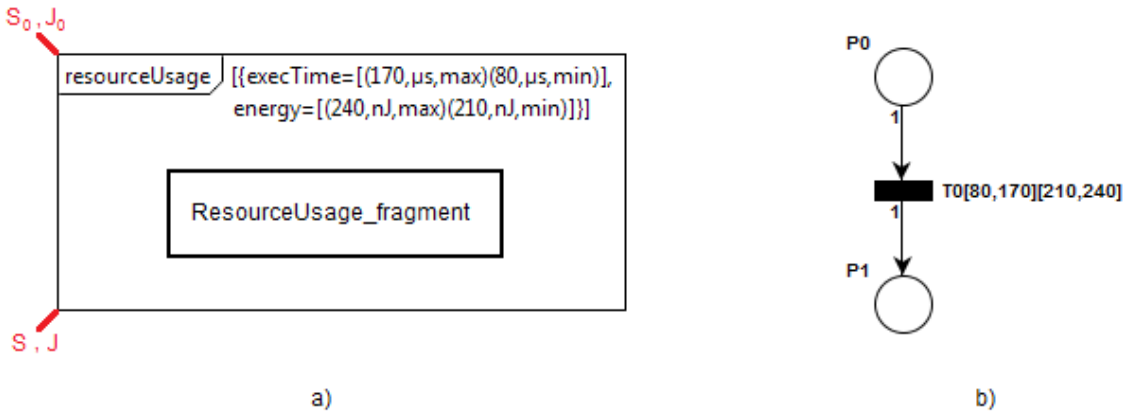

Figura 3.14: Transformação aplicada ao *combined fragment "Resource Usage"*.

3.3 Implementação

A implementação realizada consiste de uma transformação de modelos de diagrama de sequência UML em rede de Petri temporal através da ATL e é descrita na seção 3.3.1. Uma segunda implementação foi feita relativa à conversão de uma rede de Petri temporal em formato *.xmi* para uma rede de Petri temporal em formato *.net*, padrão da ferramenta TINA [28]. Esta conversão é apresentada na seção 3.3.2.

3.3.1 Projeto "UMLSD2TPN"

A transformação de diagrama de sequência UML em rede de Petri temporal é realizada pela definição de regras ATL. Dado o modelo de origem, estas regras são responsáveis pela conversão automática em um modelo alvo.

Para o desenvolvimento do projeto, utilizou-se o *Eclipse Modeling Framework (EMF)* [12]. A figura 3.15 ilustra a raiz do projeto *"UMLSD2TPN"* que possui em seu diretório,

quatro pastas que servem para uma melhor organização dos arquivos que o compõem. São elas: *ATLFile, Models, TimePetriNet* e *UML*.

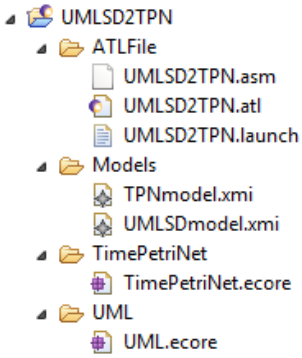

Figura 3.15: Projeto UMLSD2TPN.

As pastas *"UML"* e *"TimePetriNet"*, contém os metamodelos de diagrama de sequência e rede de Petri temporal respectivamente, ambos apresentados na seção 3.1. Estes metamodelos são ilustrados na figura 3.15 pelos arquivos *"UML.ecore"* e *"TimePetriNet.ecore"*.

Tanto o modelo de entrada da transformação, modelo de origem, quanto o modelo de saída, modelo alvo, localizam-se na pasta *"Models"*. O arquivo *"UMLSDmodel.xmi"* é referente ao modelo de diagrama de sequência que é o modelo de origem e o arquivo *"TPNmodel.xmi"* é relativo à rede de Petri temporal que é o modelo alvo, o qual é gerado após a execução da transformação.

Por fim, a pasta *"ATLFile"* é o local onde se encontra o arquivo *"UMLSD2TPN.atl"*, este responsável por conter as regras ATL, principal elemento da transformação em questão.

3.3.2 Conversor XMI para TINA

Para que a RdP-T seja usada como entrada das ferramentas TINA e GTT, deve estar em formato *".net"* [28]. Por isso, desenvolveu-se um programa conversor na linguagem de programação JAVA [23] chamado *"XMI to TINA Converter"*. Este é capaz de transformar o arquivo no formato *.xmi* – gerado pela transformação ATL – para o formado *.net*.

O programa desenvolvido é dividido em três camadas. São elas: *View, VO* e *BusinessRule*. Na camada *View*, encontram-se classes de interação com o usuário, como por

exemplo, a *Tela XMItoTINAView* e a *Tela About*. Na camada *VO*, localizam-se as classes relacionadas aos elementos com seus respectivos atributos que são necessários para a realização da transformação. Por fim, na camada *BusinessRule*, é encontrada a classe responsável pela regra de negócio do software, chamada de *XMItoTINABR*.

Além destas camadas, o projeto possui outras, tais como: *Image* e *Media*. A primeira armazena as imagens presentes nas telas de interação e a segunda os sons de alerta que são reproduzidos caso ocorram erros ou a transformação feita com sucesso.

A figura 3.16 ilustra as camadas descritas.

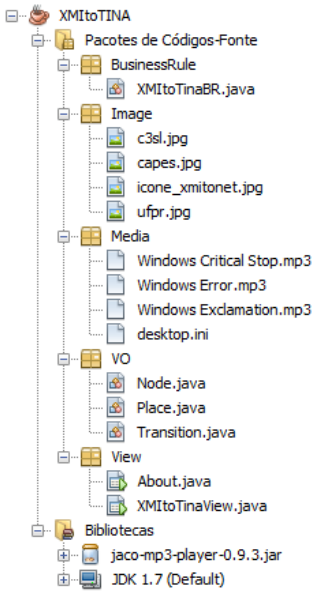

Figura 3.16: Camadas do software *"XMI to TINA Converter"*.

A utilização deste programa encontra-se no apêndice A.

3.4 Considerações do Capítulo

Neste capítulo foram apresentadas as atividades realizadas para o atendimento dos objetivos definidos na seção 1.1 desta monografia.

As adaptações realizadas nos metamodelos de diagrama de sequência UML e rede de Petri temporal apresentadas na seção 3.1 atendem ao objetivo específico 1 da seção 1.1.

Na seção 3.2 foram definidas as representações das restrições nos diagramas de sequência e posteriormente as transformações destes diagramas para rede de Petri temporal. Também

foi proposto um novo *combined fragment* chamado *Resource Usage* que tem por finalidade restringir tempo e energia a todos os seus elementos internos.

A implementação descrita na seção 3.3 foi detalhada em duas etapas. A primeira responsável pela transformação dos modelos que tem como entrada o diagrama de sequência UML e gera um modelo de rede de Petri temporal como saída. A segunda apresenta a conversão de extensões de rede de Petri temporal, de formato *.xmi* para *.net*. Tanto a implementação descrita na seção 3.3 quanto as definições das representações das restrições, descritas na seção 3.2, atendem ao objetivo 2 da seção 1.1.

A transformação implementada gera uma rede de Petri temporal que pode ser analisada com o método de Peres [40] de verificação intervalar de tempo e energia, atendendo ao objetivo 3 da seção 1.1. No próximo capítulo são apresentados experimentos a partir das RdP-T geradas.

CAPÍTULO 4

RESULTADOS

Este capítulo aborda os resultados obtidos de estudos de caso das transformações de modelos e análise intervalar realizadas nos modelos gerados.

Os estudos de caso tem como objetivo validar o projeto *"UMLSD2TPN"* e apresentar a utilização do programa conversor do formato *.xmi* para o formato *.net*, requerido pelas ferramentas TINA/GTT.

Os estudos de caso seguiram os seguintes passos:

1. encontrar na literatura diagramas de sequência que modelam software embarcado;
2. adicionar restrições de tempo e energia quando necessário;
3. realizar a transformação de modelos utilizando a ferramenta desenvolvida neste trabalho e apresentada na seção 3.3.1;
4. converter a rede de Petri gerada para o formato *.net*, usando o programa descrito na seção 3.3.2;
5. analisar a rede de Petri em formato *.net* pela ferramenta GTT [40, 41], como citado na seção 2.6.

As seções 4.1, 4.2 e 4.3 apresentam os resultados destes estudos de casos. O primeiro refere-se a um sistema de elevador [43], o segundo a um pulsoxímetro [4] e o terceiro a um *framework* para simular o comportamento de sistemas embarcados de tempo real [55].

Na seção 4.4 é apresentada uma tabela comparativa entre os modelos utilizados na validação do projeto.

4.1 Estudo de Caso 1 – Sistema de Elevador

Neste primeiro exemplo, a transformação de modelos é aplicada à modelagem de um sistema de elevador proposto por Ribeiro [43]. Esta modelagem está representada na figura 4.1. Este sistema é responsável por interpretar o piso em que o passageiro se encontra e levá-lo até o piso desejado.

