PONTIFÍCIA UNIVERSIDADE CATÓLICA DE MINAS GERAIS

Programa de Pós-Graduação em Informática

Amanda Danielle Lima de Oliveira Tameirão

VERIFICAÇÃO SIMBÓLICA DE MODELOS APLICADA A DIAGRAMAS DE TEMPORIZAÇÃO

Belo Horizonte

2015

Amanda Danielle Lima de Oliveira Tameirão

VERIFICAÇÃO SIMBÓLICA DE MODELOS APLICADA A DIAGRAMAS DE TEMPORIZAÇÃO

Dissertação apresentada ao Programa de Pós-Graduação em Informática da Pontifícia Universidade Católica de Minas Gerais, como requisito parcial para obtenção do título de Mestre em Informática.

Orientador: Prof. Dr. Mark Alan Junho Song

Belo Horizonte

2015

FICHA CATALOGRÁFICA
Elaborada pela Biblioteca da Pontifícia Universidade Católica de Minas Gerais

Tameirão, Amanda Danielle Lima de Oliveira
T157v Verificação simbólica de modelos aplicada a diagramas de temporização / Amanda Danielle Lima de Oliveira Tameirão. Belo Horizonte, 2015.
94 f. : il.

Orientador: Mark Alan Junho Song
Dissertação (Mestrado) - Pontifícia Universidade Católica de Minas Gerais. Programa de Pós-Graduação em Informática.

1. Engenharia de software. 2. Programas de computador - Verificação. 3. UML (Computação). 4. Software de sistemas. I. Song, Mark Alan Junho. II. Pontifícia Universidade Católica de Minas Gerais. Programa de Pós-Graduação em Informática. III. Título.

CDU: 681.3.031

Amanda Danielle Lima de Oliveira Tameirão

VERIFICAÇÃO SIMBÓLICA DE MODELOS APLICADA A DIAGRAMAS DE TEMPORIZAÇÃO

Dissertação apresentada ao Programa de Pós-Graduação em Informática como requisito parcial para qualificação ao Grau de Mestre em Informática pela Pontifícia Universidade Católica de Minas Gerais.

Prof. Dr. Mark Alan Junho Song (Orientador) - PUC Minas

Prof. Dr. Autran Macedo – Universidade Federal de Uberlândia

Prof. Dr. Henrique Cota de Freitas - PUC Minas

Belo Horizonte, 22 de abril de 2015.

À minha família, Bruno, Bruna e Vitor, que, com seu amor incondicional, me motivam diariamente. Amo muito vocês.

AGRADECIMENTOS

Tempo de agradecer...

A Deus, que em sua grandeza e magnitude me abençoou e esteve comigo durante esta caminhada.

A minha amada família, sem os quais eu não poderia alcançar essa vitória e a quem eu a dedico. Meu amado esposo Bruno, que com sua paciência me incentivou e sempre me apoiou. Companheiro amoroso e especial que faz parte da minha vida e demonstra diariamente seu amor. Minha filha Bruna que, mesmo com minha ausência, demonstra a todo momento seu amor e admiração. Meu filho Vitor que, ainda em gestação, permanece tranquilo, mesmo quando preciso de muita dedicação. Tenham certeza do meu amor por vocês.

Aos meus pais, que tanto me amam e acreditam em mim. Minha mãe, com sua agitação "tranquila", sempre disposta a ajudar, compartilhar e nos alegrar. Uma mulher guerreira e carinhosa, e por muitas vezes engraçada, capaz de tornar nossos dias sempre especiais e abençoados. Meu pai, uma das pessoas que mais me incentivou a estudar e a vencer. Homem de valores indescritíveis, exemplo de força, amor e humildade, que sempre se orgulhou das minhas conquistas e que certamente estaria festejando esta. A vocês meu eterno amor.

Aos meus irmãos, Karla, Bruno, Paloma e Gabrielle, que sempre me incentivam e me acompanham. Assim como meus sogros, Teresa e Armando, e cunhados, Elaine, Felipe, Denise, Marcelo, Clarissa e Amanda, que completam e alegram minha família.

Aos meus queridos amigos, anjos que fazem parte da minha vida e que a tornam melhor. Agradecimentos especiais aos amigos que fiz durante o mestrado, que fizeram esta caminhada ser mais feliz.

Durante este tempo li e reli diversos agradecimentos a equipe de apoio do mestrado da PUC, confesso que no início achei que fazia parte de um "clichê", pois eu estava completamente enganada... uma enorme gratidão a Giovana e sua equipe, que são incapazes de nos abandonar, sempre nos apoiando e por diversas vezes acreditando em nós mais do que nós acreditamos.

Ao meu orientador Mark Alan, que me concedeu seus conhecimentos e me incentivou. À Capes pelo precioso incentivo, com o qual garanti essa vitória. Aos professores da PUC, que, sempre prestativos, nos transmitem seus conhecimentos.

A Universidade FUMEC, por seu apoio, confiança e incentivo constante.

E a você, que muito carinhosamente lê esses agradecimentos e considera meu trabalho.

Tempo de comemorar...

“Digno és, Senhor, de receber glória e honra, e poder; porque tu criaste todas as coisas, e por tua vontade são e foram criadas.”

Apocalipse, capítulo 4, versículo 11

RESUMO

Durante todo o ciclo de vida de um produto de *software* erros e falhas são introduzidos, e o ideal é que estes sejam corrigidos o quanto antes.

Diversos sistemas utilizam o diagrama de temporização da UML como uma especificação para o comportamento do sistema. Garantir a correta especificação antecipadamente possibilita uma modelagem livre de erros, evitando propagação para as demais fases em que o custo de correção se torna consideravelmente elevado.

Este trabalho, utiliza a verificação simbólica de modelos para avaliar a especificação fornecida através de um diagrama de temporização, este é traduzido automaticamente para o modelo de estado finito em que se verificam as propriedades desejadas, com o intuito de garantir uma correção antecipada.

Este trabalho teve por objetivo desenvolver um arcabouço para realizar a verificação de diagramas de temporização da UML, utilizando técnicas formais, em especial a verificação simbólica de modelos.

Palavras-chave: engenharia de *software*; verificação de modelos; UML; diagrama de temporização.

ABSTRACT

Errors and faults are introduced during the whole life cycle of a software product, and should ideally be corrected as soon as possible.

Several systems use UML timing diagrams as a means to specify system behavior. Guaranteeing in advance that specifications are correct leads to error-free modeling, avoiding error propagation to further development stages, in which correction costs are substantially higher.

In this work, we use symbolic checking to evaluate a provided specification through a timing diagram, which is automatically converted into a finite state model in which the desired properties are verified, to guarantee an early correction.

This work had the goal of developing a framework to perform the verification of UML timing diagrams using formal techniques, especially symbolic model checking.

Keywords: software engineering; model checking; UML; timing diagram

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE ABREVIATURAS E SIGLAS

BDD – *Binary Decision Diagram*

CTL – *Computation Tree Logic*

LTL – *Linear Temporal Logic*

MOF – *Meta Object Facility*

OBDD – *Ordered Binary Decision Diagrams*

OMG – *Object Management Group*

OMT – *Objetct-Oriented Software Engeneering*

OOSE – *Objetct-Oriented Software Engeneering*

SMV – *Symbolic Model Checker*

UML – *Unified Modeling Language*

VDM – *Vienna Development Model*

V&V – Verificação e Validação

W3C – *World Wide Web Consortium*

XMI – XML *Metadata Interchange*

XML – *Extensible Markup Language*

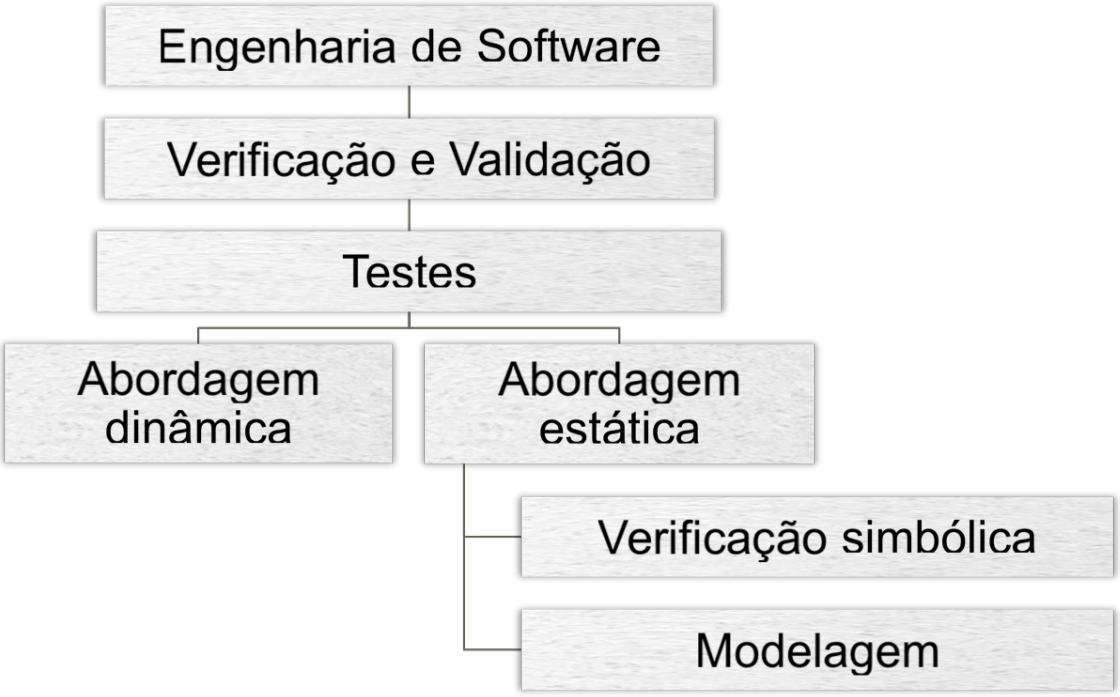
SUMÁRIO

1 INTRODUÇÃO

A Associação Brasileira das Empresas de *software*, em seu relatório anual, que apresenta as tendências dos produtos e serviços prestados para a área de *software*, informa que o mercado nacional movimentou, em 2013, o total de 10,7 bilhões de dólares, considerando um crescimento de 13,5%, se comparado ao ano de 2012 (ABES - ASSOCIAÇÃO BRASILEIRA DAS EMPRESAS DE SOFTWARE, 2014).

O valor movimentado e a importância da pesquisa demonstra que o *software* é um produto com grande visibilidade e que, se seguir tendências de anos anteriores, crescerá, por isso, o mercado de desenvolvimento exige cuidados e maior qualidade para que o produto não apresente erros.

Figura 1 – Conceitos imprescindíveis a este trabalho



Fonte: Elaborado pela autora

A Figura 1 demonstra os estudos necessários para que este trabalho seja realizado, seus conceitos serão explicados, e exemplificados, a seguir.

A Engenharia de *software* é uma área destinada aos cuidados imprescindíveis no desenvolvimento, que visa garantir qualidade na produção, desde a especificação de requisitos à evolução do produto final. Todas as etapas devem ser acompanhadas e desenvolvidas considerando padrões e boas práticas, visando melhoria contínua e diminuição dos esforços de produção (SOMMERVILLE, 2011).

Uma das técnicas desta área é a Verificação e Validação (V&V) que objetiva constatar se

os requisitos descritos para o produto atendem ao que foi modelado. É uma análise presente em todas as áreas do desenvolvimento, não sendo necessário aguardar a codificação do produto para aplicá-la. Esta técnica possui duas atividades distintas: a verificação, que atesta se o produto atende aos requisitos levantados, e a validação, que atesta se o mesmo atende às expectativas do cliente. Ambas procuram garantir qualidade e segurança à produção.

Os testes de *software* correspondem a uma parte da V&V, e são executados para garantir que erros não se propaguem às etapas posteriores, afinal, estes são inseridos durante todo o processo, e muitas vezes identificá-los e corrigí-los é uma tarefa desgastante. Estes testes podem ser automatizados ou manuais, e permitem duas abordagens: a dinâmica, que ocorre na execução do *software*, e a estática, que ocorre na especificação ou implementação.

Uma técnica auxiliar à abordagem estática de testes é a verificação simbólica de modelos (*Model Checking*) que evidencia se um modelo de estado finito satisfaz certas propriedades definidas. O propósito é avaliar expressões predefinidas, buscando afirmativas ou obtendo contraexemplos em casos de reprovações (CLARKE; GRUMBERG; PELED, 1999).

Para que a verificação simbólica seja aplicada é necessário que o sistema seja modelado e posteriormente traduzido para a linguagem específica de um verificador de modelos. Por modelar um sistema, entende-se representar graficamente o fluxo do produto, permitindo que os itens propostos sejam visualizados em vários níveis de abstração, para maior compreensão (MULLER et al., 2012).

Os modelos, também conhecidos como diagramas, são criados na fase de especificação e análise, e constantemente precisam ser inspecionados e testados para garantir que todo o processo atenda à proposta principal do negócio, pois o custo de encontrar e remover defeitos aumenta durante o ciclo de desenvolvimento de um produto de *software* (KANER; FALK; NGUYEN, 1999; MYERS, 2011; BLACK; VEENENDAAL; GRAHAM, 2012).

Visando permitir e ampliar a compreensão e interação das partes envolvidas no processo, a modelagem permite três perspectivas:

a) externa, que demonstra o ambiente do sistema modelado;
b) comportamental, que demonstra o comportamento e as funcionalidades;
c) e estrutural, que demonstra a arquitetura ou a estrutura de dados do sistema.

Para apoiar a modelagem existem linguagens de modelagem visual, como por exemplo a *Unified Modeling Language* (UML), que desde 1997 é amplamente utilizada para modelar, especificar, documentar e construir sistemas (BOOCH; RUMBAUGH; JACOBSON, 2012).

A UML possui inúmeras funcionalidades, o que a torna uma opção vantajosa para que os requisitos sejam devidamente levantados e compreendidos. Entretanto, não existe uma formalidade para a modelagem, por isso, não há como garantir que os modelos gerados estejam corretos.

1.1 Motivação

Considerando os aspectos apresentados anteriormente, e a possibilidade da introdução de erros durante o processo de desenvolvimento de um *software*, a motivação central deste trabalho é contribuir para a área de Engenharia de *software*, especificamente em V&V, provendo uma abordagem para a verificação de diagramas temporais da UML, utilizando métodos formais para detecção e correção de erros.

Myers (2011) cita que um erro não encontrado será propagado para as demais fases e seu custo será consideravelmente elevado, por isso, a prevenção de defeitos no início do processo gera menores problemas a serem encontrados nas fases posteriores.

Uma modelagem livre de erros beneficia o projeto em desenvolvimento, considerando que será possível levantar e implementar requisitos com mais clareza e objetividade. A V&V empenha-se em garantir que este benefício seja alcançado, e que as necessidades de um requisito sejam devidamente atendidas. As inúmeras revisões e inspeções são aplicadas durante o ciclo de vida de um produto.

Testar, revisar e inspecionar todas as combinações de um sistema não é tarefa trivial. Para minimizar a complexidade desta etapa pode-se utilizar a verificação simbólica de modelos, que é um método formal pertencente a etapa de V&V, direcionada principalmente a sistemas complexos (BLACK; VEENENDAAL; GRAHAM, 2012).

Através da verificação de modelos é possível aplicar uma notação matemática precisa, permitindo ao engenheiro de *software* verificar e explicitar os requisitos minuciosamente, assim as funcionalidades serão avaliadas de forma sistemática.

Na verificação simbólica o sistema modelado é representado como uma máquina de estados finitos, de forma a comprovar se as especificações atendem ao propósito principal do negócio. Para os casos em que atendem, é possível verificar um caminho de execução do estado inicial ao estado final definido, caso contrário, será apresentado um contraexemplo para confirmar o não atendimento ao que foi predeterminado.

Entretanto, verificar um sistema exige o conhecimento de técnicas e formalismos matemáticos que não são comuns aos desenvolvedores, por isso a verificação é pouco explorada, principalmente em sistemas triviais.

Para que a aplicação deste método seja possível, algumas ferramentas já foram desenvolvidas, e dão suporte a esta análise, permitindo automatizar o processo e não exigindo do engenheiro de *software* um conhecimento específico do formalismo, auxiliando nas mais diversas formas de análise, como exemplo, tem-se verificadores de modelos, *Vienna Development Model* (VDM) e outros (KHAN et al., 2012; FERRARI et al., 2010).

1.2 Problema

Considerando a necessidade de garantir que os sistemas sejam mais bem desenvolvidos e que apresentem menos erros em sua entrega, o problema que permeia este trabalho pretende responder as seguintes questões:

a) como garantir que os *softwares* sejam desenvolvidos de forma mais eficiente, minimizando os erros na entrega e durante o desenvolvimento do produto?
b) como inserir a utilização da verificação simbólica de modelos em um ambiente onde os profissionais não possuem um amplo conhecimento nesta área?
c) quais os principais questionamentos durante os testes e validações de *softwares*?

1.3 Objetivos

Considerando a importância de garantir qualidade ao desenvolvimento dos requisitos, com o intuito de melhorar o produto, e ainda a relevância da fase de levantamento e análise, em que a maioria dos modelos são definidos e avaliados, esta dissertação tem o objetivo geral de apresentar uma abordagem que efetue a tradução do diagrama de temporização da UML para uma linguagem formal, permitindo verificações automáticas sem o conhecimento específico e matemático do método.

Este objetivo geral divide-se nos seguintes objetivos específicos:

a) traduzir o formato XML *Metadata Interchange* (XMI) gerado a partir de um diagrama de temporização da UML para a linguagem formal de entrada do verificador de modelos NuSMV;
b) automatizar a tradução do diagrama para a linguagem formal. Desta forma, o engenheiro de *software* não terá obrigação de conhecer o formalismo matemático, e assim, poderá aplicar a verificação simbólica;
c) predefinir um conjunto de validações, visando possibilitar a avalição do diagrama de temporização;
d) avaliar a abordagem proposta através de estudos de caso que permitam a utilização do diagrama no ambiente proposto.

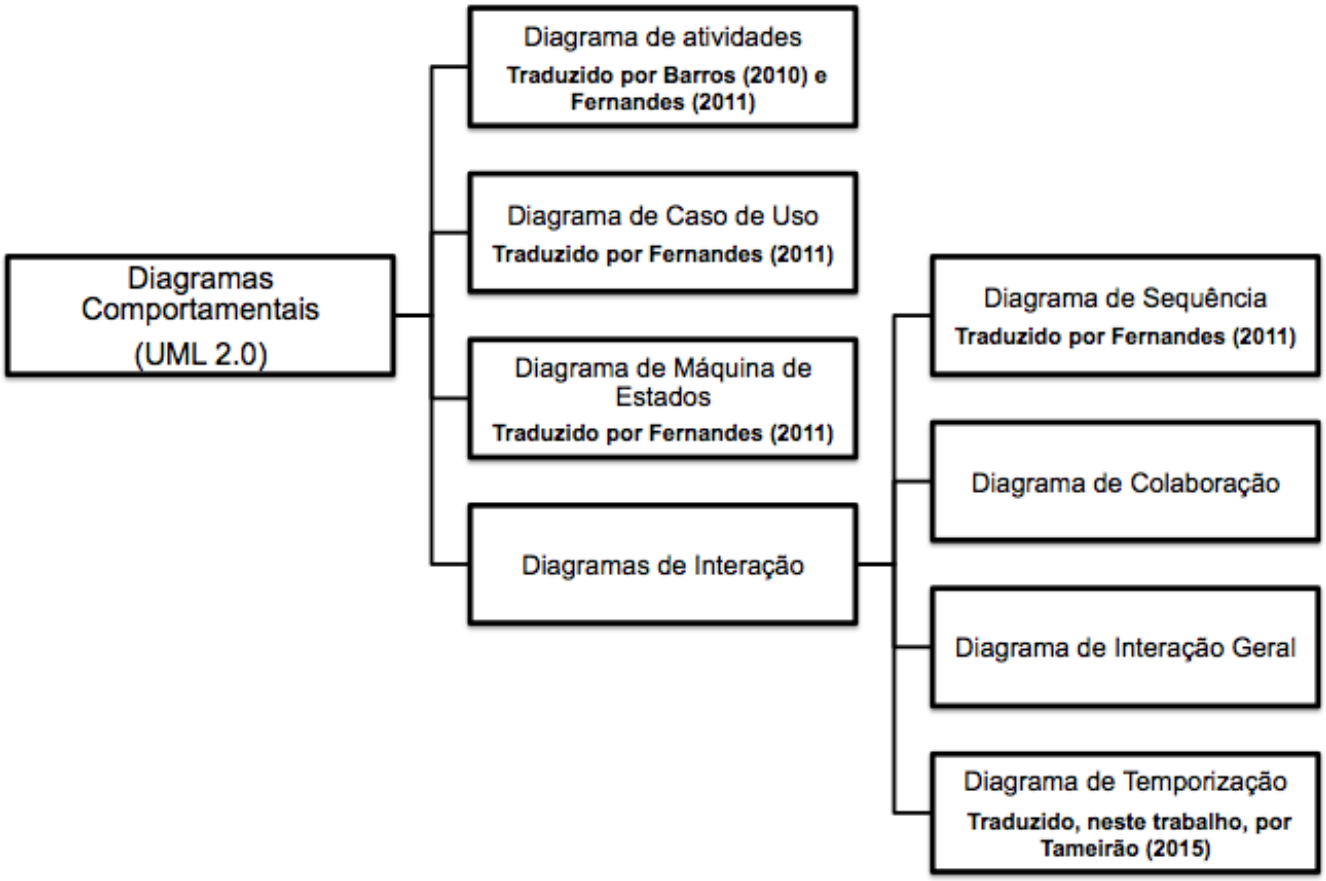
1.4 Principais contribuições

A principal contribuição é permitir aos engenheiros de *software* aplicarem a verificação simbólica de modelos em sistemas onde o tempo é fator determinante para sua execução, aumentando a eficiência do produto e minimizando os erros que seriam inseridos durante o processo, o que acarreta inúmeros benefícios financeiros à empresa.

Além de utilizarem a abordagem proposta para traduzir e verificar modelos de temporização da UML o presente trabalho complementa as seguintes dissertações já defendidas:

a) a de Barros (2010) que traduz o diagrama comportamental de atividades para a verificação simbólica de modelos;
b) a de Fernandes (2011), que complementou a de Barros (2010), introduzindo a tradução dos diagramas comportamentais de caso de uso, máquina de estados e sequência para a verificação simbólica de modelos.

Figura 2 – Diagramas UML x Trabalhos realizados



Fonte: Elaborado pela autora

Conforme Figura 2, é possível visualizar que os estudos anteriores, e este, visam permitir que os analistas possam minimizar os erros inseridos durante o processo de modelagem de dados, além de garantir mais efetividade no desenvolvimento do projeto.

Salienta-se que como resultado deste trabalho em 2014, na conferência *Software Engineering Research and Practice* (SERP), publicou-se o artigo *Symbolic Model Checking Applied to Timing Diagrams* (TAMEIRÃO; SONG, 2014).

1.5 Estrutura da dissertação

A presente dissertação é composta por seis capítulos:

a) No Capítulo 2, conceitos e embasamentos teóricos permitem a compreensão referente a metodologia utilizada, como por exemplo, UML e Verificação Simbólica de Modelos. Além da descrição de trabalhos relacionados que permitem a análise atual e as contribuições desta dissertação;
b) No Capítulo 3, apresenta-se a abordagem proposta, detalhando aspectos de mapeamento do diagrama de temporização para o modelo de verificação;
c) No Capítulo 4, estudos de caso são apresentados com o objetivo de validar a abordagem proposta;
d) Por fim, o Capítulo 5 conclui e destaca as principais contribuições efetuadas, apresentando também possíveis trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA E TRABALHOS RELACIONADOS

Este capítulo apresenta a fundamentação teórica que permitiu a realização deste trabalho.

Conforme explicado no capítulo anterior, o trabalho é desenvolvido a partir da junção dos conceitos de verificação de modelos e diagrama de temporização, portanto, todos os itens necessários para que este relacionamento seja efetuado serão descritos e detalhados nas seções a seguir da seguinte forma:

a) O diagrama de temporização pertence a proposta da UML 2.0, algumas informações referentes a esta proposta serão descritas, seguindo de uma atenção especial ao diagrama de temporização, que será primordial para este trabalho;
b) A tradução do diagrama para a verificação de modelos é feita a partir de um documento XMI, por isso, as informações referentes a este tipo de arquivo estão presentes nesta fundamentação;
c) Para compreender Verificação Simbólica de Modelos é necessário saber, além do conceito, sua forma de aplicação, por isso, este trabalho fundamenta este conceito, considerando a estrutura necessária para sua execução e a lógica de desenvolvimento utilizada para aplicá-la;
d) Ao final, serão relacionados os trabalhos que deram embasamento teórico para a realização deste.

2.1 Unified Modeling Language (UML)

Construir modelos que permitam a visualização do comportamento e das características de um produto de *software* corresponde ao que se denomina modelagem. Os modelos permitem a identificação das características e funcionalidades dos requisitos solicitados pelo cliente, bem como o planejamento de sua construção.

Frequentemente a modelagem utiliza alguma ferramenta para suporte às notações gráficas, isto permite facilidade e rapidez na construção dos diagramas. As linguagens de modelagens definem e permitem que tais notações gráficas sejam aplicadas e utilizadas no mercado de trabalho, com o intuito de aumentar a eficiência durante a construção e abstração dos requisitos de *software* apresentados durante as primeiras fases do ciclo de vida de um produto.

Entre 1970 e 1980 diversas linguagens foram propostas, porém, apenas algumas obtiveram destaque suficiente para serem utilizadas na fase de análise de dados, dentre elas estão *Booch Method*, proposto por Grady Booch, e que tem foco principal na fase de projeto, *Objetct-Oriented Software Engeneering* (OOSE), proposto por Ivar Jacobson, e tem foco principal em caso de uso e, por último, *Objetct-Oriented Software Engeneering* (OMT) proposto por James Rumbaugh e tem foco principal na análise de sistemas (FOWLER, 2003; ERICKSON; SIAU, 2013).

Cada linguagem se destacava positivamente em um item específico, portanto, em meados dos anos 90 alguns esforços indicavam que a união dos três métodos seria vantajosa.

Em 1994 a união dos modelos de Booch e Rumbaugh deu início a esta unificação, criando então a UML, homologada pela *Object Management Group* (OMG) em 1995, dando origem a primeira versão 0.8 que representa o método unificado.

Em 1995 Jacobson une-se a este projeto e, agora, os três métodos estão presentes na linguagem e deram início a versão 0.9 da UML.

Em 1997 a OMG homologou a versão 1.0, já incluindo os três métodos anteriores e, na ocasião, a UML apresentava grande visibilidade entre as empresas de desenvolvimento de *software*, por isso, empresas como *Digital Equipment Corporation*, *Hewlett-Packard*, I-*Logix*, *Intellicorp*, IBM, ICON *Computing*, MCI *Systemhouse*, *Microsoft*, *Oracle*, *Rational*, *Texas Intruments* e *Unisys* deram apoio e importantes sugestões (BOOCH; RUMBAUGH; JACOBSON, 2012).

Em 2005 foi liberada a versão 2.0 que corresponde a uma importante revisão da 1.0, incluindo recursos adicionais, que ampliam a visualização dos sistemas e podem ser aplicados em diversas etapas do desenvolvimento de *software*. Os documentos oficiais e atualizados são liberados pela OMG em seu site (OBJECT MANAGEMENT GROUP, 2014).

2.1.1 Visão geral

A UML é uma linguagem de modelagem visual que se propõe a documentar projetos e padrões de *software*. Além disso possui uma notação eficiente e de fácil compreensão (PILONE; PITMAN, 2006).

A versão 2.0 da UML, homologada em 2005, possui 13 diagramas que possibilitam representações distintas e particulares para um produto de *software*. A aplicação de seus diagramas depende das particularidades de cada projeto (OBJECT MANAGEMENT GROUP, 2014).

Utilizar diagramas distintos permite aos engenheiros de *software* uma melhor visualização do sistema em diferentes níveis de abstração, além de ampliar a compreensão dos envolvidos no desenvolvimento do produto, afinal, a UML possui uma linguagem abrangente, mas não exaustiva, o que possibilita que clientes e desenvolvedores de sistemas tenham a mesma visão do produto, ou parte dele (PENDER, 2004).

Os 13 diagramas da UML 2.0 atingem camadas distintas de abstração. Sua utilização deve considerar a necessidade e características do projeto e produto aplicado.

São classificados como estruturais e comportamentais, que possuem significados distintos, conforme verificado a seguir:

a) Diagramas Estruturais – seu objetivo é evidenciar a estrutura física do sistema modelado, documentando seus aspectos estáticos.
   Pertencem a esta classificação os diagramas de classe, objetos, implantação, componentes, estruturas compostas e pacotes;
b) Diagramas Comportamentais – seu objetivo é evidenciar o comportamento dos artefatos, documentando seus aspectos dinâmicos.
   Pertencem a esta classificação os diagramas atividade, caso de uso e máquina de estado.
   - Dentro da classificação comportamental existe uma subclassificação chamada interação, o intuito é indicar que estes diagramas, além de evidenciar o comportamento dos artefatos, também evidenciam a interação entre estes.
     Pertencem a esta subclassificação os diagramas de temporização, visão geral de interação, comunicação e sequência.

O Quadro 1 descreve e detalha cada um dos diagramas disponíveis, que são utilizados no mercado de trabalho para facilitar a compreensão e avaliação dos sistemas construídos.