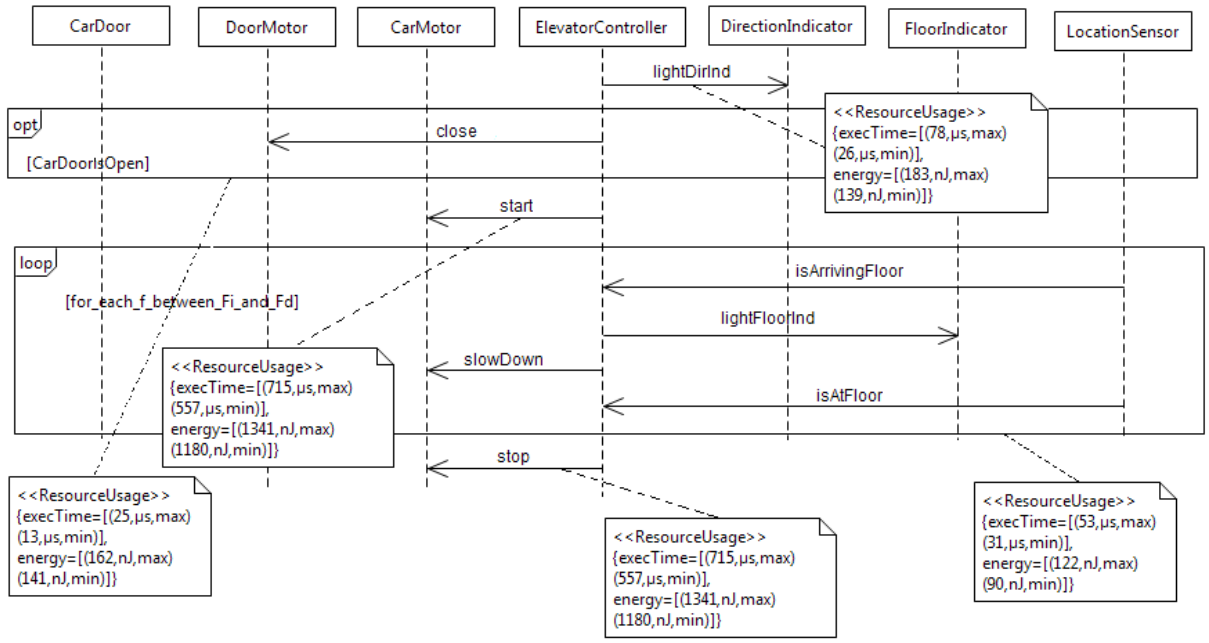

Figura 4.1: Modelagem em DS de um controle de elevador. Adaptado de Ribeiro [43].

Como resultado da transformação de modelos, tem-se a RdP-T respectiva, ilustrada na figura 4.2.

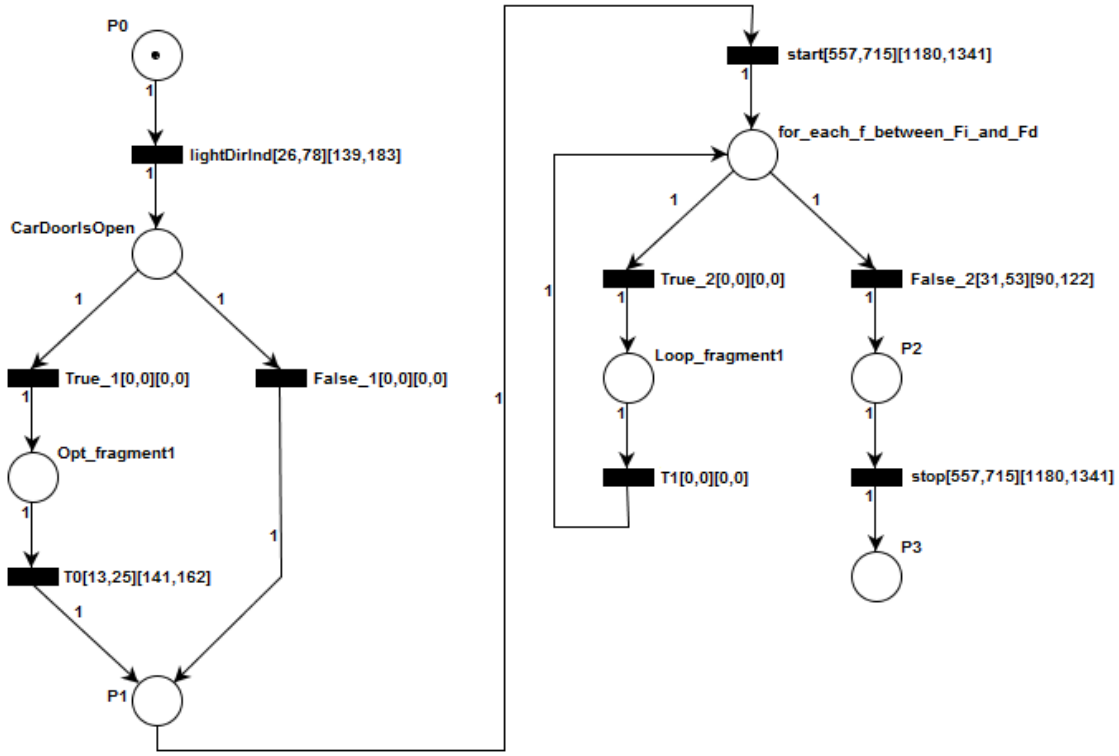

Figura 4.2: Modelagem em RdP-T de um controle de elevador.

Esta RdP-T proveniente da transformação de modelos possui um código correspondente no formato *.xmi* que é convertido em um código *.net* equivalente, esta etapa descrita na seção 3.3.2. A análise realizada na RdP-T não admite que os valores das restrições de tempo e de energia estejam no mesmo código de acordo com a seção 2.3.1, ou seja, tem-se dois códigos em formato *.net*, um contendo apenas restrições de tempo e outro de energia. O código a seguir representa a RdP-T ilustrada na figura 4.2. Este código leva em consideração apenas as restrições de tempo.

```
1 net controle_elevador_tempo
2 tr lightDirInd [26,78] P0 -> CarDoorIsOpen
3 tr start [557,715] P1 -> for_each_f_between_Fi_and_Fd
4 tr stop [557,715] P2 -> P3
5 tr True_1 [0,0] CarDoorIsOpen -> Opt_fragment1
6 tr False_1 [0,0] CarDoorIsOpen -> P1
7 tr T0 [13,25] Opt_fragment1 -> P1
8 tr True_2 [0,0] for_each_f_between_Fi_and_Fd -> Loop_fragment1
9 tr False_2 [31,53] for_each_f_between_Fi_and_Fd -> P2
```

```
10 tr T1 [0,0] Loop_fragment1 -> for_each_f_between_Fi_and_Fd
11
12 pl P0 (1)
13 pl P1 (0)
14 pl P2 (0)
15 pl CarDoorIsOpen (0)
16 pl Opt_fragment1 (0)
17 pl for_each_f_between_Fi_and_Fd (0)
18 pl Loop_fragment1 (0)
19 pl P3 (0)
```

A primeira linha é formada pela representação *net* seguida do nome da rede. Neste caso, *controle_elevador_tempo*. A partir da segunda linha há um detalhamento de todas as transições pertencentes à RdP-T. Estas transições são descritas iniciando com *tr* em referência a própria transição seguida de seu nome, sua restrição, nome dos lugares precedentes e procedentes a esta transição. Em seguida há o detalhamento dos lugares pertencentes a RdP-T. Estes lugares são descritos iniciando com a palavra *pl* seguida do nome do lugar e seu número de marcas.

O código a seguir representa a RdP-T levando em consideração apenas as restrições de energia.

```
1  net controle_elevador_energia
2  tr lightDirInd [139,183] P0 -> CarDoorIsOpen
3  tr start [1180,1341] P1 -> for_each_f_between_Fi_and_Fd
4  tr stop [1180,1341] P2 -> P3
5  tr True_1 [0,0] CarDoorIsOpen -> Opt_fragment1
6  tr False_1 [0,0] CarDoorIsOpen -> P1
7  tr T0 [141,162] Opt_fragment1 -> P1
8  tr True_2 [0,0] for_each_f_between_Fi_and_Fd -> Loop_fragment1
9  tr False_2 [90,122] for_each_f_between_Fi_and_Fd -> P2
10 tr T1 [0,0] Loop_fragment1 -> for_each_f_between_Fi_and_Fd
11
12 pl P0 (1)
13 pl P1 (0)
```

```
14 pl P2 (0)
15 pl CarDoorIsOpen (0)
16 pl Opt_fragment1 (0)
17 pl for_each_f_between_Fi_and_Fd (0)
18 pl Loop_fragment1 (0)
19 pl P3 (0)
```

A partir dos códigos *.net* da RdP-T são realizadas análises intervalares utilizando a ferramenta GTT (em inglês, *Global Time Technique*) [40, 41]. O resultado dessa ferramenta é um grafo de classes que representa a RdP-T de entrada. Cada nó deste grafo representa uma classe $Ck\_n \in C$, cada arco é uma transição disparada na classe Ck_n e que gera uma classe sucessora $C(k+1)\_n$. onde k é o nível do grafo e n é um número identificador da classe, a marcação de rede na classe M_k e o tempo global da classe G. As próximas linhas do nó representam a informação de disparo de uma transição t_i que será usada para a geração da próxima classe no caso de t_i ser disparada: o nome da transição t_i; seu tempo relativo $r_k(t_i)$; seu coeficiente de ajuste $ac_k(t_i)$; e seu tempo global $g_k(t_i)$.

As figuras 4.3 e 4.4 ilustram os grafos de classe referente à RdP-T da figura 4.2. A primeira leva em consideração os intervalos de tempo, o segundo, os intervalos de energia.

Na figura 4.3, o nó *C0_0* é o ponto inicial do grafo. Este possui um intervalo de tempo relativo de $R = [26, 78]$ e dispara para a transição *lightDirInt*. A partir dessa classe é possível disparar para a transição *False_1* ou *True_1*. Seguindo a sequência de disparos pela transição *False_1*, tem-se *start, True_2* e *T1*, com tempo global $G = [583, 793]$, apresentado na classe *C5_8*. Em contrapartida, seguindo a sequência de disparos pela transição *True_1*, tem-se a sequência: *T0, start* e *True_2*, finalizando na classe *C5_9*, com tempo global $G = [596, 818]$.