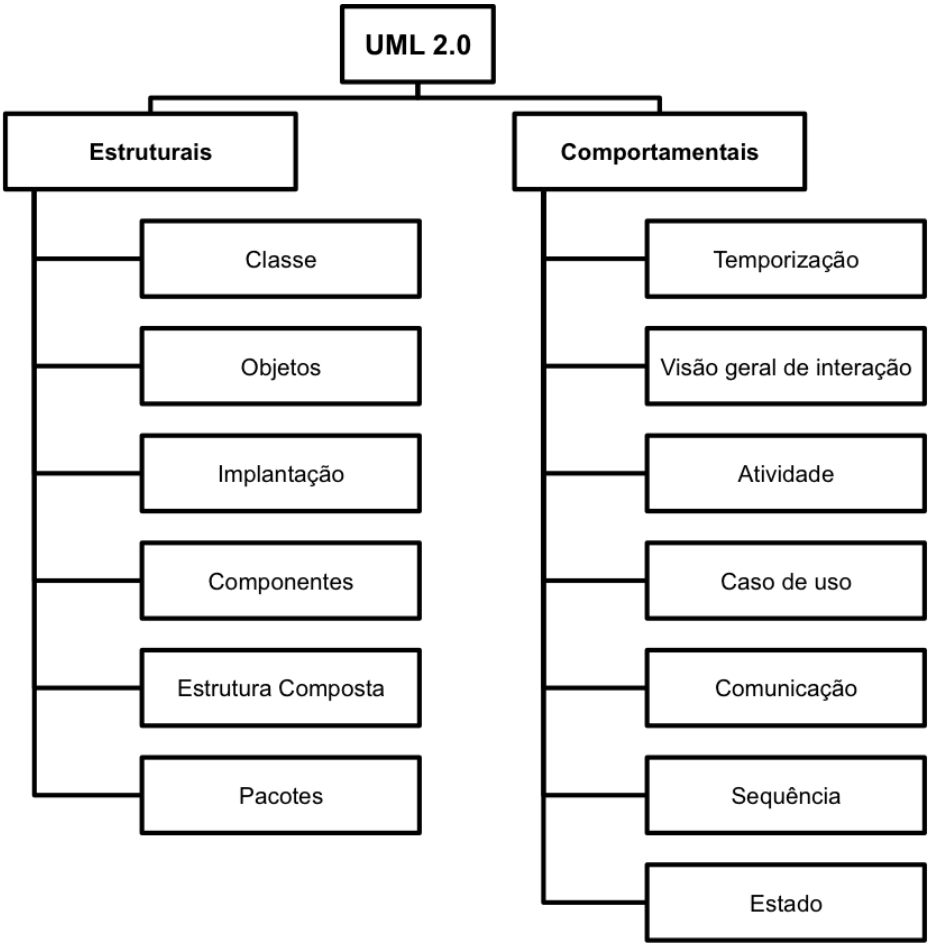
Quadro 1 – Diagramas da UML 2.0

Diagrama	Utilização
de classe	Evidencia os atributos, métodos e os relacionamentos existentes entre as classes do sistema.
de componentes	Evidencia os componentes de *software* e seus relacionamentos, associados às linguagens de programação.
de objetos	Complementa o diagrama de classe, informando e detalhando o que cada objeto armazena.
de estrutura composta	Demonstra a estrutura e os relacionamentos internos de um classificador.
de implantação	Modela o relacionamento entre recursos de infraestrutura, de rede e artefatos de sistemas.
de pacotes	Descreve a organização e dependência dos pacotes de sistema, inclusive os importados e as extensões.
de atividade	Exibe a sequência em que as atividades ocorrem. Correspondem às computações, workflows, fluxos e controles de objetos.
de caso de uso	Demonstra os requisitos do sistema e seu comportamento com os usuários externos.
de estado	Demonstra os estados de um objetos em relação às respostas a seus eventos.
de sequência	Representa o comportamento de um ou vários objetos em um contexto específico, a partir das mensagens trocadas entre eles.
de comunicação	Evidencia os relacionamentos entre os objetos.
de interação geral	Demonstra o fluxo de controle entre os diagramas.
de temporização	Demonstra as mudanças de estados existentes entre os objetos, através de um espaço de tempo definido.

Fonte: (BOOCH; RUMBAUGH; JACOBSON, 2012)

A Figura 3 evidencia a classificação de cada diagrama detalhado anteriormente no Quadro 1.

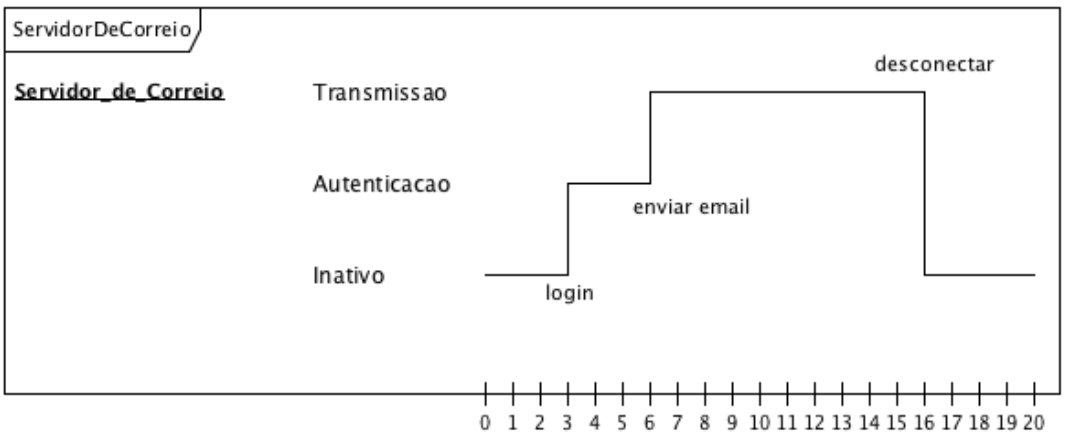
Figura 3 – Classificação dos diagramas da UML 2.0



Fonte: (BOOCH; RUMBAUGH; JACOBSON, 2012)

2.1.2 Diagrama de temporização

Figura 4 – Diagrama de temporização de um Servidor de Correio



Fonte: (PILONE; PITMAN, 2006)

O diagrama de temporização, ou de tempo, é utilizado quando existe a necessidade de modelar um artefato considerando um tempo de execução mínimo ou máximo, seu foco principal permeia a troca de informações entre os objetos e o tempo preestabelecido para que determinada ação aconteça. Muito utilizado em sistemas reais ou embutidos (PILONE; PITMAN, 2006).

A Figura 4 exemplifica um diagrama de temporização que simula a execução de um servidor de correio. Nele é possível perceber que o servidor inicia no tempo 0 (zero) na ação Inativo, após 2 tempos, ele entra na ação Autenticação, após 2 tempos, altera para ação de Trasmissão de dados e permanece nesta por 9 tempos, desconectando-se e retornando a ação de Inativo, totalizando 20 tempos de execução.

A seguir serão descritos os principais itens necessários para a construção deste diagrama.

Todo diagrama de temporização permite a inclusão de:

a) Linhas de vida (*lifelines*) – que correspondem a ação que o objeto deseja realizar, um diagrama pode conter mais de uma linha vida;
b) Estados condicionais (*state conditions*) – que correspondem a subdivisão de uma linha de vida, evidenciarão a divisão do tempo em cada ação específica;
c) Unidades de tempo (*time units*) – fragmentos do tempo em que a ação será executada. As mudanças e comunicações entre os objetos deverão ser explícitas em tempos distintos;
d) Instâncias de tempo (*time instance*) – não ficam explícitos no diagrama, porém, correspondem a junção de um estado condicional que alterará em um tempo específico.

O Quadro 2 associa os itens existentes em um diagrama de temporização e sua aplicação na Figura 4.

Quadro 2 – Itens do diagrama de temporização presentes no Servidor de Correio

Item do diagrama	**Exemplo no Servidor de Correio**
Linha de vida	Servidor_de_Correio
Estados condicionais	Inativo Autenticacao Transmissao
Unidades de tempo	De 0 a 20
Instâncias de tempo	O estado condicional do servidor de correio será alterado de inativo para autenticação no tempo 3; O estado condicional do servidor de correio será alterado de autenticação para transmissão no tempo 6; O estado condicional do servidor de correio será alterado de transmissão para inativo no tempo 16.

Fonte: Elaborado pela autora

2.2 XML Metadata Interchange (XMI)

Após desenhar o diagrama de temporização, em uma ferramenta de UML disponível no mercado, é necessário exportá-la para XMI, assim, o ambiente deste trabalho será capaz de efetuar a tradução do arquivo exportado para um arquivo SMV.

Segundo Pender (2004) o XMI é uma especificação criada e mantida pela OMG que visa permitir a troca de informações entre as ferramentas de modelagem de dados, e tornou-se padrão para ambientes distribuídos. É um documento baseado na *Extensible Markup Language* (XML) do *World Wide Web Consortium* (W3C) que oferece suporte ao metamodelo UML e a todo modelo baseado na *Meta Object Facility* (MOF).

O XMI é utilizado como padrão para evitar que cada empresa tenha seu próprio modelo, o que dificulta a troca e compreensão das informações. Permite agilidade de identificação dos dados enviados e também a tradução de seus objetos para linguagens como Java, Visual Basic, entre outras (OBJECT MANAGEMENT GROUP, 2014).

As ferramentas que permitem a exportação de UML para XMI consideram o mapeamento de cada elemento utilizado nos diagramas para a descrição e definição do XMI.

Este trabalho considera o padrão de mapeamento do diagrama de temporização a partir do XMI e efetua a tradução para a linguagem formal do verificador NuSMV.

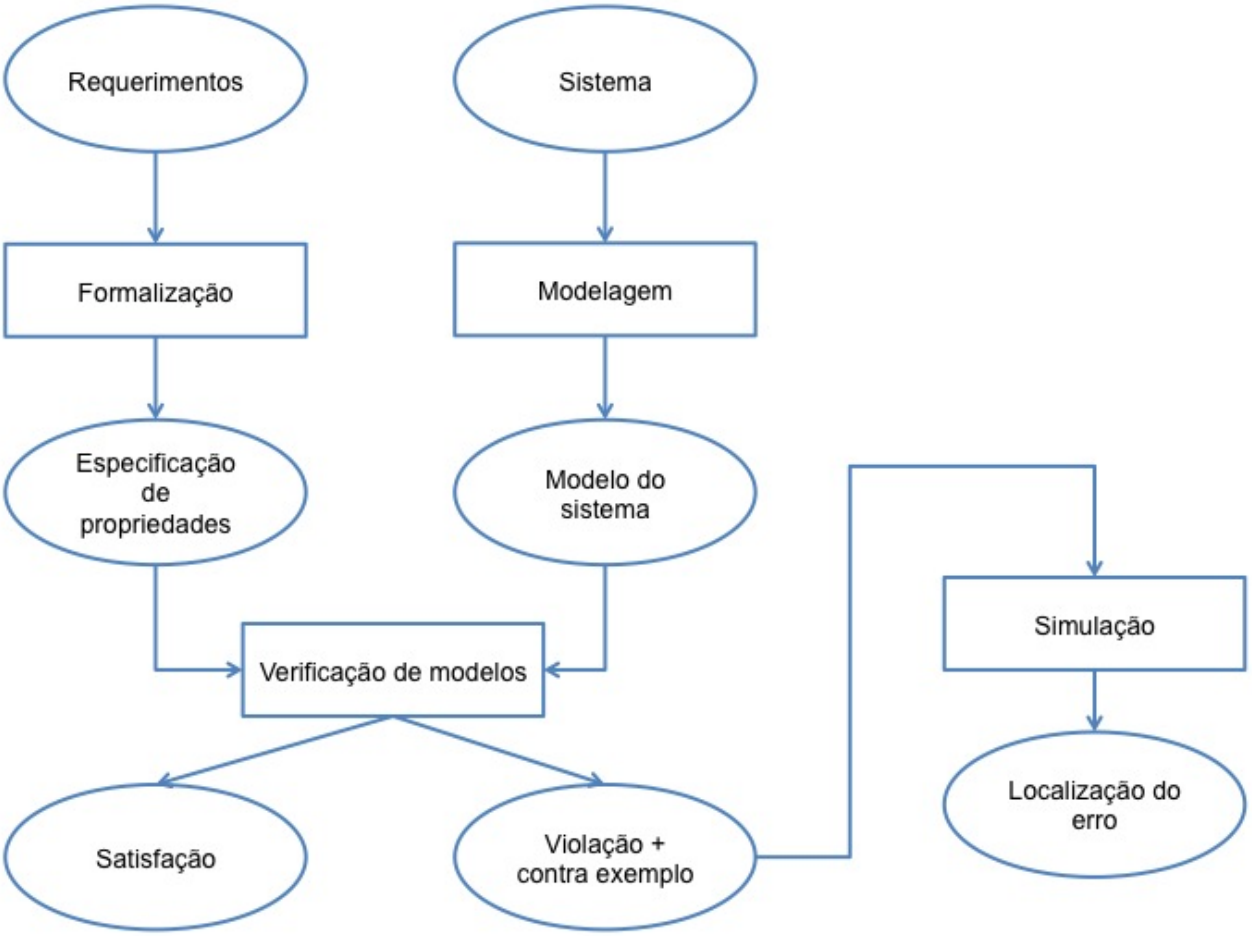
2.3 Verificação Simbólica de Modelos (*Model Checking*)

A busca de técnicas eficazes para minimização de erros no processo de desenvolvimento tem sido cada dia mais constante para *softwares* e *hardwares* complexos, gasta-se muito tempo corrigindo, ou buscando métodos para um desenvolvimento mais seguros e eficazes (BAIER; KATOEN, 2008).

Os métodos formais são fortemente recomendados para verificação e desenvolvimento de *softwares* e *hardwares*, permitindo uma análise mais abrangente das possibilidades de execuções efetuadas por estes. Uma das técnicas possíveis de ser aplicada é a verificação simbólica de modelos.

A verificação de modelos é muito eficaz quanto a detectar possíveis falhas em um produto, sua aplicação consiste em explorar todas as possibilidades do sistema, no intuito de encontrar alguma computação que não atenda ao que foi predeterminado pelo analista. É importante considerar que manualmente, ao utilizarmos testes comuns, não é possível essa abrangência. A Figura 5 demonstra a abordagem proposta para utilizar um verificador de modelos.

Figura 5 – Abordagem da verificação de modelos



Fonte: (BAIER; KATOEN, 2008)

A automatização da verificação de modelos considera que, a partir de um modelo de estado finito e uma propriedade previamente definida com apresentação formal, é possível aplicar uma análise sistemática que confirma se esta propriedade é válida, ou não, para o modelo avaliado.

Para que a verificação de modelo seja aplicada, é necessário seguir alguns passos, que são:

a) Fase de modelagem
   - Modelar (desenhar) o sistema utilizando a linguagem formal do verificador de modelos que pretende aplicar;
   - Formalizar a(s) propriedade(s) a ser(em) verificada(s) em linguagem formal, considerando a linguagem do verificador utilizado para análise.

b) Fase de análise (após execução das propriedades no verificador de modelos)
   - Se a propriedade for satisfeita corresponde a indicação de um caminho de execução que a justifique;

- Se a propriedade não for satisfeita, o verificador de modelos retornará um contraexemplo, que corresponde a um caminho de execução que invalida tal propriedade, ou seja, que comprova que a propriedade é falsa.

2.3.1 Estrutura Kripke

A estrutura Kripke corresponde a um grafo utilizado para representar o modelo que será verificado. Nele os vértices representam os estados possíveis e as arestas representam as transições existentes entre os estados citados (CLARKE; GRUMBERG; PELED, 1999). É utilizada para descrever sistemas concorrentes ou reativos com comportamento infinito.

Sharma (2013) define formalmente que a estrutura Kripke é uma quadrúpla K = (S, S_0, R, L), onde:

a) S é um conjunto finito de estados;
b) $S_0 \subseteq$ S é um conjunto de estados iniciais;
c) R $\subseteq$ S x S é a relação da transição de estados, onde, para todo estado s pertencente a S, existe um s' tal que (s, s') $\in$ R;
d) L : S $\rightarrow 2^{AP}$ é uma função que define cada estado s $\in$ S com os valores inicias 0 ou 1 associados a cada $p_i \in$ P.