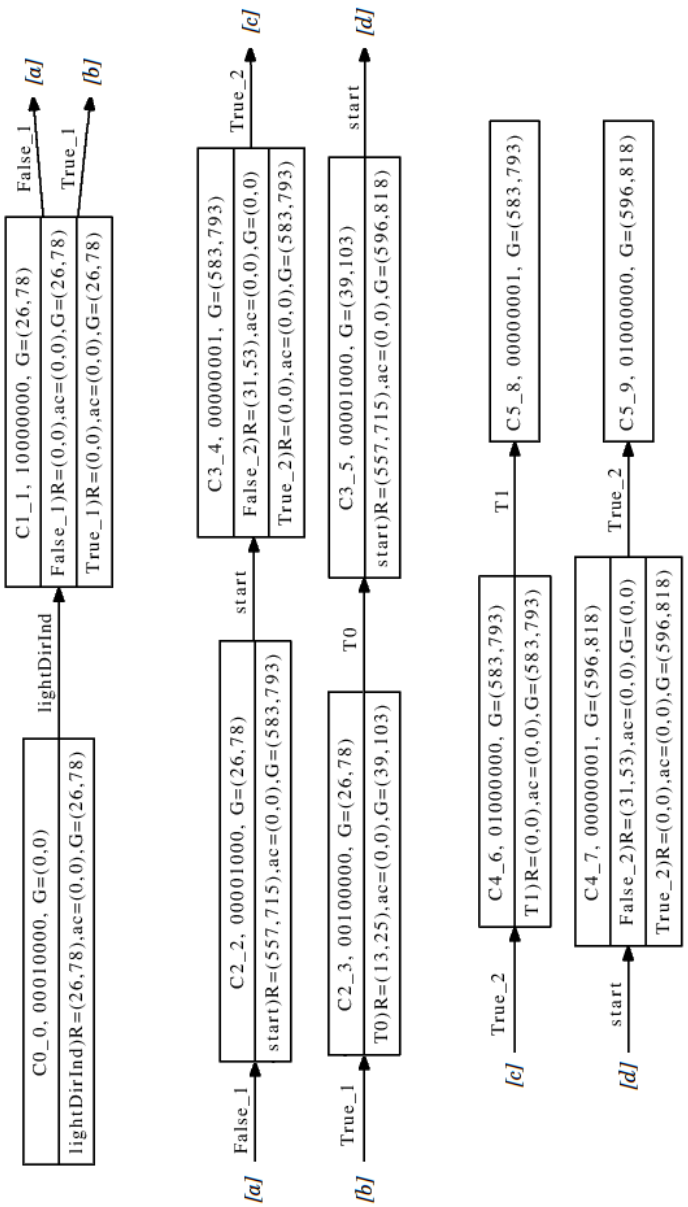

Figura 4.3: Grafo de classes de 5 níveis, da RdP-T da figura 4.2 representando tempo.

A seguir, a figura 4.4 ilustra a análise intervalar realizada. O ponto inicial do grado é representado pelo nó *C0_0*. Este possui um intervalo de tempo relativo de $R = [139, 183]$ e dispara para a transição *lightDirInt*. A partir dessa classe é possível disparar para a transição *False_1* ou *True_1*. Seguindo a sequência de disparos pela transição *False_1*, tem-se *start, True_2* e *T1*, com tempo global $G = [1319, 1524]$, apresentado na classe *C5_8*. Em contrapartida, seguindo a sequência de disparos pela transição *True_1*, tem-se a sequência: *T0, start* e *True_2*, finalizando na classe *C5_9*, com tempo global $G = [1460, 1686]$.

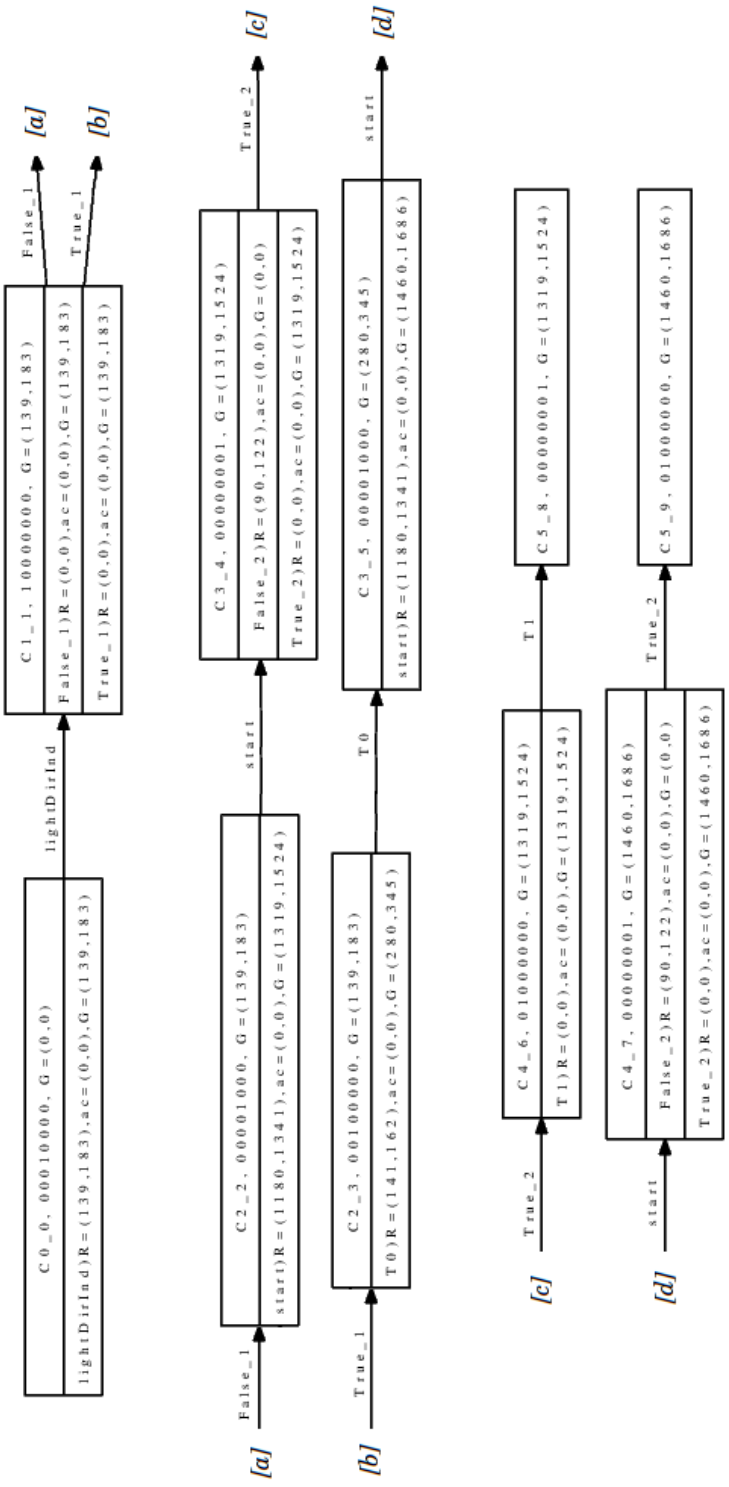

Figura 4.4: Grafo de classes de 5 níveis, da RdP-T da figura 4.2 representando energia.

4.2 Estudo de Caso 2 – Pulsoxímetro

O segundo estudo de caso é referente a um pulsoxímetro [4]. Este dispositivo eletrônico é responsável por medir a saturação de oxigênio no sangue por meio de um método não-invasivo. Um oxímetro de pulso pode ser empregado quando um paciente é sedado durante os procedimentos cirúrgicos. O aparelho verifica a saturação de oxigênio para garantir um nível aceitável de oxigenação do sangue. A figura 4.5 ilustra a modelagem em DS de um pulsoxímetro.

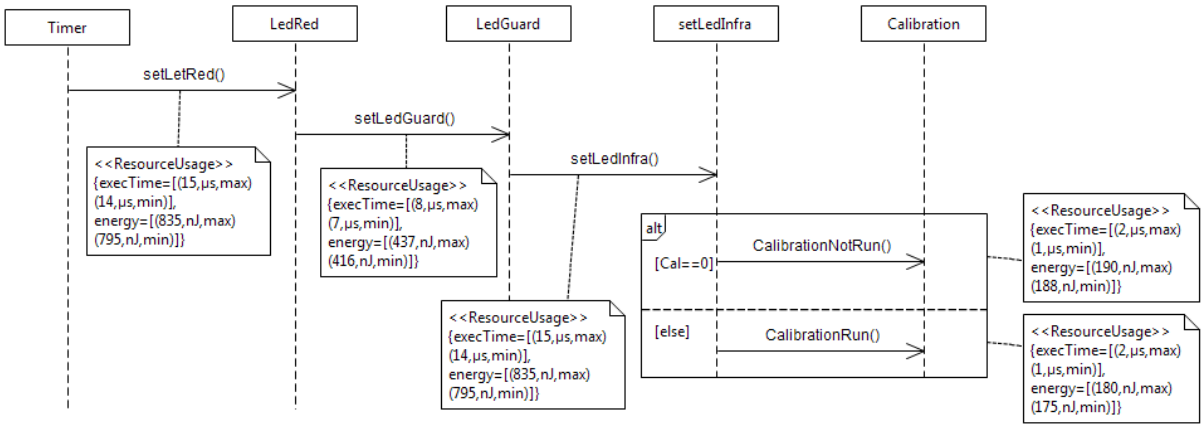

Figura 4.5: Modelagem em DS de um pulsoxímetro [4].

A figura 4.6 representa a RdP-R gerada a partir da aplicação da transformação de modelos no DS referente ao pulsoxímetro.