A partir do estado inicial S_0 defini-se um caminho para a estrutura K considerando uma sequência infinita de estados $\sigma = s_0 s_1 ... s_n$ tal que $s_0 \in$ S e R(s_i, s_{i+1}) onde todo i ≥ 0.

Para representar a explicação acima podemos citar o exemplo do sistema de um forno microondas, que possui o número de estados finitos, como mostrado na Figura 6, e que é devidamente apresentado e explorado pelos autores Clarke, Grumberg e Peled (1999).

Sabe-se que um forno microondas trabalha com os possíveis estados *Start*, *Close*, *Heat* e *Error*. As definições para cada estado serão descritas a seguir:

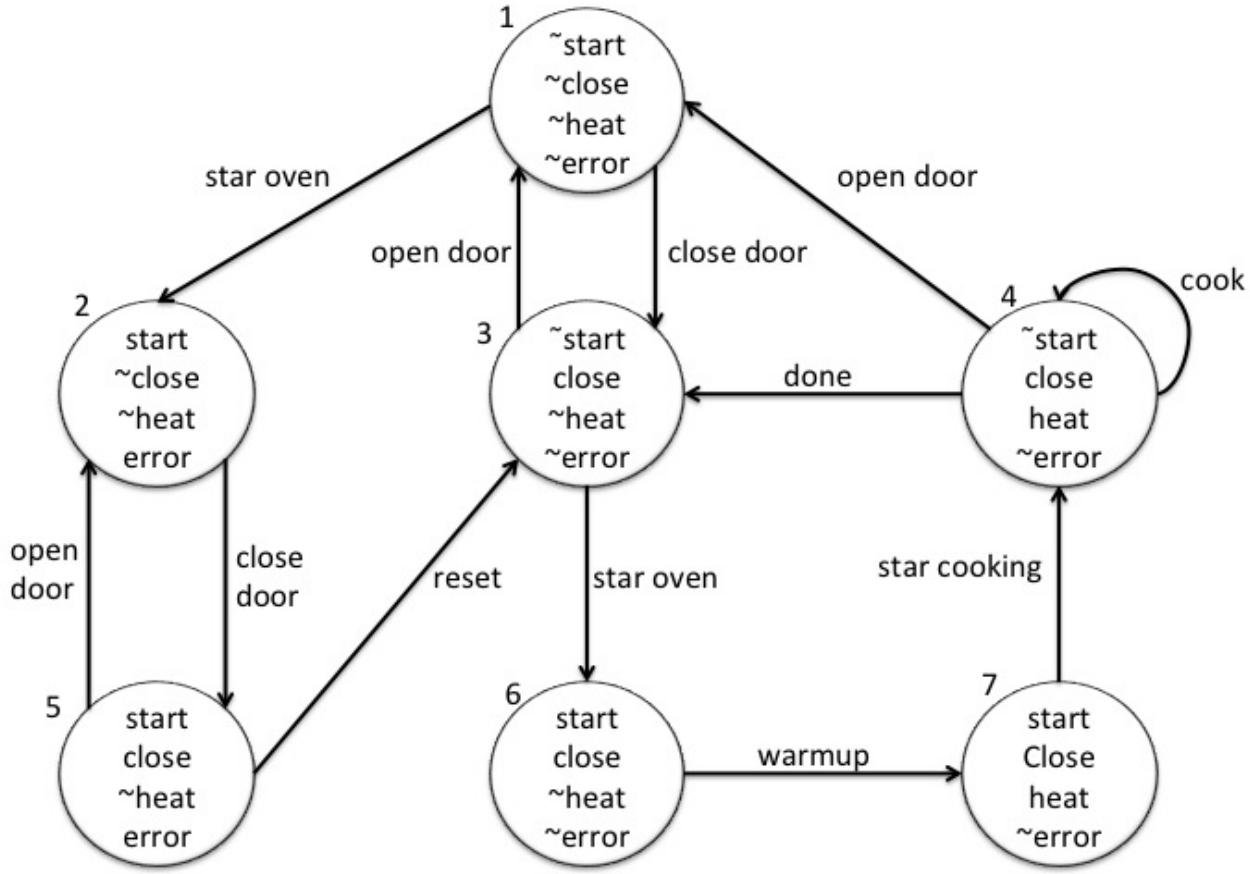
a) *start* – Representa o forno ligado;
b) *close* – Representa que a porta está fechada;
c) *heat* – Representa que o forno está aquecido;
d) *error* – Representa que houve algum erro durante o funcionamento.

Qualquer um destes possíveis estados, que também são conhecidos como variáveis, só poderá assumir um dos valores entre 0 (zero) ou 1 (um), onde o valor 0 indicará que a variável está negativa, e o valor 1 indica que ela está positiva.

A Figura 6 ilustra o sistema citado, e para melhor identificação considera-se que o símbolo $\sim$, em frente ao nome da variável, indica que ela possui o valor 0, e, caso contrario, a variável possui o valor 1.

Exemplo: A variável *Start* indica que o forno está ligado enquanto a variável ∼*Start* indica que o forno está desligado.

Figura 6 – Modelo de um forno microondas



Fonte: (CLARKE; GRUMBERG; PELED, 1999)

Ao contextualizar a Figura 6, considera-se que:

a) A partir do estado inicial 1 (um), onde o forno encontra-se com a porta aberta (∼*close*), desligado (∼*start*), não aquecido (∼*heat*) e sem erro de funcionamento (∼*error*), a partir deste estado é possivel efetuar as transições $R_{(1,2)}$ ou $R_{(1,3)}$;
b) Consideraremos a transição $R_{(1,2)}$, neste caso a situação será alterada do estado 1 para o estado 2, na Figura 6, $R_{(1,2)}$ está demoninada como "*start oven*", ou em tradução livre, iniciar cozimento.
   Neste estado, o forno encontra-se ligado (*start*), com a porta aberta (∼*close*), não aquecido (∼*heat*) e com erro de execução (*error*). A partir deste estado é possível efetuar a transição $R_{(2,5)}$;
c) Transição $R_{(2,5)}$, neste caso a situação será alterada do estado 2 para o estado 5, na Figura 6, $R_{(2,5)}$ está demoninada como "*close door*", ou em tradução livre, fechar porta.

Neste estado, o forno encontra-se ligado (*start*), com a porta fechada (*close*), não aquecido ($\sim$*heat*) e com erro de execução (*error*). A partir deste estado é possível efetuar a transição $R_{(5,3)}$;

d) Transição $R_{(5,3)}$, neste caso a situação será alterada do estado 5 para o estado 3, na Figura 6, $R_{(5,3)}$ está demoninada como "*reset*", ou em tradução livre, reiniciar.
Neste estado, o forno encontra-se desligado ($\sim$*start*), com a porta fechada (*close*), não aquecido ($\sim$*heat*) e sem erro de execução ($\sim$*error*). A partir deste estado é possível efetuar as transições $R_{(3,1)}$ e $R_{(3,6)}$;

e) Consideraremos a transição $R_{(3,1)}$, neste caso a situação será alterada do estado 3 para o estado 1, na Figura 6, $R_{(3,1)}$ está demoninada como "*open door*", ou em tradução livre, abrir porta. Este é o estado inicial apresentado nesta contextualização, o que nos leva ao início dela. Lembrando que a descrição explorou apenas uma situação possível ao forno microondas, possuindo ainda outros caminhos e estados possíveis.

Levando-se em conta o cenário apresentado na Figura 6, pode-se definir que a quadrúpla da estrutura K = (S, S_0, R, L) para este modelo é a seguinte:

a) $S=i \mid 0 \leq i \leq 9$;
b) $S_0 = 1$;
c) R=(1,2), (1,3), (2,5), (5,2), (5,3), (3,1), (3,6), (6,7), (7,4), (4,3), (4,1), (4,4);
d) $L_{(1)}$ = (0,0,0,0), $L_{(2)}$ = (1,0,0,1), $L_{(3)}$ = (0,1,0,0), $L_{(4)}$ = (0,1,1,0), $L_{(5)}$ = (1,1,0,1), $L_{(6)}$ = (1,1,0,0), $L_{(7)}$ = (1,1,1,0).

2.3.2 Diagrama Binário de Decisão

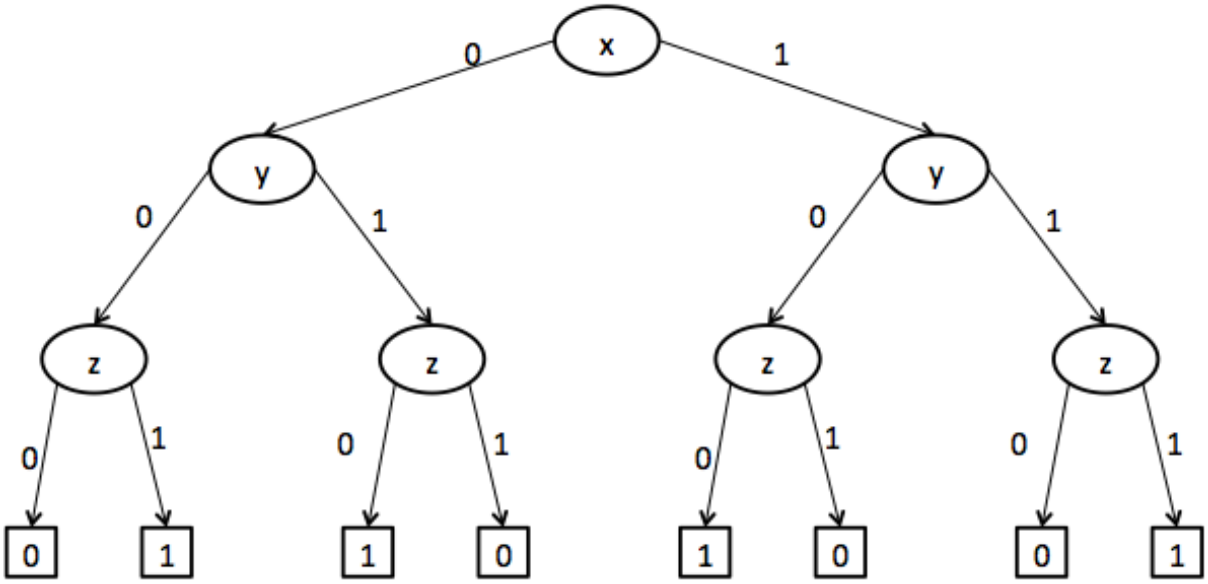
Um diagrama binário de decisão, do termo *Binary Decision Diagram* (BDD), foi definido e formalizado por Bryant (1986) e corresponde a uma árvore de decisão utilizada para representar estados finitos de uma computação de fórmulas booleanas, onde 0 (zero) é falso e 1 (um) é verdadeiro.

Um BDD é um grafo acíclico direcionado com vértices terminais e não terminais, cada vértice simboliza uma variável V em uma fórmula booleana e possui duas arestas de saída e uma entrada, as exceções são: o vértice raiz, este possui apenas arestas de saída, e os vértices folha, que possuem apenas arestas de entrada. O valor falso (V = 0) é atribuído a variável através da aresta de saída localizada a esquerda do vértice, o valor verdadeiro (V = 1) é atribuído a variável através da aresta de entrada localizado a direita (SEGALL; TZORF-BRILL; FARCHI, 2011).

Os BDD's passam por um processo de redução, com o intuito de maximizar os compartilhamentos de nós, onde todos os nós filhos e subárvores isomórficas são combinadas, reduzindo assim a quantidade de caminhos para a representação de uma fórmula booleana, que é representada por um ponteiro para a o seu nó raiz. A simplificação existe quando os valores possíveis indicam

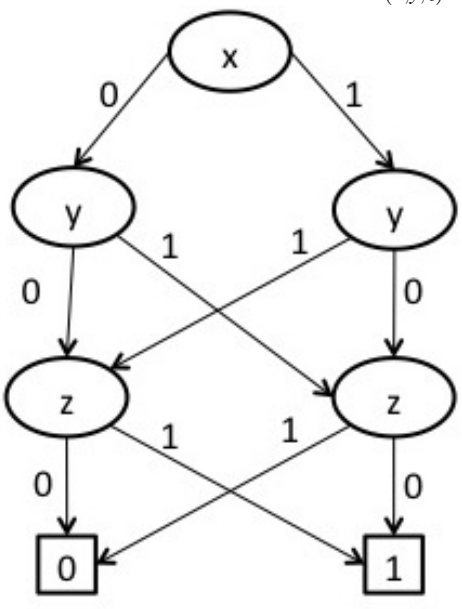
que o resultado será falso, onde a fórmula não foi satisfeita, ou verdadeiro, onde a fórmula foi satisfeita.

Figura 7 – BDD da fórmula $f_{(x,y,z)} = x \oplus y \oplus z$



Fonte: (CLARKE; GRUMBERG; PELED, 1999)

Figura 8 – BDD reduzido da fórmula $f_{(x,y,z)} = x \oplus y \oplus z$



Fonte: (CLARKE; GRUMBERG; PELED, 1999)

A Figura 7 exemplifica a construção de um BDD para a fórmula booleana $f_{(x,y,z)} = x \oplus y \oplus z$, enquanto a Figura 8 exemplifica a redução do BDD para a mesma fórmula.

Segundo Bryant (1986) um BDD pode crescer de forma exponencial, dependendo da ordem aplicada a suas variáveis, isto porque ele é sensível a esta ordenação, o que o torna um

problema NP-Completo. Heurísticas já foram aplicadas para otimizar este processo, porém, existem casos onde a ordenação manual é necessária.

2.3.3 Lógica Temporal

Para aplicar propriedades através da estrutura Kripke, explicada na Subseção 2.3.1 deste capítulo, é necessário aplicar e conhecer a lógica temporal, que é um formalismo utilizado para descrever sequências atômicas e transições entre estados de uma computação. As propriedades são especificadas utilizando operadores especiais e são consideradas sequências específicas de escrita, não sendo determinado o tempo exato de cada item (BAIER; KATOEN, 2008).

Para aplicar e exemplificar a lógica temporal, podemos citar dois tipos específicos, que são: *Linear Temporal Logic* (LTL), que considera uma perspectiva de tempo linear e a *Computation Tree Logic* (CTL) que considera uma perspectiva sob uma árvore de estados ramificada.

O verificador NuSMV, utilizado neste trabalho, explora ambas as lógicas citadas, porém, a abordagem proposta utiliza a lógica temporal CTL e alguns comandos da LTL. Essas são duas lógicas complementares, pois uma propriedade descrita e testada através dos comandos da LTL não pode ser verificada pela linguagem CTL, e vice versa.

2.3.3.1 Linguagem LTL

A linguagem LTL permite verificar caminhos lineares em uma estrutura Kripke de trajetória infinita, $\sigma = s_0 s_1 s_2 s_3 \ldots$, onde s_0 corresponde ao estado inicial, não sendo obrigatoriamente o estado da estrutura inicial testada.

O resultado sera verdadeiro em casos onde todos os caminhos testados sejam verdadeiros, a partir do estado verificado.

Clarke e Lerda (2007) descreveram os operadores para que estes sejam avaliados em um único caminho, da seguinda forma:

a) Fp (No future p) – A propriedade p é satisfeita em qualquer instante de tempo futuro, no caminho verificado;
b) Gp (Globalmente p) – A propriedade p é satisfeita em todos os instantes de tempo futuro (globalmente satisfeita);
c) p $\bigcup$ q (p até que q) – A propriedade p é satisfeita até que a propriedade q seja satisfeita;
d) Xp (próximo p) – A propriedade p é satisfeita no próximo estado (imediatamente).