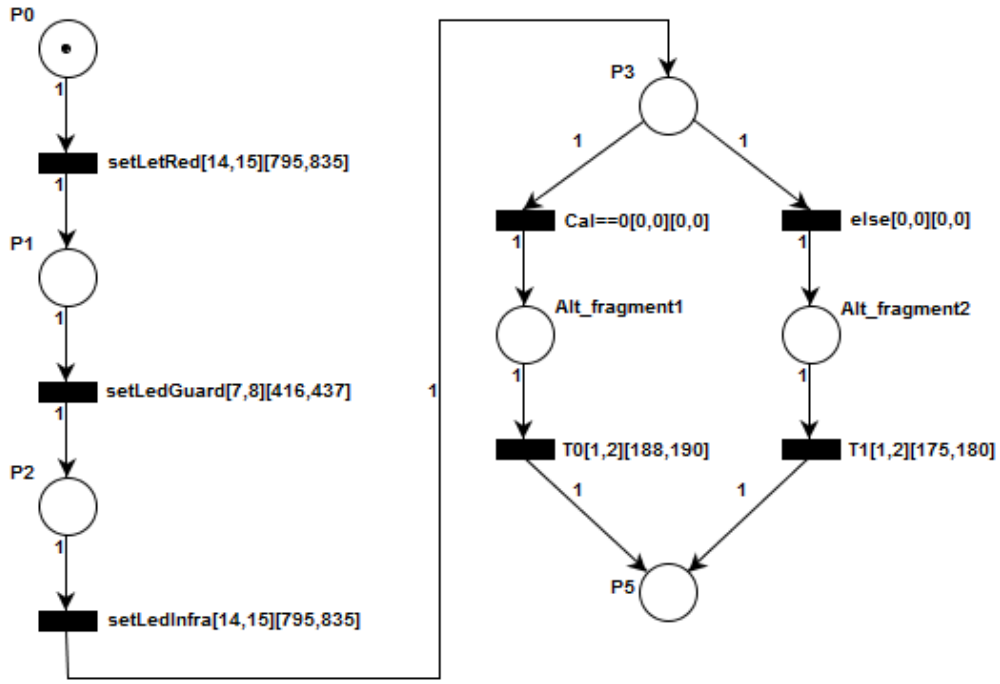

Figura 4.6: Modelagem em RdP-T de um pulsoxímetro.

A seguir é apresentados os códigos *.net* da RdP-T ilustrado pela figura 4.6. O primeiro leva em consideração apenas às restrições de tempo presentes na RdP-T.

```
1  net pulsoximetro_tempo
2  tr setLetRed [14,15] P0 -> P1
3  tr setLedGuard [7,8] P1 -> P2
4  tr setLedInfra [14,15] P2 -> P3
5  tr CalIgual0 [0,0] P3 -> Alt_fragment1
6  tr T0 [1,2] Alt_fragment1 -> P5
7  tr else [0,0] P3 -> Alt_fragment2
8  tr T1 [1,2] Alt_fragment2 -> P5
9
10 pl P0 (1)
11 pl P1 (0)
12 pl P2 (0)
13 pl P3 (0)
14 pl Alt_fragment1 (0)
15 pl Alt_fragment2 (0)
16 pl P5 (0)
```

O código a seguir representa a RdP-T levando em consideração apenas as restrições de energia.

```
net pulsoximetro_energia
tr setLetRed [795,835] P0 -> P1
tr setLedGuard [416,437] P1 -> P2
tr setLedInfra [795,835] P2 -> P3
tr CalIgual0 [0,0] P3 -> Alt_fragment1
tr T0 [188,190] Alt_fragment1 -> P5
tr else [0,0] P3 -> Alt_fragment2
tr T1 [175,180] Alt_fragment2 -> P5

pl P0 (1)
pl P1 (0)
pl P2 (0)
pl P3 (0)
pl Alt_fragment1 (0)
pl Alt_fragment2 (0)
pl P5 (0)
```

A análise intervalar realizada representando tempo é ilustrada na figura 4.7. O ponto inicial do grafo de classes é dado pelo nó *C0_0* que possui um tempo relativo de $[R = 14, 15]$ e dispara para a transição *setLedRed*. O próximo nó é representado por *C1_1* que possui um tempo relativo $[R = 7, 8]$ e um tempo global $[G = 14, 15]$. O disparo ainda segue as transições *setLedGuard* e *setLetInfra* antes do nó *C3_3*. Este possui um tempo global $[G = 35, 38]$. A partir desse nó é possível disparar para a transição *CalIgual0* ou *else*. Disparando para a transição *CalIgual0*, esta é seguida do disparo para a transição *TO*, finalizando na classe *C5_6* com tempo global $G = [36, 40]$. Em contrapartida, seguindo a sequência de disparos pela transição *else*, esta é acompanhada do disparo para a transição *T1* e tem seu tempo global $G = [36, 40]$ representado pela classe *C5_7*.

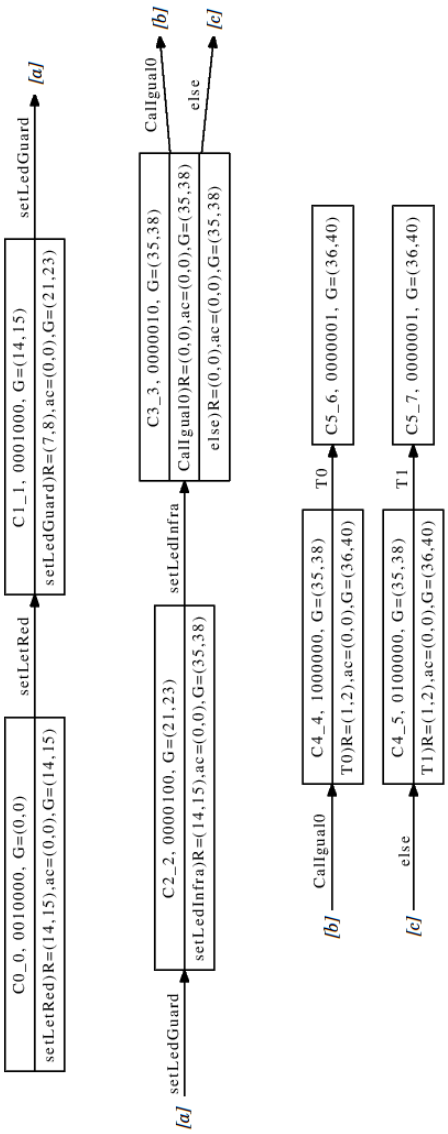

Figura 4.7: Grafo de classes de 5 níveis, da RdP-T da figura 4.6 representando tempo.

A figura 4.8 ilustra a análise intervalar realizada na RdP-T da figura 4.6. Esta análise leva em consideração as restrições de energia da RdP-T. O nó *C0_0* é o ponto inicial do grafo e o mesmo dispara para a transição *setLedRed* seguida de *setLedGuard* e *setLedInfra* antes de chegar ao nó *C3_3*. Este possui um tempo global $G = [2006, 2107]$ e pode disparar para a transição *CalIgual0* ou *else*. Disparando para a transição *CalIgual0*, esta é seguida do disparo para a transição *TO*, finalizando na classe *C5_6* com tempo global $G = [2194, 2297]$. Em contrapartida, seguindo a sequência de disparos pela transição *else*, esta é acompanhada do disparo para a transição *T1* e tem seu tempo global $G = [2181, 2287]$ representado pela classe *C5_7*.

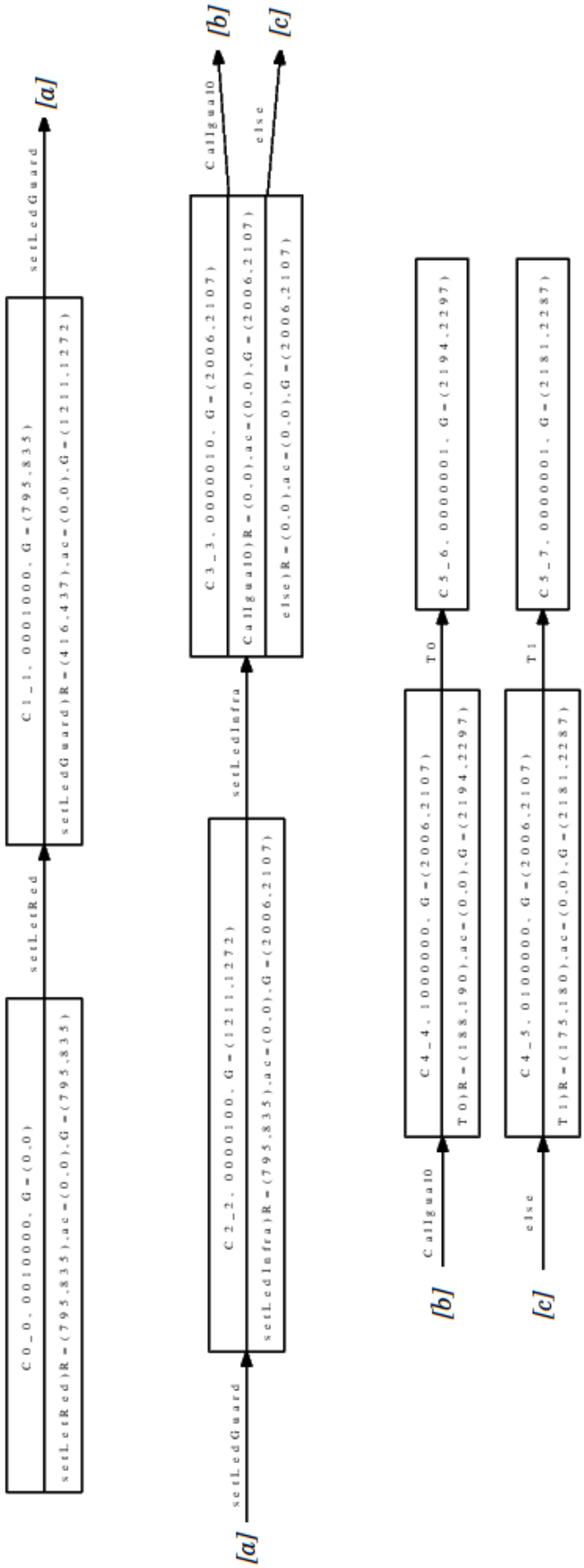

Figura 4.8: Grafo de classes de 5 níveis, da RdP-T da figura 4.6 representando energia.