2.3.3.2 Linguagem CTL

A linguagem CTL permite verificar caminhos não lineares, ou seja, podendo ser aplicado a partir do estado inicial ou de qualquer estado. Corresponde a uma combinação dos operadores, já citados, LTL e da *Branching-Time Logic*, que quantificam caminhos.

Clarke e Lerda (2007) descreveram os operadores da seguinda forma:

a) E (Existe um caminho) – A propriedade p é verdadeira sempre que existir uma trajetória a partir do primeiro estado de s tal que p seja verdadeiro;
b) A (Todos os caminhos) – A propriedade p é verdadeira sempre que todas as trajetórias possíveis, a partir do estado de s, satisfaçam p.

2.3.3.3 CTL e LTL

As linguagens CTL e LTL são subconjuntos da linguagem CTL$*$, e conforme explicado anteriormente, possuem comandos distintos para as verificações de suas propriedades.

LTL permite a utilização dos operadores X, F, G, $\bigcup$ ou R enquanto CTL utiliza os quantificadores A e G, quando combinados, permite criar propriedades de verificação das fórmulas da estrutura Kripke. Como por exemplo:

a) AX ou EX;
b) AF ou EF;
c) AG ou EG;
d) AU ou EU;
e) AR ou ER.

Por isso, a utilização de CTL$*$ amplia a aplicação e verificação das propriedades.

A seguir é possível verificar algumas aplicações, propostos do Clarke, Grumberg e Peled (1999), executadas na Figura 6, computação de um forno microondas:

a) EF($Start \wedge \sim Ready$) – Existe um estado onde, no futuro, o microondas está iniciado e não funcionando;
b) AG($Start \wedge Close$) – É globalmente verdade que sempre que o microondas estiver iniciado a porta estará fechada;
c) EX($Heat \wedge \sim Close$) – Existe algum estado, em que no próximo estado, o microondas estará esquentando com a porta aberta.

Estes e outros exemplos podem ser aplicados as fórmulas impostas.

A abordagem deste trabalho considera que o engenheiro de *software*, mesmo não conhecendo os formalismos matemáticos, poderá transformar suas perguntas e dúvidas referentes ao diagrama desenhado em fórmulas utilizando a linguagem CTL$*$.

2.3.4 A Linguagem do verificador de modelos NuSMV

Segundo Cavada et al. (2015), o NuSMV é um verificador de modelo simbólico criado a partir da reengenheria do *Symbolic Model Checker* (SMV).

Sua funcionalidade principal é permitir a representação de sistemas finitos síncronos e assíncronos, através da aplicação de fórmulas booleanas criadas com a linguagem CTL$*$.

A escrita é modularizada, permitindo reutilizar módulos escritos e chamados a partir de um módulo principal.

Aplica técnicas simbólicas fundamentadas em diagramas ordenados, baseados em *Ordered Binary Decision Diagrams* (OBDD) que permite executar de forma eficiente e precisa verificações com um grande número de estados (LOMUSCIO; PECHEUR; RAIMONDI, 2007).

E sua linguagem permite uma descrição simbólica de uma estrutura Kripke.

2.3.4.1 Módulo Principal (Module main)

Considere o exemplo do Quadro 3, que descreve o módulo principal – linhas 1 a 37 – e uma especificação CTL – linha 39 e 40 – referente ao diagrama de temporização do Servidor de Correio, a partir da abordagem proposta neste trabalho.

Na linguagem, as fórmulas da estrutura Kripke podem ser definidas através de variáveis do tipo booleanas, enumeradas ou inteiras. A declaração destas variáveis é feita na instrução identificada como VAR – linhas 2 a 4.

No exemplo é possível perceber que foram criadas duas variáveis enumeradas, a primeira identificada como “Servidor_de_Correio”, que pode receber os valores Inativo, Autenticacao e Transmissao, que correspondem aos estados possíveis de um servidor de *email*, e a segunda identificada como “temporizador”, que pode receber os valores de 1 a 20 e corresponde as alterações de tempos indicadas no diagrama.

A instrução ASSIGN evidencia as relações e os estados iniciais da fórmula Kripke e ocorrem em paralelo – linha 5 a 7.

A inicialização é efetuada através do comando *init*. No exemplo, a variável “Servidor_de_Correio” é inicializada com o valor Inativo, indicando que o servidor de *email* encontra-se inativo, e a variável “temporizador” é inicializada com o valor 0, que corresponde ao primeiro valor possível entre as unidades de tempo, conforme observado no diagrama da Figura 4. Em casos onde a inicialização não seja indicada o compilador indicará valores aleatórios.

A alternância de estados é executada a partir do comando *NEXT* sendo sempre finalizada com o comando *esac*, e permitindo a utilização de um ou mais comandos deste tipo, lembrando que estes serão executados de forma estrutural – linha 9 a 37.

Quadro 3 – Exemplo do módulo principal do Servidor de Correio

```
1 MODULE main
2 VAR
3    Servidor_de_Correio : {Transmissao, Autenticacao, Inativo};
4    temporizador : {0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19, 20};
5 ASSIGN
6    init(Servidor_de_Correio) := Inativo;
7    init(temporizador) := 0;
8
9    next(temporizador) := case
10       temporizador = 0 : 1;
11       temporizador = 1 : 2;
12       temporizador = 2 : 3;
13       temporizador = 3 : 4;
14       temporizador = 4 : 5;
15       temporizador = 5 : 6;
16       temporizador = 6 : 7;
17       temporizador = 7 : 8;
18       temporizador = 8 : 9;
19       temporizador = 9 : 10;
20       temporizador = 10 : 11;
21       temporizador = 11 : 12;
22       temporizador = 12 : 13;
23       temporizador = 13 : 14;
24       temporizador = 14 : 15;
25       temporizador = 15 : 16;
26       temporizador = 16 : 17;
27       temporizador = 17 : 18;
28       temporizador = 18 : 19;
29       TRUE : 0;
30    esac;
31
32    next(Servidor_de_Correio) := case
33       temporizador >= 0 & temporizador <= 2 : Inativo;
34       temporizador >= 3 & temporizador <= 6 : Autenticacao;
35       temporizador >= 7 & temporizador <= 14 : Transmissao;
36       temporizador >= 15 & temporizador <= 19 : Inativo;
37    esac;
38
39 SPEC AG ((Servidor_de_Correio = Autenticacao) -> AF (temporizador > 5))
40 SPEC AX ((Servidor_de_Correio = Inativo) -> AG (temporizador > 10))
```

Fonte: Elaborado pela autora, gerado através da abordagem

Observando o exemplo da linha 34, é possível perceber que a variável Servidor_de_Correio receberá o valor Autenticacao apenas se o temporizador estiver no intervalo de 3 a 6, inclusive.

A instrução SPEC descreve a propriedade a ser testada no modelo, esta foi descrita através do ambiente proposto neste trabalho – linhas 39 e 40.

A confirmação, ou um contraexemplo, para a propriedade é apresentada a partir da execução desta instrução.

Para as linhas 39 e 40, do módulo principal, o verificador apresentará as seguintes respostas:

a) Linha 39 –

SPEC AG ((SERVIDOR_DE_CORREIO = AUTENTICACAO) -> AF (TEMPORIZADOR > 5))

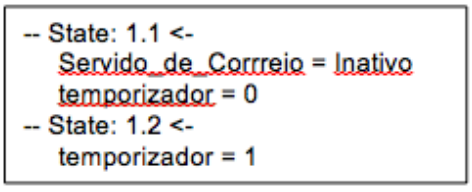
O exemplo avalia se, para todos os estados do modelo, sempre que a variável Servidor_de_Correio possuir o valor Autenticacao o temporizador será maior que 5.
Ao executar o verificador retornará true, indicando que a propriedade será satisfeita;

b) Linha 40 –

SPEC AX ((SERVIDOR_DE_CORREIO = INATIVO) -> AG (TEMPORIZADOR > 10))

O exemplo avalia, se para todos os estados do modelo, no próximo estado em que o Servidor_de_correio estiver na ação Inativo há a implicação de que o temporizador sempre será maior que 10.
Ao executar o verificador retornará falso, indicando que existe alguma situação que não confirma a propriedade. O retorno será exibido em estados e suas execuções, conforme Figura 8.

Figura 9 – Retorno do verificador para a propriedade indicada



Fonte: Elaborado pela autora.

2.4 Trabalhos relacionados

Considerando a importância do mercado de *software* e o crescimento das descrições de sistemas utilizando diagramas da UML, este capítulo tem o intuito de apresentar alguns estudos efetuados na área de V&V, modelagem UML e aplicação de verificação de modelos.

O mercado de desenvolvimento de sistemas tem se mostrado cada dia mais exigente em

sua entrega de produtos, portanto, é essencial que verificações eficientes sejam aplicadas durante o processo de produção. Com isto, a fase de análise é importante pois diversos diagramas são descritos e ampliam a compreensão do negócio para os analistas e clientes envolvidos.

Ainda assim, muitos erros são encontrados, diagramas são escritos de forma incorreta e, por vezes, deixam de ser reescritos em fases posteriores a análise. Uma das formas de garantir segurança na entrega é efetuar testes e revisões eficientes. Pinheiro (2012), por exemplo, apresenta ferramentas que apoiam a automatização de casos de testes, a partir do diagrama de máquina de estados, segundo a autora, a geração automática de testes faz com que o número de erros sejam minimizado, ocasionando ganhos para os envolvidos.

Porém, apenas testes comuns não tem se mostrado suficientes na correção de erros, por isso, aplicar metódos formais tem sido uma forma eficaz de garantir segurança às diversas revisões aplicadas aos sistemas. Mesmo com inúmeras vantagens, tais métodos não são amplamente explorados em sistemas não críticos, devido às inúmeras exigências de conhecimentos técnicos necessários a sua aplicação.

Uma das técnicas formais é a verificação de modelos. Entretanto, grande parte dos profissionais do mercado de trabalho não possuem estes conhecimentos, o que impossibilita que seja aplicada (BRADLEY, 2011).

Bhaduri e Ramesh (2008), por exemplo, concluem que, grande parte das abordagens só permite uma aplicação da verificação em diagramas de estados após a tradução do sistema avaliado para a linguagem do verificador, fato este que demanda conhecimento específico e dificulta sua utilização. Mesmo assim, ainda consideram vantajoso utilizar a técnica.

Este trabalho reconhece as inúmeras vantagens de aplicar a verificação aos sistemas comuns, e diferencia-se do trabalho de Bhaduri e Ramesh por utilizar o diagrama de temporização em sua tradução.

A associação da modelagem de dados e verificação de modelos é comum em estudos recentes. Por exemplo, a aplicação da verificação a diagramas da UML foi efetuada por Gagnon, Mokhati e Badri (2008), em que autores efetuam a tradução dos diagramas de classe, estados e comunicação para a linguagem orientada a objetos MAUDE (CLAVEL et al., 2011). Após a tradução, executa-se a verificação, também contida na linguagem, considerando o resultado e as propriedades predefinidas manualmente pelo analista. Assim como o trabalho anterior, Chama, Elmansouri e Chaoui (2012) efetuam a tradução dos diagramas de classe, estados e comunicação para a linguagem MAUDE, associando a utilização de grafos para ampliar e formalizar as necessidades e regras definidas. Em ambos, os autores evidenciam a contribuição no intuito de facilitar a utilização da verificação em aplicações diversas durante o desenvolvimento de *software*.

Neste trabalho também se utiliza modelagem UML como apoio, porém, o verificador de

modelos utilizado para análise será o NuSMV e o foco principal serão sistemas que necessitam de resultados em tempo real.

Zhao et al. (2010) mapearam cada item do diagrama de estados para a linguagem do verificador de modelos PRISM (KWIATKOWSKA; NORMAN; PARKER, 2011). O texto evidencia a forma como foi definido e traduzido os objetos do diagrama na linguagem do verificador. O trabalho ressalta a importância de incluir uma validação na fase inicial do desenvolvimento de *software* evitando perdas em projetos. Como trabalho futuro sugere a utilização de traduções a partir de descrições XMI.

As descrições XMI já são incorporadas no trabalho atual para modelagem dos diagramas de temporização.

O diagrama de temporização, foco principal deste trabalho, foi utilizado por Amla et al. (2000), que através de uma ferramenta desenvolvida (*Regular Timing Diagram Translator* - RTDT) efetua a tradução do diagrama para o verificador formal COSPAN (HARDIN; HAR'EL; KURSHAN, 1996). Os autores destacam que esta tradução permite verificar e acompanhar as propriedades de tempo descritas e determinantes ao projeto. O trabalho apresentado ainda demanda grande conhecimento técnico, por isso, apesar de utilizar o mesmo diagrama como base de tradução, diferencia-se do atual.

A técnica de verificação também foi aplicada no trabalho de Barros (2010), em que ele considera o sequenciamento de fluxo de trabalho e opta por validar este detalhamento antes de uma implementação, possibilitando ampliar e visualizar as possíveis falhas do trabalho realizado. A principal contribuição evidenciada é a diminuição dos erros encontrados na fase de modelagem de produto.

Fernandes (2011), com o intuito de detectar erros ainda na fase de análise de *software*, e aplicando as técnicas de verificação formal, efetuou a tradução dos diagramas de atividades, de estado e de sequência para o verificador de modelos NuSMV. Através de uma *interface* que permite parametrização personalizada e automatização do processo de validação, apresentou e ampliou a possibilidade de aplicação da técnica de validação em projetos de *software*.

Este trabalho diferencia-se dos dois anteriores por associar o diagrama de temporização da UML à verificação de modelos, no intuito de evitar erros em futuros trabalhos e consequentemente permitir mais abstração dos projetos que utilizam o tempo como fator determinante para suas execuções.

É importante ressaltar que os trabalhos de Barros (2010) e de Fernandes (2011) fazem parte do mesmo grupo de estudos do trabalho atual, e se complementam.

O Quadro 4 descreve e relaciona os trabalhos citados anteriormente.

Quadro 4 – Trabalhos Relacionados

Autor(es)	Ano	Diagrama de UML	Verificador de Modelos	Descrição
Amla, N. et al	2000	de Temporização	Cospan	Apresenta um algoritmo que decompõe o diagrama de temporização, através de uma ferramenta própria. Permintindo a tradução ao verificador, mesmo que tenha sido modelado de maneira informal.
Gagnon, P. Mokhati, F. Badri, M.	2008	de Classe de Estados de Comunicação	Maude	Quadro de apoio a verificação de modelos, utilizando o verificador MAUDE, aplicado aos diagramas citados.
Zhao, Y. et al	2010	de Estados	Prism	Regras de mapeamento para aplicação da verificação de modelos ao diagrama de estados.
Barros, C.	2010	de Atividades	NuSMV	Aplica verificação de modelos ao diagrama de atividades, através de um abordagem própria.
Fernandes, F.	2011	de Atividades de Sequência de Estados	NuSMV	Aplica verificação de modelos aos diagramas citados, através de um abordagem própria.
Pinheiro, A.C.	2012	de Estados	-	Defende a geração automática de testes, a partir de um diagrama.
Bhaduri, P. Ramesh, S.	2012	de Estados	-	Levantamento das abordagens existentes onde se aplica verificação de modelos a diagrama de estados.
Chama, W. Elmansouri, R. Chaqui, A.	2012	de Classe de Estados de Comunicação	Maude	Quadro de apoio a verificação de modelos, utilizando o verificador MAUDE, aplicado aos diagramas citados. Utilizando abordagem de grafos.
Tameirão, A.	2015	de Temporização	NuSMV	Aplica verificação de modelos a diagramas de temporização, através de um abordagem própria, permitindo a tradução para o verficador citado.

Fonte: Elaborado pela autora

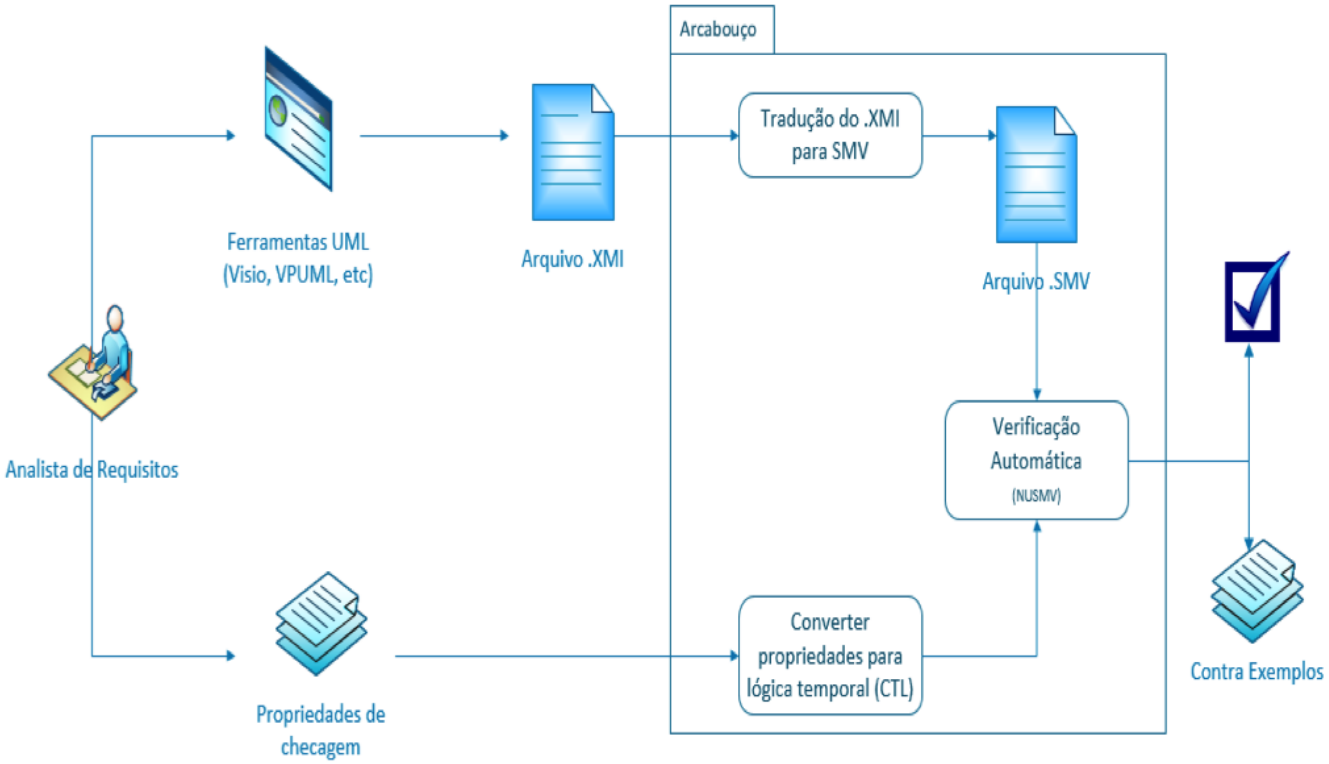
Este capítulo apresentou a fundamentação teórica, imprescindível para a compreensão do trabalho efetuado, descrevendo termos como UML, XMI, estrutura Kripke, dentre outros. Relata também os trabalhos defendidos na área de verificação de modelos e modelagem UML, bem como evidencia a diferença destes para o atual.

O próximo capítulo apresentará a metodologia utilizada para a abordagem citada.

3 METODOLOGIA

Este capítulo apresenta a metodologia aplicada nas validações do diagrama de temporização da UML, utilizando a interface desenvolvida pela autora para gerar o arquivo SMV, que será executado no verificador de modelos NuSMV. A Figura 10 ilustra o processo.

Figura 10 – Metodologia proposta na interface



Fonte: Elaborado pela autora

A metodologia foi desenvolvida em algumas etapas distintas e são descritas da seguinte forma:

a) Utilizando uma ferramenta de modelagem da UML, um engenheiro de *software* deve desenhar o diagrama de temporização idealizado.
   Para este trabalho os exemplos foram desenvolvidos na ferramenta VPUML, porém várias outras estão disponíveis no mercado e podem ser utilizadas (e.g. *Enterprise* UML, *Visio*, *Astha*, etc);
b) Através da ferramenta escolhida, o diagrama de temporização desenhado pelo engenheiro deve ser exportado para XMI, permitindo que através do ambiente proposto o engenheiro possa efetivar as traduções;
c) O profissional deve definir as propriedades que serão checadas – via *interface*. Estas propriedades correspondem às regras que serão aplicadas à verificação. O detalhamento e exemplificação do processo serão descritos na Seção 3.2;

d) O arquivo XMI será utilizado na tradução do modelo para o modelo de verificação;
e) Incorporar ao modelo de verificação, que corresponde ao diagrama de temporização, as propriedades a serem verificadas;
f) O arquivo gerado será executado no verificador de modelos NuSMV para obtenção das respostas correspondentes às propriedades.
É importante considerar que apenas duas respostas são possíveis, uma resposta positiva, ou seja, confirmando que a propriedade é verdadeira e existe uma computação executável que a sustente, e a outra é falsa, ou seja, existe uma execução que contradiz a propriedade, e neste caso o verificador apresentará um contra exemplo para justificar a negativa.

3.1 Tradução do diagrama

O arquivo SMV será criado considerando as etapas citadas, conforme quadro 5.

Quadro 5 – Tradução do diagrama de temporização para a linguagem do NuSMV

Objeto do diagrama	**Correspondente no arquivo do NuSMV**	**Sintaxe**
Linha de vida	Variáveis do modelo	{contidas no bloco VAR} VAR Linha_vida_1 :{...}; Linha_vida_2: {...}; Linha_vida_n: {...};
Estados condicionais	Mudanças de transações ocorridas nas variáveis geradas, a partir das linhas de vida	{contidos no bloco VAR} VAR Linha_vida_1 : {estado_1, estado_2, estado_n}; Linha_vida_2: {estado_1, estado_2, estado_3, estado_n}; Linha_vida_n: {estado_1, estado_n};
Unidades de tempo	Representam as mudanças temporais que podem ocorrer no diagrama	{contidas no bloco VAR – são os valores possíveis de uma variável denominada temporizador} VAR temporizador: {unid_1, unid_2, unid_3, unid_n};
Instâncias de tempo	Representam o tempo exato em que ocorreu uma mudança de ação em uma linha de vida	{contidas no bloco ASSIGN – expressas a partir do comando NEXT} ASSIGN next (linha_de_vida_1) := case condição_1 : estado_1; condição_2 : estado_2; condição_n : estado_n; esac;

Fonte: Elaborado pela autora

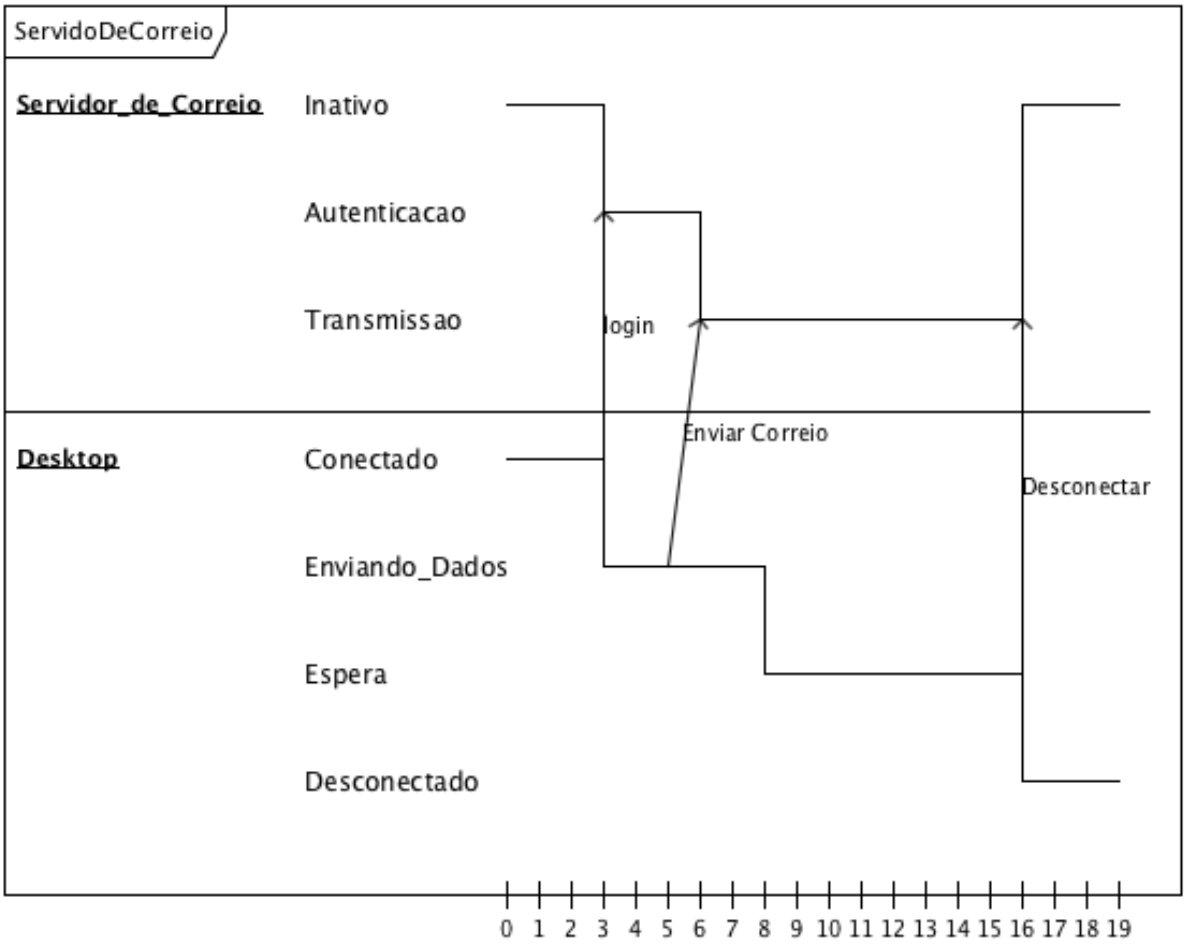
Considerando as etapas descritas anteriormente, cada item do diagrama é reportado ao XMI através de tags e tipos específicos.

A partir do arquivo XMI, gerado através da ferramenta de modelagem, os objetos são identificados e traduzidos para a linguagem do verificador.

As etapas da tradução, já descritas no Quadro 5, serão exemplificadas nas subseções a seguir e terão como referência o diagrama de temporização apresentando na Figura 11.

O diagrama representa a execução de um Servidor de Correio e, paralelamente, exibe as ações de um **desktop** enquanto o servidor é executado, é possível perceber quais serão as ações que serão executadas em cada um, e os tempos necessários para cada execução.

Figura 11 – Diagrama de temporização de um Servidor de Correio



Fonte: (PILONE; PITMAN, 2006)

3.1.1 Tradução das linhas de vida

As linhas de vida representam os estados máximos em que o diagrama se subdividirá. Em um arquivo XMI elas são representadas através de tags do tipo "*timingFrameLifeline*".

O exemplo a seguir representa o registro da linha de vida "Servidor de Correio" e é

apresentado da seguinte forma:

< PACKAGEDELEMENT NAME = “SERVIDOR_DE_CORREIO” ...
XMI:TYPE = “TIMINGFRAMELIFELINE” >

A partir da linha apresentada é possível encontrar os seguintes itens:

a) NAME – Corresponde ao nome que identifica a linha de vida;
b) XMI:TYPE – Corresponde ao tipo de objeto referido, neste caso é o que possibilita à ferramenta a identificação da linha de vida.

Conforme descrito no Quadro 5, as linhas de vida definem uma variável no modelo SMV (Figura 12).

Figura 12 – Linhas de vida do Servidor de Correio representadas no arquivo SMV

```
1    MODULE main
2    VAR
3        Servidor_de_Correio : {...};
4        Desktop : {...};
```

Fonte: Elaborado pela autora

3.1.2 Tradução dos estados condicionais

Os estados condicionais representam as mudanças de transações ocorridas em cada variável gerada a partir de uma linha de vida. Em um arquivo XMI elas são representadas através de tags do tipo “*stateCondition*”.

O exemplo a seguir representa o estado condicional “Inativo” da linha de vida “Servidor de Correio” e é apresentado da seguinte forma:

< PACKAGEDELEMENT NAME = “INATIVO” XMI:ID = “VZLXWPKGAQACBAUD”
XMI:TYPE = “STATECONDITION” >

A partir da linha apresentada é possível encontrar os seguintes itens:

a) NAME – Corresponde ao nome que identifica o estado condicional;
b) ID – Corresponde a um identificador gerado pela ferramenta de modelagem. Sua necessidade será melhor compreendida no objeto instância de tempo;
c) XMI:TYPE – Corresponde ao tipo de objeto referido, neste caso é o que possibilita à ferramenta a identificação do estado condicional.

Conforme descrito no Quadro 5, os estados condicionais definem as possíveis ações no modelo SMV (Figura 13).

Figura 13 – Estados condicionais do Servidor de Correio representadas no arquivo SMV

```
1    MODULE main
2    VAR
3        Servidor_de_Correio : {Inativo, Autenticacao, Transmissao};
4        Desktop : {Conectado, Espera, Enviando Dados, Desconectado};
```

Fonte: Elaborado pela autora

3.1.3 Tradução das unidades de tempo

As unidades de tempo definem o momento exato da ocorrência, ou troca, de uma ação. São descritas para todas as linhas de vida existentes. Em um arquivo XMI elas são representadas através do tipo "*timeUnit*".

O exemplo a seguir representa a unidade de tempo "0" e é apresentado da seguinte forma:

< PACKAGEDELEMENT NAME = "0" XMI:ID = "KVYLWPKGAQACBAP_"
XMI:TYPE = "TIMEUNIT" >

A partir da linha apresentada é possível encontrar os seguintes itens:

a) NAME – Corresponde ao nome que identifica a unidade de tempo;
b) ID – Corresponde a um identificador gerado pela ferramenta de modelagem. Sua necessidade será melhor compreendida no objeto instância de tempo;
c) XMI:TYPE – Corresponde ao tipo de objeto referido, neste caso é o que possibilita à ferramenta a identificação da unidade de tempo.

É importante ressaltar que as unidades de tempo corresponderão aos valores possíveis de alteração de uma variável denominada "temporizador" (Figura 14).