4.3 Estudo de Caso 3 – *Framework* para Simular o Comportamento de Sistemas Embarcados de Tempo Real

Neste estudo de caso, a transformação de modelos é aplicada à modelagem de um *framework* que simula o comportamento especificado em modelos UML de sistemas embarcados de tempo real [55]. Esta modelagem está representada na figura 4.9.

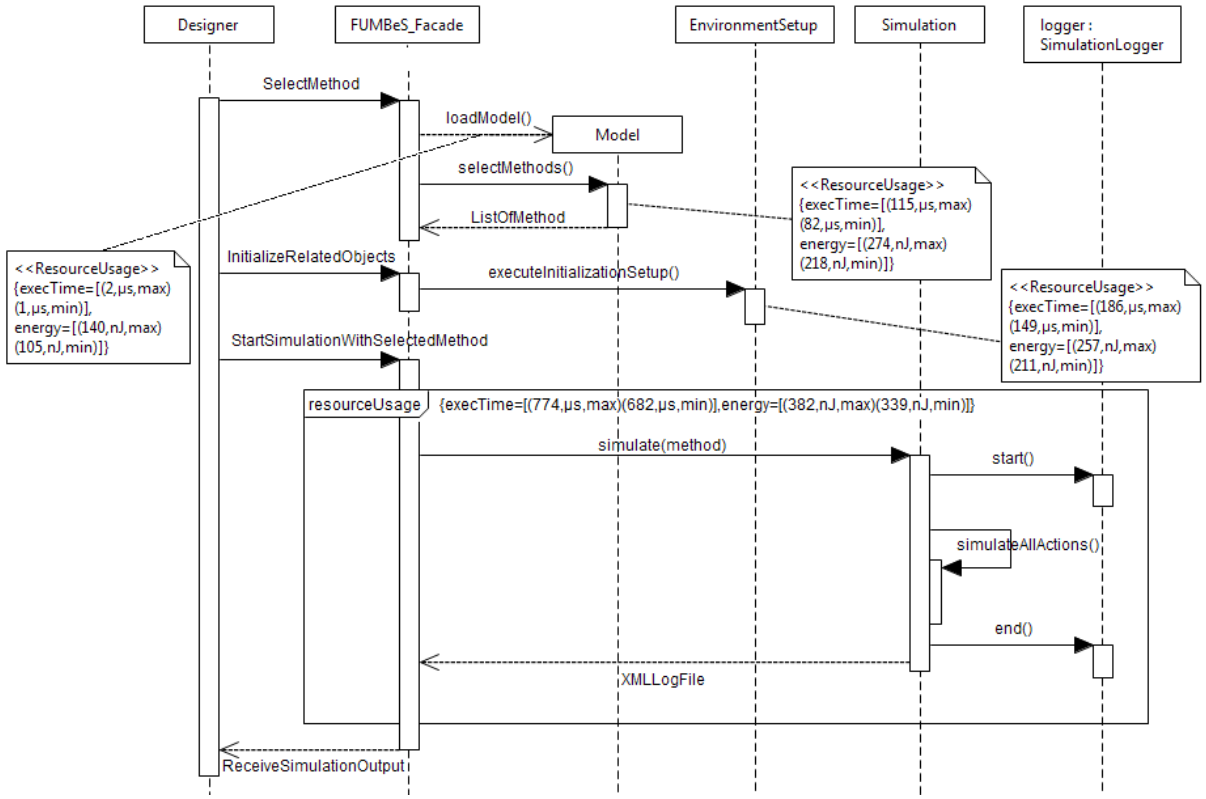

Figura 4.9: Modelagem em DS de um *framework* para simular o comportamento de sistemas embarcados de tempo real. Adaptado de Wehrmeister [55].

Primeiramente o projetista, na figura *"Designer"*, seleciona um ou mais métodos, representado pela mensagem *"selectMethods"*. Antes de iniciar a simulação, é importante configurar o estado inicial dos objetos do sistema relacionados com o método a ser simulado, ilustrado pelas mensagens *"InitializeRelatedObjects"* e *"executeInitializationSetup"*. Após a inicialização, o comportamento do método selecionado pode ser simulado. Esta simulação é representada pelas mensagens *"StartSimulationWithSelectedMethod"*, *"simulate(method)"*, *"start"*, *"simulateAllActions()"* e *"end"* . Ao fim da simulação o projetista recebe os resultados da simulação em forma de um arquivo XML contendo o traçado de

todas as ações executadas. Esta ultima etapa é representada no diagrama pelas mensagens *"XMLLogFile"* e *"ReceiveSimulationOutput"*.

Como resultado da transformação de modelos, tem-se a RdP-T respectiva, ilustrada na figura 4.10.

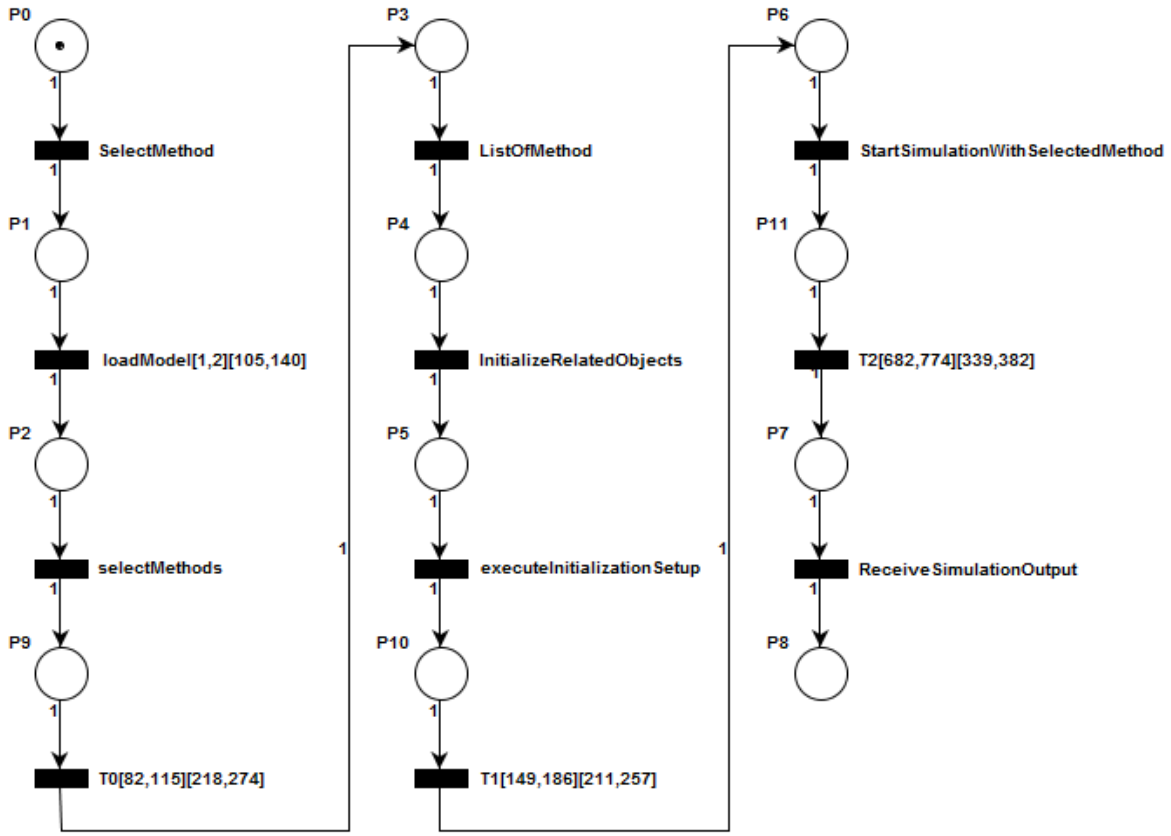

Figura 4.10: Modelagem em RdP-T de um *framework* para simular o comportamento de sistemas embarcados de tempo real.