Figura 14 – Unidades de tempo do Servidor de Correio representadas no arquivo SMV

```
1    MODULE main
2    VAR
3        Servidor_de_Correio : {Inativo, Autenticacao, Transmissao};
4        Desktop : {Conectado, Espera, Enviando_Dados, Desconectado};
5        temporizador : {1, 2, 3, 4, 5, 6, 7, 8, 9, 10,11, 12, 13, 14, 15, 16, 17, 18, 19};
```

Fonte: Elaborado pela autora

3.1.4 Tradução das instâncias de tempo

As instâncias de tempo determinam qual será o tempo exato da mudança de ação de uma linha de vida. Armazenam os momentos iniciais e finais de cada evento. Em um arquivo XMI

elas são representadas através do tipo "*timeInstance*".

O que permite a compreensão das mudanças de estado em tempos específicos é o item descrito como *stateCondition* (estado condicional) e *timeUnit* (unidade de tempo). Cada um determina o id (identificação) dos objetos mencionados, formando uma junção destes.

O exemplo a seguir refere-se a uma instância de tempo da linha de vida "Servidor de Correio", em que a unidade de tempo é "0" (id = "KVYLWPKGAQACBAP_") e o estado condicional é "Inativo" (id = "VZLXwpKGAqACBAUD"), apresentado da seguinte forma:

< PACKAGEDELEMENT ... STATECONDITION = "VZLXWPKGAQACBAUD"
TIMEUNIT = "KVYLWPKGAQACBAP_" ... XMI:TYPE = "TIMEINSTANCE" >

Esta linha indica que no tempo 0 (zero) o Servidor de Correio estará na ação Inativo. Os seguintes itens são identificados:

a) STATECONDITION – Corresponde a identificação do estado condicional;
b) TIMEUNIT – Corresponde a identificação da unidade de tempo;
c) XMI:TYPE – Corresponde ao tipo de objeto referido, neste caso é o que possibilita à ferramenta a identificação da instância de tempo.

As instâncias de tempo são representadas através do comando NEXT, existente no bloco ASSIGN (Figura 15).

Figura 15 – Instâncias de tempo do Servidor de Correio no arquivo SMV

```
1    next(Servidor_de_correio) := case
2            temporizador >= 0 & temporizador <= 2 : Inativo;
3            temporizador >= 3 & temporizador <= 5 : Autenticacao;
4            temporizador >= 6 & temporizador <= 15 : Transmissao;
5            temporizador >= 16 & temporizador <= 20 : Inativo;
6    esac;
```

Fonte: Elaborado pela autora

3.1.5 Tradução completa

O arquivo SMV será gerado no mesmo diretório indicado para a leitura do XMI, e como padrão, o nome será definido a partir da junção do nome do diagrama de temporização e a identificação fixa "*_TimingDiagram*".

Após todas as traduções, o código final, gerado para o XMI da Figura 11 é apresentado na Figura 16.

Figura 16 – Tradução completa do Servidor de Correio para o SMV

```
1       MODULE main
2       VAR
3               Servidor de Correio : {Inativo, Autenticacao, Transmissao};
4               Desktop : {Conectado, Enviando Dados, Espera, Desconectado};
5               temporizador : { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18,
19 };
6
7       ASSIGN
8               init(Servidor de Correio) := Inativo;
9               init(Desktop) := Conectado;
10              init(temporizador) := 0;
11
12              next(temporizador) := case
13                      temporizador = 0 : 1;
14                      temporizador = 1 : 2;
15                      temporizador = 2 : 3;
16                      temporizador = 3 : 4;
17                      temporizador = 4 : 5;
18                      temporizador = 5 : 6;
19                      temporizador = 6 : 7;
20                      temporizador = 7 : 8;
21                      temporizador = 8 : 9;
22                      temporizador = 9 : 10;
23                      temporizador = 10 : 11;
24                      temporizador = 11 : 12;
25                      temporizador = 12 : 13;
26                      temporizador = 13 : 14;
27                      temporizador = 14 : 15;
28                      temporizador = 15 : 16;
29                      temporizador = 16 : 17;
30                      temporizador = 17 : 18;
31                      temporizador = 18 : 19;
32                      TRUE : 0;
33              esac;
34
35              next(Servidor de Correio) := case
36                      temporizador >= 0 & temporizador <= 2 : Inativo;
37                      temporizador >= 3 & temporizador <= 5 : Autenticacao;
38                      temporizador >= 6 & temporizador <= 15 : Transmissao;
39                      temporizador >= 16 & temporizador <= 19 : Inativo;
40              esac;
41
42              next(Desktop) := case
43                      temporizador >= 0 & temporizador <= 2 : Conectado;
44                      temporizador >= 3 & temporizador <= 7 : Enviando Dados;
45                      temporizador >= 8 & temporizador <= 15 : Espera;
46                      temporizador >= 16 & temporizador <= 19 : Desconectado;
47              esac;
```

Fonte: Elaborado pela autora

3.2 Verificações

As propriedades serão descritas no intuito de permitir que diversas verificações sejam efetuadas no diagrama de temporização, permitindo mais compreensão, abrangência e análise dos itens.

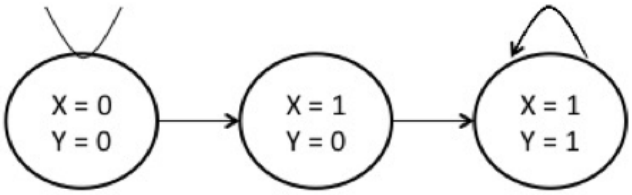
A seguir serão apresentadas algumas propriedades possíveis para a validação de qualquer

código em um verificador de modelos.

Para melhor compreensão das verificações é importante saber que duas variáveis serão utilizadas, X e Y, que representarão as atividades, e em ambas, dois valores possíveis serão declarados 0 (zero), indicando que a atividade não foi executada, e 1 (um), indicando que a atividade foi executada.

3.2.1 Executar uma atividade implica em executar outra

Figura 17 – Executar uma atividade implica em executar outra



Fonte: (BARROS; SONG, 2012)

A Figura 17 ilustra a verificação da execução de uma atividade após a outra, sendo obrigatória que a atividade X, compreendida aqui como primeira atividade, seja executada, para que então, a execução da atividade Y aconteça, e este status permanecerá até o final do fluxo. O comando CTL$*$ criado para atender a esta situação é:

$$\text{SPEC AG(ATIVIDADEX} \rightarrow \text{AF(ATIVIDADEY));}$$

Sua compreensão pode ser descrita como *"sempre que a atividade X for executada, implica em que, no futuro, em todos os caminhos, a atividade Y também seja executada"*.

Como exemplo, para a Figura 11, podemos considerar a seguinte verificação *"É sempre verdade que quando o ServidorDeCorreio estiver na ação Autenticacao o temporizador será maior ou igual a 5"*, o que gerará o comando

$$\text{SPEC AG ((SERVIDORDECORREIO = AUTENTICACAO)} \rightarrow \text{AF (TEMPORIZADOR > 5));}$$

3.2.2 Executar uma atividade implica em não executar outra

Figura 18 – Executar uma atividade implica em não executar outra

Fonte: (BARROS; SONG, 2012)

A Figura 18 ilustra a verificação da não execução de uma atividade após a outra, ou seja, após a execução da atividade X a atividade Y não mais será executada, em nenhum outro caminho. O comando CTL∗ criado para atender a esta situação é:

SPEC AG(ATIVIDADEX → AG(!ATIVIDADEY));

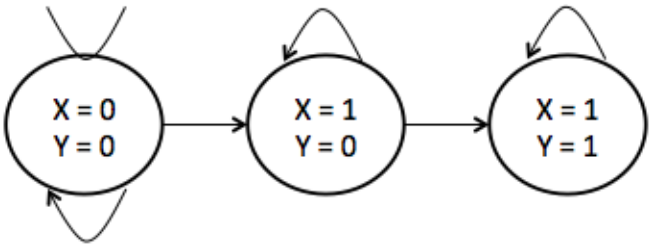
E sua compreensão pode ser descrita como *"sempre que a atividade X for executada, implica em que, em todos os caminhos a atividade Y não mais seja executada"*.

Como exemplo, para a Figura 11, podemos considerar a seguinte verificação *"É sempre verdade que quando o ServidorDeCorreio estiver na ação Transmissao o temporizador não será maior que 15"*, o que gerará o comando

SPEC AG ((SERVIDORDECORREIO = TRANSMISSAO) → AG (!TEMPORIZADOR > 15));

3.2.3 Executar uma atividade depende da execução de outra

Figura 19 – Executar uma atividade depende da execução de outra



Fonte: (BARROS; SONG, 2012)

A Figura 19 ilustra que uma atividade Y só será executada a partir da execução da atividade X, neste caso, a verificação primeiramente descarta as seguintes hipóteses:

a) de que exista algum caminho em que a atividade X seja executada e a atividade Y nunca;
b) as atividades X e Y sejam executadas ao mesmo tempo.

Descartadas as hipóteses anteriores a verificação consegue garantir o que está sendo avaliado. O comando CTL∗ criado para atender a esta situação é:

SPEC !EF(ATIVIDADEX & ATIVIDADEY) & !EF(!ATIVIDADEX & !ATIVIDADEY & EX(ATIVIDADEX & ATIVIDADEY));

E sua compreensão pode ser descrita como *"existe algum caminho em que no futuro a atividade Y só será executada a partir da execução da atividade X"*.

Como exemplo, para a Figura 11, podemos considerar a seguinte verificação *"É sempre verdade que o Desktop estará desconectado apenas quando o ServidorDeCorreio estiver inativo"*, o que gerará o comando:

SPEC !EF(SERVIDODECORREIO = INATIVO & DESKTOP = DESCONECTADO) & !EF(!SERVIDORDECORREIO = INATIVO & !DESKTOP = DESCONECTADO & EX(SERVIDORDECORREIO = INATIVO & DESKTOP = DESCONECTADO));

3.2.4 Existe a possibilidade de uma atividade não ser executada

Figura 20 – Existe a possibilidade de uma atividade não ser executada

Fonte: (BARROS; SONG, 2012)

A Figura 20 ilustra a verificação da possibilidade de uma atividade não ser executada no fluxo determinado pelo diagrama. O comando CTL∗ criado para atender a esta situação é:

SPEC !EG(!ATIVIDADEX);

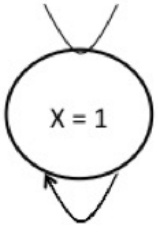
E sua compreensão pode ser descrita como *"existe algum caminho onde a atividade X não será executada"*.

Como exemplo, para a Figura 11, podemos considerar a seguinte verificação *"Existe algum caminho onde o ServidorDeCorreio não ficará inativo"*, o que gerará o comando

SPEC !EG ((!SERVIDORDECORREIO = INATIVO);

3.2.5 Uma determinada atividade sempre será executada

Figura 21 – Uma determinada atividade sempre será executada



Fonte: (BARROS; SONG, 2012)

A Figura 21 ilustra que uma atividade é executada e continua neste estado até o final do fluxo. O comando CTL∗ criado para atender a esta situação é:

SPEC AG(ATIVIDADEX);

E sua compreensão pode ser descrita como *"é sempre verdade que a atividade será a mesma do início do final do fluxo"*.

Como exemplo, para a Figura 11, podemos considerar a seguinte verificação *"É sempre verdade que o Desktop ficará desconectado do início ao final do fluxo"*, o que gerará o comando

SPEC AG (DESKTOP = DESCONECTADO);

3.2.6 É sempre possível atingir o final do fluxo

A verificação considera se sempre será possível alcançar o final da execução, para isto dois itens não poderão ocorrer:

a) inexistência do final do fluxo, ou seja, sempre deve existir possibilidade de fim;
b) se existe algum caminho que nunca alcançará o final do fluxo;

O comando CTL∗ criado para atender a esta situação é:

SPEC AG(ATIVIDADEX → AF(ATIVIDADEY));

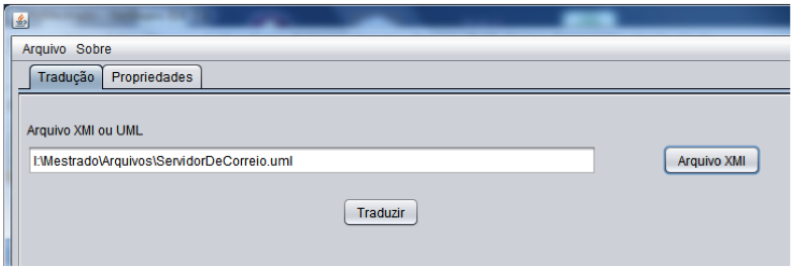
E sua compreensão pode ser descrita como "é sempre verdade que a execução será finalizada" Como exemplo, para a Figura 11, podemos considerar a seguinte verificação *"É sempre verdade que ao executar uma atividade o ServidorDeCorreio volte a ficar inativo"*, o que gerará o comando

SPEC AG(!SERVIDORDECORREIO = INATIVO → AF(SERVIDORDECORREIO = INATIVO));

3.3 Detalhes sobre a ferramenta de tradução e descrição das propriedades

Com o intuito de validar a metodologia proposta para a tradução do diagrama de temporização, a partir deste trabalho a autora gerou uma ferramenta que automatiza o processo, permitindo a inclusão de um arquivo XMI ou UML, efetuando a tradução para um arquivo SMV, e logo após permite que diversas propriedades sejam definidas.

Figura 22 – Tela inicial para entrada de XMI e geração de SMV

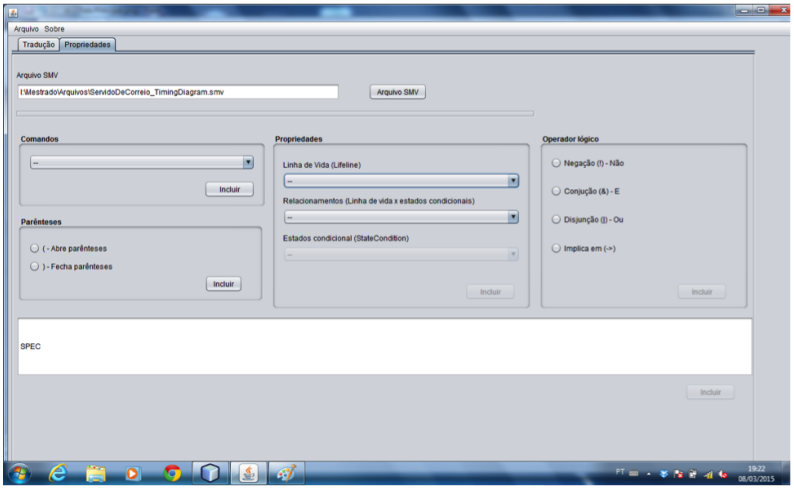


Fonte: Elaborado pela autora

A Figura 22 exibe a tela inicial da ferramenta, onde o engenheiro de *software* incluirá o arquivo XMI para que a tradução do diagrama ocorra. É importante reforçar que este arquivo será gerado em qualquer ferramenta de modelagem existente no mercado.

A partir da tradução será possível utilizar o arquivo SMV gerado e definir novas propriedades. O engenheiro de *software* as definirá e, através da ferramenta, escreverá o comando CTL$*$ pertinente a elas, conforme exibido na Figura 23.