A seguir é apresentado o código *.net* da RdP-T ilustrado na figura 4.10. O primeiro leva em consideração apenas às restrições de tempo presentes na RdP-T.

```
1 net framework_validacao_tempo
2 tr SelectMethod [100,200] P0 -> P1
3 tr loadModel [1,2] P1 -> P2
4 tr selectMethods [100,200] P2 -> P9
5 tr ListOfMethod [100,200] P3 -> P4
6 tr InitializeRelatedObjects [100,200] P4 -> P5
7 tr executeInitializationSetup [100,200] P5 -> P10
8 tr StartSimulationWithSelectedMethod [100,200] P6 -> P11
```

```
tr ReceiveSimulationOutput [100,200] P7 -> P8
tr T0 [82,115] P9 -> P3
tr T1 [149,186] P10 -> P6
tr T2 [682,774] P11 -> P7

pl P0 (1)
pl P1 (0)
pl P2 (0)
pl P3 (0)
pl P4 (0)
pl P5 (0)
pl P6 (0)
pl P7 (0)
pl P9 (0)
pl P10 (0)
pl P11 (0)
pl P8 (0)
```

O código a seguir representa a RdP-T levando em consideração apenas as restrições de energia.

```
net framework_validacao_energia
tr SelectMethod [100,200] P0 -> P1
tr loadModel [105,140] P1 -> P2
tr selectMethods [100,200] P2 -> P9
tr ListOfMethod [100,200] P3 -> P4
tr InitializeRelatedObjects [100,200] P4 -> P5
tr executeInitializationSetup [100,200] P5 -> P10
tr StartSimulationWithSelectedMethod [100,200] P6 -> P11
tr ReceiveSimulationOutput [100,200] P7 -> P8
tr T0 [218,274] P9 -> P3
tr T1 [211,257] P10 -> P6
tr T2 [339,382] P11 -> P7

pl P0 (1)
pl P1 (0)
```

```
16 pl P2 (0)
17 pl P3 (0)
18 pl P4 (0)
19 pl P5 (0)
20 pl P6 (0)
21 pl P7 (0)
22 pl P9 (0)
23 pl P10 (0)
24 pl P11 (0)
25 pl P8 (0)
```

O resultado da análise intervalar utilizando a ferramenta GTT [40, 41] é ilustrada na figura 4.11. Esta análise considera apenas o intervalo temporal. O ponto inicial do grafo de classes é dado pelo nó *C0_0*. Este nó possui um tempo relativo de $R = [100, 200]$. Este tempo é atribuído como valor padrão pelo software *"XMI to TINA Converter"* quando uma transição da RdP-T não possui explicitamente uma restrição.

A marcação dispara para a transição *SelectMethod*, seguindo a sequência de disparos, o nó *C1_1* possui um tempo relativo de $R = [1, 2]$. Após toda a sequência de disparos, a classe *C5_5* possui um tempo global $G = [383, 717]$.

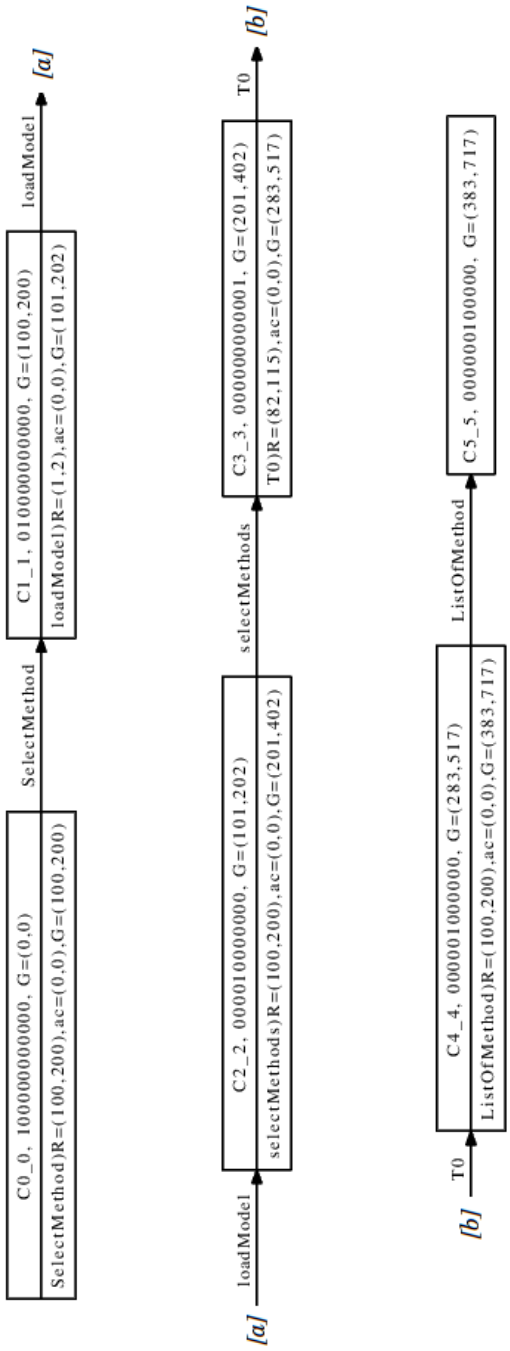

Figura 4.11: Grafo de classes de 5 níveis, da RdP-T da figura 4.10 representando tempo.

A figura 4.12 ilustra a análise intervalar realizada na RdP-T da figura 4.10. Esta análise leva em consideração as restrições de energia da RdP-T. O nó *C0_0* é o ponto inicial do grafo e possui um tempo relativo de $R = [100, 200]$. Seguindo a sequência de disparos, tem-se *loadModel, selectMethods, T0* e *ListOfMethod*, finalizando na classe *C5_5*, com tempo global $G = [623, 1014]$.

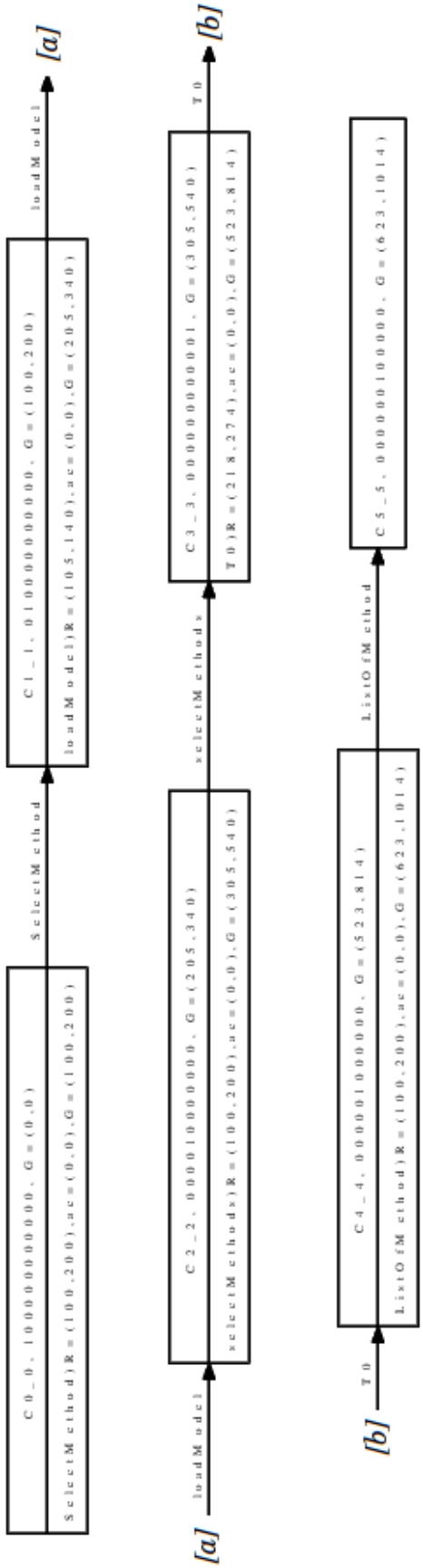

Figura 4.12: Grafo de classes de 5 níveis, da RdP-T da figura 4.10 representando energia.

4.4 Experimentos

Além dos modelos apresentados nas seções 4.1, 4.2 e 4.3, outros três modelos de DS foram transformados e analisados de acordo com a metodologia apresentada no início deste capítulo. Alguns dados deste experimento encontram-se na tabela 4.1.

Os experimentos foram conduzidos em um ambiente Intel Core 2 Duo 2.00 GHz, 3 GB de memória RAM e em um sistema operacional Ubuntu 11.04.

O primeiro modelo experimentado se refere a processos básicos de um caixa eletrônico [56]. Este possui um total de 14 mensagens, 3 *execution specification* com restrições e 1 *combined fragment*. O tempo de execução para efetuar a transformação deste modelo foi de 0,011s. O segundo experimento ilustra as etapas de um *check-in* por um agente de companhia aérea [43]. O terceiro modelo é um *framework* responsável por simular o comportamento de sistemas embarcados de tempo real e foi apresentado por Wehrmeister et al. [55]. Um sistema de pulsoxímetro [4] é o quarto modelo. O quinto exemplo é um modelo alusivo a um processo de quebra de segurança por um invasor [3, 22]. O modelo de Ribeiro [43] é referente a um sistema de elevador. Os resultados dos experimentos com seus modelos são apresentados a seguir na tabela 4.1.

Tabela 4.1: Comparação das transformações

Modelo	Mensagem	Execution Specification	Combined Fragment	Lugares	Transições
caixa eletrônico	14	3	1	14	13
check-in	10	0	1	12	10
framework	13	2	1	12	11
pulsoxímetro	5	0	1	7	7
quebra de segurança	12	2	1	14	14
sistema de elevador	8	0	2	8	9

4.5 Considerações do Capítulo

Este capítulo apresentou em detalhes três estudos de caso utilizados para validar a transformação de modelos implementada neste trabalho. Também foram apresentados os resultados de experimentos realizados em seis modelos de diagrama de sequência UML.

Os estudos de caso apresentam primeiramente o diagrama de sequência a ser transformado. Após realizar a transformação, a rede de Petri gerada é ilustrada. A conversão da rede de Petri inicialmente em formato *.xmi* para o formato *.net* é realizada e o código de saída *.net* é apresentado. Com este código é realizada uma análise intervalar na ferramenta GTT [40, 41] e os grafos de classes são apresentados.

CAPÍTULO 5

CONCLUSÕES

Este trabalho apresentou uma transformação de modelos entre diagrama de sequência UML [35] contendo restrições de tempo e energia para rede de Petri temporal [6]. Adaptações foram realizadas nos metamodelos de diagrama de sequência e rede de Petri temporal para que ambos permitissem a representação de tempo e energia em seus modelos. A representação das restrições no diagrama de sequência somente foi possível através do estudo e aplicação do perfil MARTE [37].

Para a realização da implementação da transformação de modelos, foram definidas as transformações de elementos específicos de diagrama de sequência para rede de Petri temporal.

Foi implementada uma ferramenta para a transformação automática de modelos e um programa para a conversão de formatos de *.xmi* para *.net* com o qual é possível realizar verificações formais em ferramentas como TINA [28] e GTT [40, 41].

Por fim, foram pesquisados na literatura modelagens de software embarcados com a finalidade de aplicar o trabalho realizado no contexto de verificação temporal e de energia.

5.1 Trabalhos Futuros

A seguir são apresentadas algumas sugestões para trabalhos futuros:

- Integrar as diversas redes de Petri geradas pelos diagramas de sequência para representar o sistema completo.
- Implementar na transformação outros elementos de diagrama de sequência, como por exemplo: *Critical, StateInvariant, DestructionOccurenceSpecification, Ignore,* entre outros.
- Realizar a conversão de modelos por outras técnicas de transformação, como por

exemplo *Query / View / Transformation (QVT)* [36] para que seja possível uma comparação de desempenho e nível de abstração das diferentes linguagens.

- Estender as transformações com restrições de tempo e energia para outros diagramas UML, como por exemplo: Diagrama de Tempo, Diagrama de Atividades, Diagrama de Estados, entre outros.

BIBLIOGRAFIA

[1] S. W. Ambler. Agile model driven development is good enough. *IEEE Softw.*, 20(5):71–73, setembro de 2003.

[2] Mohamed Ariff Ameedeen, Behzad Bordbar, e Rachid Anane. Model interoperability via model driven development. *J. Comput. Syst. Sci.*, 77(2):332–347, 2011.

[3] Kyriakos Anastasakis, Behzad Bordbar, Geri Georg, e Indrakshi Ray. Uml2alloy: A challenging model transformation. *In: ACM/IEEE 10th International Conference on Model Driven Engineering Languages and Systems (MoDELS*, páginas 436–450. Springer, 2007.

[4] Ermeson Andrade, Paulo Maciel, Gustavo Callou, Bruno Nogueira, e Carlos Araújo. Mapping uml sequence diagram to time petri net for requirement validation of embedded real-time systems with energy constraints. *Proceedings of the 2009 ACM symposium on Applied Computing*, SAC '09, páginas 377–381, New York, NY, USA, 2009. ACM.

[5] LINA & INRIA ATLAS group. Atl: Atlas transformation language, 2005.

[6] Bernard Berthomieu e Michel Diaz. Modeling and verification of time dependent systems using time petri nets. *IEEE Trans. Softw. Eng.*, 17(3):259–273, março de 1991.

[7] Jean Bézivin e Olivier Gerbé. Towards a precise definition of the omg/mda framework. *Proceedings of the 16th IEEE international conference on Automated software engineering*, ASE '01, páginas 273–, Washington, DC, USA, 2001. IEEE Computer Society.

[8] V Bhanot, D Paniscotti, A Roman, e B Trask. Using domain-specific modeling to develop software defined radio components and applications. *5th OOPSLA Workshop on Domain-Specific Modeling (DSMâ05), San Diego, California, USA*, 2005.

[9] G. Booch, J. Rumbaugh, e I. Jacobson. *UML: guia do usuário*. Elsevier, 2006.

[10] Erwan Breton e Jean Bézivin. Towards an understanding of model executability. *Proceedings of the international conference on Formal Ontology in Information Systems - Volume 2001*, FOIS '01, páginas 70–80, New York, NY, USA, 2001. ACM.

[11] Erwan Breton e Jean Bézivin. Towards an understanding of model executability. *Proceedings of the international conference on Formal Ontology in Information Systems - Volume 2001*, FOIS '01, páginas 70–80, New York, NY, USA, 2001. ACM.

[12] F. Budinsky, D. Steinberg, E. Merks, R. Ellersick, e T. Grose. *Eclipse Modeling Framework*. Addison Wesley Professional, 2003.

[13] Janette Cardoso e Robert Valette. *Redes de Petri*. UFSC, 1997.

[14] Benoît Combemale, Xavier Crégut, Pierre-Loïc Garoche, e Xavier Thirioux. Essay on semantics definition in mde - an instrumented approach for model verification. *JSW*, 4(9):943–958, 2009.

[15] Benoit Combemale, Xavier Crégut, Pierre-Loic Garoche, Xavier Thirioux, e Francois Vernadat. A Property-Driven Approach to Formal Verification of Process Models. Jorge Cardoso, José Cordeiro, Joaquim Filipe, e Vitor Pedrosa, editors, *Enterprise Information System IX*, volume 12, páginas 286–300. LNBIP, Springer, 2008.

[16] Pedro Dias. Atlas transformation language: Transformação de modelos java em modelos de classes uml. Relatório técnico, Out de 2009. 75 p. Relatório – DEI-ISEP.

[17] Zhanwei Du, Yongjian Yang, Jiaqi Xu, e Jie Wang. Mapping uml models to colored petri nets models based on edged graph grammar. *Journal of Computational Information Systems*, 11(7):3838–3845, novembro de 2011.

[18] Florida Estrella, Zsolt Kovacs, Jean-Marie Le Goff, Richard McClatchey, e Norbert Toth. Meta-data objects as the basis for system evolution. *Proceedings of the Second International Conference on Advances in Web-Age Information Management*, WAIM '01, páginas 390–399, London, UK, UK, 2001. Springer-Verlag.

[19] M. Fowler e K. Scott. *UML essencial: um breve guia para a linguagem-padrão de modelagem de objetos*. Bookman, 2000.

[20] J. Ganssle. *The Art of Designing Embedded Systems*. Elsevier Science, 1999.

[21] Vahid Garousi, Lionel Briand, e Yvan Labiche. Control flow analysis of uml 2.0 sequence diagrams. Relatório técnico, Proc. of European Conference on Model Driven Architecture-Foundations and Applications (ECMDA-FA), 2005.

[22] Geri Georg, Indrakshi Ray, Kyriakos Anastasakis, Behzad Bordbar, Manachai Toahchoodee, e Siv Hilde Houmb. An aspect-oriented methodology for designing secure applications. *Inf. Softw. Technol.*, 51(5):846–864, maio de 2009.

[23] James Gosling, Bill Joy, e Guy L. Steele. *The Java Language Specification*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1st edition, 1996.

[24] Elhillali Kerkouche, Algeria Allaoua Chaoui, El Bay Bourennane, e Ouassila Labbani. A uml and colored petri nets integrated modeling and analysis approach using graph transformation. *Journal of Object Technology*, 9(4):25–43, julho de 2010.

[25] Ekkart Kindler e Robert Wagner. Triple graph grammars: Concepts, extensions, implementations, and application scenarios. *University of Paderborn*, 2007.

[26] A.G. Kleppe, J.B. Warmer, e W. Bast. *Mda Explained, the Model Driven Architecture: Practice and Promise*. The Addison-Wesley Object Technology Series. Addison-Wesley, 2003.

[27] Andres Knöpfel, Bernhard Gröne, e Peter Tabeling. *Fundamental modeling concepts*. Wiley, West Sussex UK, 2005.

[28] Laas. Tina - time petri net analyzer, 2010.

[29] C. Larman. *Utilizando UML e padrões: uma introdução à analise e ao projeto orientados a objetos*. Bookman, 2002.

[30] P. Maciel, R. Lins, e P. Cunha. *Introdução às Redes de Petri e Aplicações*. Sociedade Brasileira de Computação, 1996.

[31] José Mauro. Programação imperativa versus programação declarativa, Janeiro de 2009. Acesso em: 17 de fev. 2012.

[32] Marjan Mernik, Jan Heering, e Anthony M. Sloane. When and how to develop domain-specific languages. *ACM Comput. Surv.*, 37(4):316–344, dezembro de 2005.

[33] T. Murata. Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*, 77(4):541–580, April de 1989.

[34] Tammy Noergaard. *Embedded Systems Architecture: A Comprehensive Guide for Engineers and Programmers*. Newnes, 2005.

[35] Object Management Group OMG. Omg unified modeling languagetm (omg uml), superstructure, Maio de 2010. Acesso em: 19 mai. 2012.

[36] Object Management Group OMG. Meta object facility (mof) 2.0 query/view/transformation specification, Janeiro de 2011. Acesso em: 01 mar. 2012.

[37] Object Management Group OMG. Meta object facility (mof) 2.0 uml profile for marte, Junho de 2011. Acesso em: 02 abr. 2012.

[38] Object Management Group OMG. Omg meta object facility (mof) core specification, Agosto de 2011. Acesso em: 30 abr. 2012.

[39] Object Management Group OMG. Omg unified modeling language (omg uml), infrastructure, Agosto de 2011. Acesso em: 21 mar. 2013.

[40] Letícia Mara Peres. *Proposta de um Método de Verificação por Tempo Global com Redes de Petri no Desenvolvimento de Software Embarcado e em Tempo Real*. Tese de Doutorado, Universidade Federal do Paraná, 2010.

[41] Leticia Mara Peres, Luis Allan Künzle, e Eduardo Todt. Applying global time petri net analysis on the embedded software context. *Sba: Controle & Automação Sociedade Brasileira de Automatica*, 22(6):610–619, 2011.

[42] R.S. Pressman. *Engenharia de software*. McGraw-Hill, 2006.

[43] Óscar Rafael da Silva Ferreira Ribeiro. *Animation-based validation of reactive software systems using behavioural models*. Tese de Doutorado, Escola de Engenharia, Universidade do Minho, Minho, Portugal, 2009.

[44] Hui Shen, Aliya Virani, e Jianwei Niu. Formalize uml 2 sequence diagrams. *High-Assurance Systems Engineering, IEEE International Symposium on*, 0:437–440, 2008.

[45] Pathfinder Solutions. Platform-independent model-driven development, novembro de 2012.

[46] Ian Sommerville. *Software engineering (5th ed.)*. Addison Wesley Longman Publishing Co., Inc., Redwood City, CA, USA, 1995.

[47] P. C. Stadzisz e D. P. B Renaux. *Software Embarcado*, páginas 107–155. Unicentro (Universidade Estadual do Centro-Oeste), SBC. (Org.). XIV Escola Regional de Informática SBC Paraná, Guarapuava, Paraná, 2007.

[48] Anthony Spiteri Staines. A triple graph grammar mapping of uml 2 activities into petri nets. *International Journal Of Computers*, 4(1):27, nov de 2009.

[49] Tony Spiteri Staines. Intuitive mapping of uml 2 activity diagrams into fundamental modeling concept petri net diagrams and colored petri nets. *Proceedings of the 15th Annual IEEE International Conference and Workshop on the Engineering of Computer Based Systems*, ECBS '08, páginas 191–200, Washington, DC, USA, 2008. IEEE Computer Society.

[50] John A. Stankovic. Misconceptions about real-time computing: A serious problem for next-generation systems. *Computer*, 21(10):10–19, outubro de 1988.

[51] Dave Steinberg, Frank Budinsky, Ed Merks, e Marcelo Paternostro. *EMF: eclipse modeling framework*. Addison-Wesley Professional, 2008.

[52] Lambert M. Surhone, Mariam T. Tennoe, e Susan F. Henssonow. *Transformation Language*. Betascript Publishing, 2010.

[53] Mathupayas Thongmak e Pornsiri Muenchaisri. Design of rules for transforming uml sequence diagrams into java code. *Asia-Pacific Software Engineering Conference*, 0:485, 2002.

[54] Arie van Deursen e Paul Klint. Little languages: little maintenance. *Journal of Software Maintenance*, 10(2):75–92, março de 1998.

[55] Marco A Wehrmeister, João G Packer, e Luis M Ceron. Framework to simulate the behavior of embedded real-time systems specified in uml models. *Computing System Engineering (SBESC), 2011 Brazilian Symposium on*, páginas 1–7. IEEE, 2011.

[56] Jon Whittle e Johann Schumann. Generating statechart designs from scenarios. *Proceedings of the 22nd international conference on Software engineering*, ICSE '00, páginas 314–323, New York, NY, USA, 2000. ACM.

[57] Xiao-Qin Zeng, Xiu-Qing Han, e Yang Zou. An edge-based context-sensitive graph grammar formalism. *Journal of Software*, 19(8):1893–1901, 2008.

APÊNDICE A

CONVERSOR XMI PARA TINA - MANUAL DE UTILIZAÇÃO

Ao executar o sistema, o usuário depara-se com a tela principal do software, ilustrado na figura A.1. nesta tela o usuário deve indicar o arquivo de entrada, escolher qual restrição o sistemas deve considerar e indicar o diretório onde será salvo o arquivo gerado.

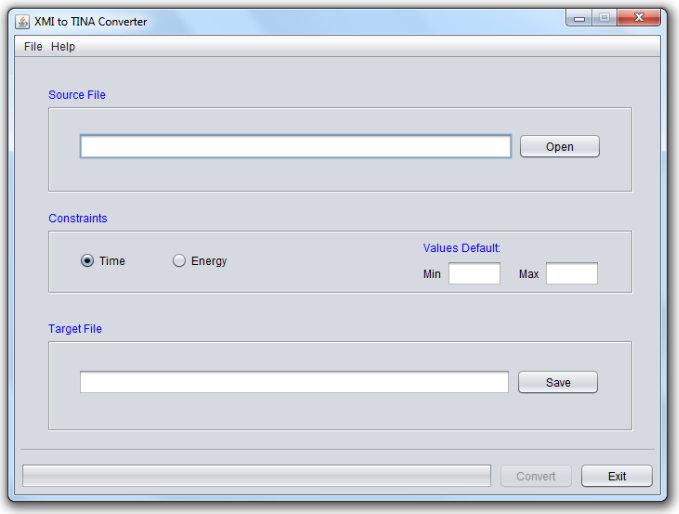

Figura A.1: XMI to TINA Converter.

O primeiro campo, chamado de *Source File*, é responsável por conter o diretório do arquivo XMI que será convertido em um arquivo no formato *".net"*. Este campo é preenchido automaticamente posterior ao usuário indicar o qual arquivo pretende efetuar a transformação através da navegação realizada após pressionar o botão *"Open"*, situado ao seu lado. A figura A.2 ilustra a navegação entre os diretórios utilizando o botão *"Open"*.

Após a definição do arquivo .xmi de entrada, o usuário deve definir qual restrição (tempo ou energia) deseja em sua rede de Petri, pois o analisador estático desenvolvido por Peres [40], apenas uma restrição associada é aceita. Esta escolha é dada pelas opções *"Time"* e *"Energy"*. Para os elementos da rede de Petri temporal que não possuem restrições explícitas, valores mínimo e máximo serão definidos como padrão pelo próprio

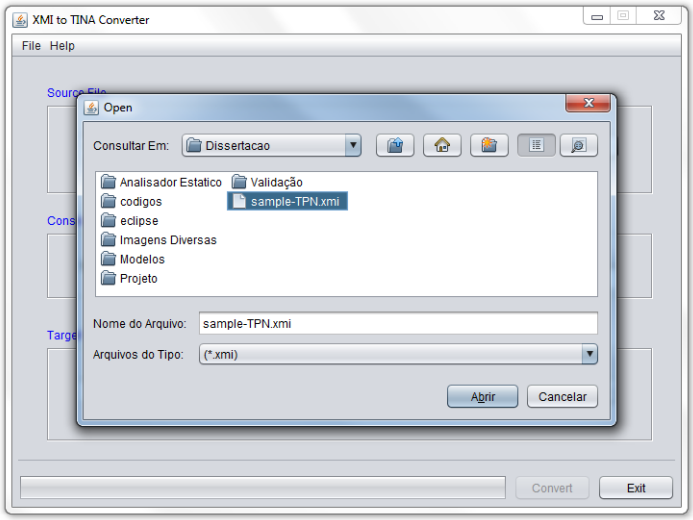

Figura A.2: Arquivo de entrada.

usuário nos campos *"Min"* e *"Max"* respectivamente.

O último campo a ser preenchido *(Target File)*, deve-se indicar o diretório o qual será salvo e o nome do novo arquivo *.net* (figura A.3), este preenchimento é realizado pressionando o botão *"Save"*.

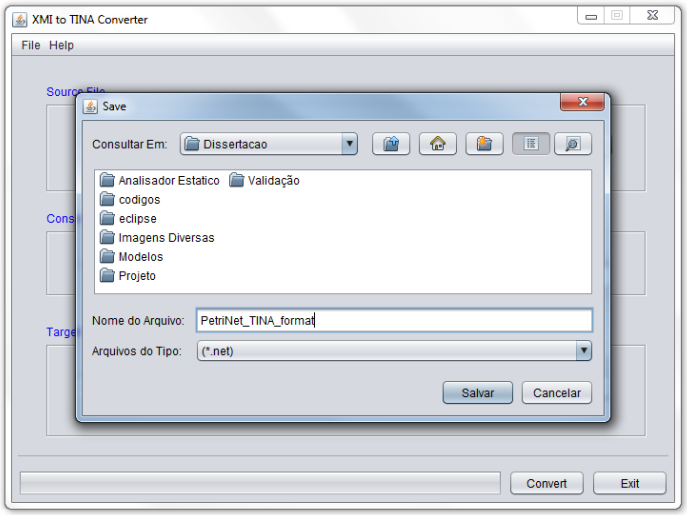

Figura A.3: Definição do arquivo de saída.

Definido o arquivo de entrada, restrições, valores padrões para restrições não explícitas, nome e o diretório do arquivo de saída, a transformação pode ser efetuar pressionando o botão *"Convert"*. Feito isto, uma mensagem de sucesso ou de erro da conversão é exibida (figura A.4).

Para finalizar, a Tela *"About"* apresenta os créditos do desenvolvimento do software e

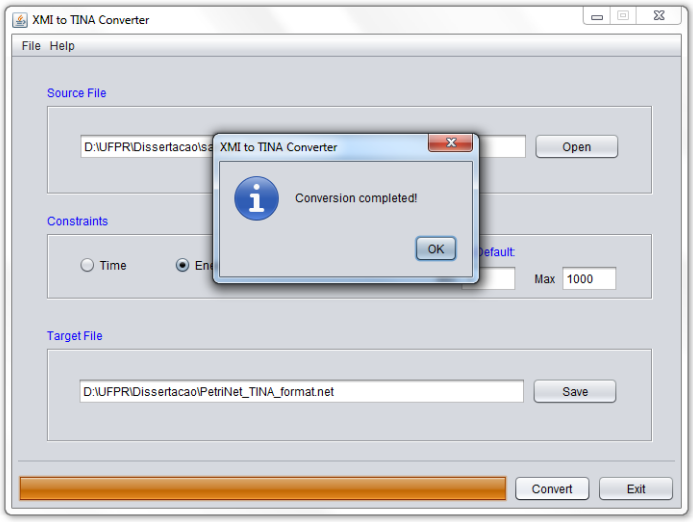

Figura A.4: Confirmação da conversão.

está ilustrada na figura A.5.

Figura A.5: Tela *About*.

.

VINÍCIUS CAMARGO ANDRADE

TRANSFORMAÇÃO DE MODELOS DE DIAGRAMA DE SEQUÊNCIA UML CONTEMPLANDO RESTRIÇÕES DE TEMPO E ENERGIA PARA REDE DE PETRI TEMPORAL

Dissertação apresentada como requisito parcial à obtenção do grau de Mestre. Programa de Pós-Graduação em Informática, Setor de Ciências Exatas, Universidade Federal do Paraná.
Orientadora: Profª. Drª. Letícia Mara Peres

CURITIBA

2013