Figura 23 – Tela de inclusão de propriedades



Fonte: Elaborado pela autora

Este capítulo apresentou a metodologia utilizada para a tradução de um diagrama de temporização para o verificador de modelos NuSMV. Exemplifica também algumas propriedades que poderão ser para a verificação do modelo. O próximo capítulo apresentará dois estudos de caso que aplicarão a metodologia descrita.

4 ESTUDO DE CASO

Uma das técnicas de pesquisa aplicada é a descritiva, que objetiva correlacionar as diversas situações e relações existentes entre os elementos. Ela pode assumir diversas formas, dentre elas o estudo de caso.

Entende-se como estudo de caso uma pesquisa detalhada sobre um ou mais objetos, permitindo assim avaliar e descrever sobre o objeto de estudo.

Neste trabalho será utilizado um estudo de caso ilustrativo, que permitirá a compreensão da metodologia já explicada em dois exemplos práticos aplicados.

4.1 Casos de análise

Foram analisados dois casos para estudo, conforme identificações abaixo:

a) Projeto de automação, denominado Profibus-PA, pertencente a uma associação não comercial;
b) Projeção de um controlador semafórico para atender às exigências de trânsito em um determinado local. Será desenhado a partir das considerações descritas pelo Departamento Nacional de Trânsito (Denatran).

Suas descrições e validações serão detalhadas nas seções a seguir.

4.2 Profibus-PA

O Profibus corresponde a um padrão de rede, de campo aberto e independente de fornecedores. Sua interface permite grande aplicação em processos, manufatura e automação. O Profibus-PA define os parâmetros e blocos de função para dispositivos de automação de processos, como, por exemplo, transmissores, posicionadores e válvulas (ASSOCIAÇÃO PROFIBUS, 2015).

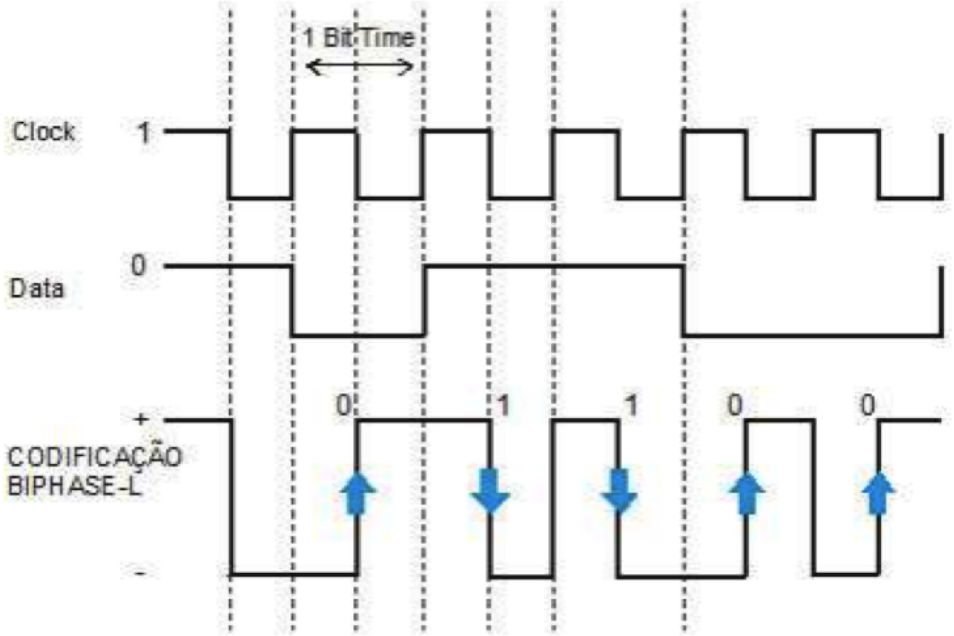
A Figura 24 ilustra um diagrama de temporização referente a passagem de sinal do Profibus-PA em modo tensão, onde fica evidenciado a passagem ou não de corrente. O Quadro 6 indica em quais situações a Codificação Bifase terá seu resultado positivo ou negativo.

Quadro 6 – Resultado da Condificação Bifase (positivo ou negativo)

Clock **(pulso)**	***Data*** **(informação)**	**Codificação Bifase**
1	0	+ (positivo - 1)
1	1	- (negativo - 0)
0	0	- (negativo - 0)
0	1	+ (positivo - 1)

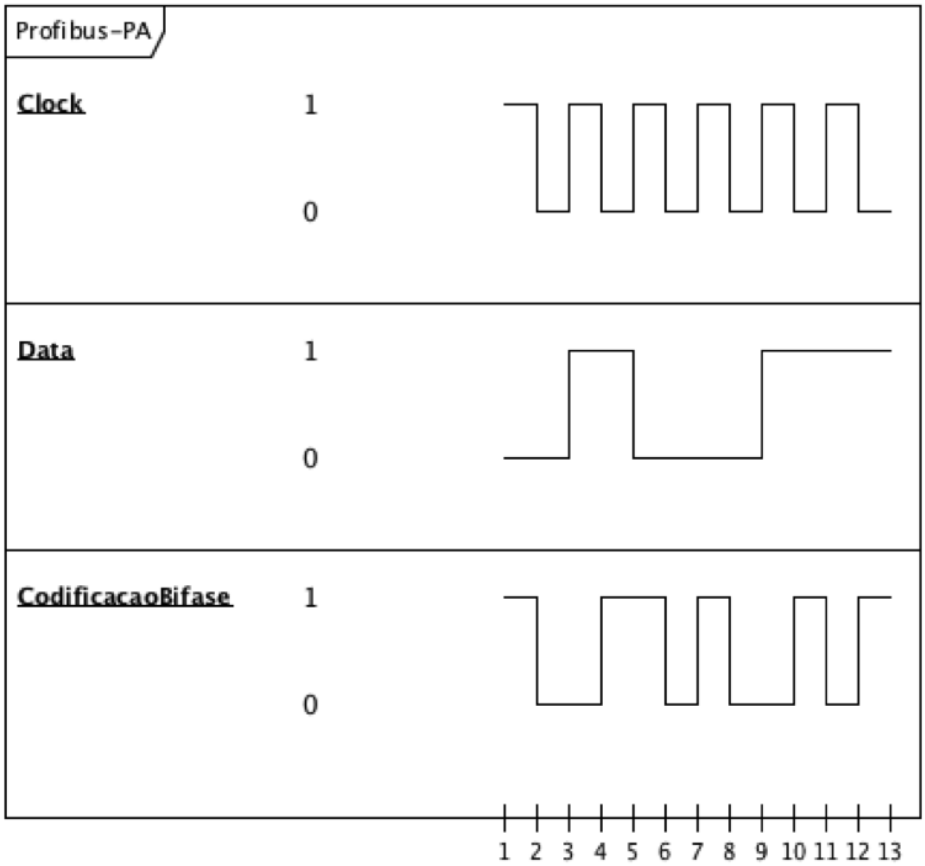
Fonte: Elaborado pela autora

Figura 24 – Exemplo de sinal PROFIBUS-PA em modo tensão



Fonte: (ASSOCIAÇÃO PROFIBUS, 2015)

Figura 25 – Diagrama de temporização do Profibus-PA



Fonte: Elaborado pela autora.

4.2.1 Validações

A Figura 25 indica o diagrama de temporização da Figura 24, reescrito para atender a metodologia indicada neste trabalho, foram inseridos tempos expressos de forma numérica, entre outras exigências.

Considerando o diagrama da Figura 25, as subseções a seguir indicam as verificações que poderão ser impostas, via interface, para a situação explicada nos diagramas acima. Para compreensão e execução através do verificador NuSMV as respostas possíveis da linha de vida CodificacaoBifase será 1 (um) para positivo ou 0 (zero) para negativo.

A seguir serão apresentadas algumas verificações que podem ser aplicadas ao padrão de rede Profibus-PA.

4.2.1.1 Um pulso *Clock* implica em, no futuro, o Data não enviar informação

A descrição considera se existe alguma situação onde no momento em que o *Clock* transmitir o bit 1, no próximo evento não ocorrerá a transmissão do bit 1 pelo data. O comando CTL$*$ para o questionamento indicado é:

$$\text{SPEC EG (CLOCK = 1)} \rightarrow \text{EF !(DATA = 1)}$$

Para esta execução, a resposta do verificador será:

$$\text{SPECIFICATION (EG CLOCK = 1} \rightarrow \text{EF ! (DATA = 1)) IS TRUE}$$

Informando que todos os caminhos confirmam o que diz a propriedade, e não há contradições com relação ao que foi afirmado.

4.2.1.2 Codificação Bifase positiva implica em o *Clock* enviar um pulso

A descrição considera que no momento em que a *CodificacaoBifase* estiver positiva (1), implicará em que o *Clock* transmita o bit 1. O comando CTL$*$ para o questionamento indicado é:

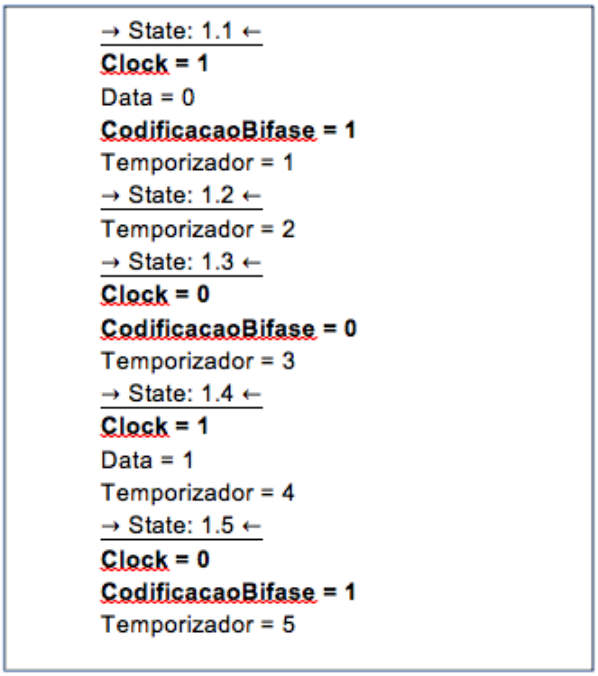
$$\text{SPEC AG (CODIFICACAOBIFASE = 1)} \rightarrow \text{(CLOCK = 1)}$$

Para esta execução, a resposta do verificador será:

$$\text{SPECIFICATION (AG CODIFICACAOBIFASE = 1} \rightarrow \text{CLOCK = 1) IS FALSE}$$

Informando que existe um caminho que contradiz o que foi solicitado, e para este caso, ele apresenta o seguinte contraexemplo (ver Figura 26):

Figura 26 – Contraexemplo da propriedade "Codificação Bifase positiva implica em o *Clock* enviar um pulso"



Fonte: Elaborado pela autora

4.2.1.3 Existe algum caminho em que nunca ocorre um pulso de *Clock*

A descrição verifica se existe algum caminho em que o *Clock* nunca enviará o bit 0. O comando CTL∗ para o questionamento indicado é:

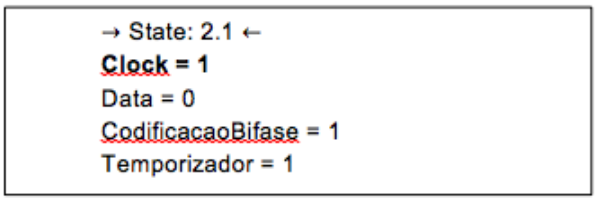
SPEC EG ! (CLOCK = 1)

Para esta execução, a resposta do verificador será:

SPECIFICATION (EG ! (CLOCK = 1) IS FALSE

Informando que existe um caminho que contradiz o que foi solicitado, e para este caso, ele apresenta o seguinte contraexemplo (ver Figura 27):

Figura 27 – Contraexemplo da propriedade "Existe algum caminho em que nunca ocorre um pulso de *Clock*"



Fonte: Elaborado pela autora

4.2.1.4 Data envia informação somente se o temporizador for maior ou igual a 10

A descrição considera que, em todo o processamento, sempre que o *Data* transmitir o bit 1, o temporizador será maior ou igual o tempo 10. O comando CTL∗ para o questionamento indicado é:

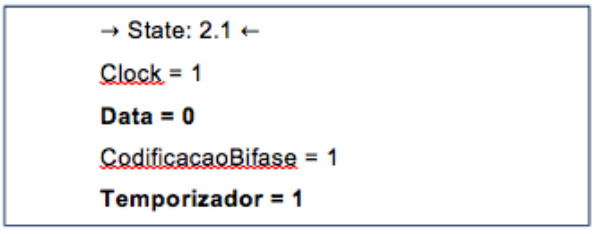
SPEC AG (DATA = 1) & (TEMPORIZADOR >= 10)

Para esta execução, a resposta do verificador será:

SPECIFICATION (AG DATA = 1 & TEMPORIZADOR >= 10) IS FALSE

Informando que existe um caminho que contradiz o que foi solicitado, e para este caso, ele apresenta o seguinte contraexemplo (ver Figura 28):

Figura 28 – Contraexemplo da propriedade "Data envia informação somente se o temporizador for maior ou igual a 10"



Fonte: Elaborado pela autora

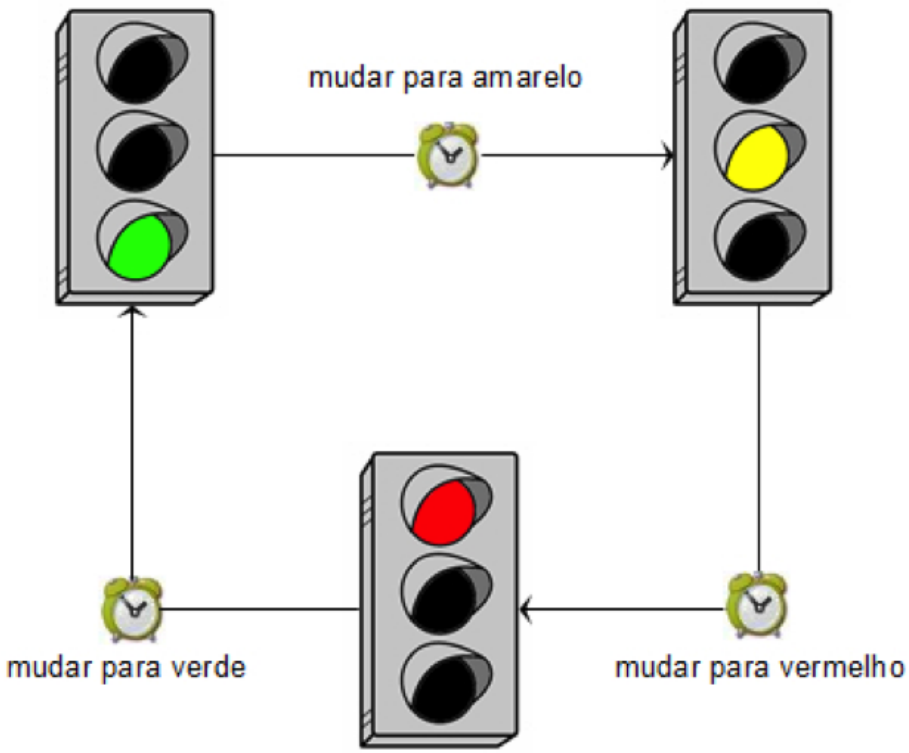
4.3 Controladores semafóricos

Entende-se como sinalização semafórica o objeto que tem por dever efetuar o controle de trânsito em uma interseção ou seção de via, utilizando indicações luminosas e alternando o direito de passagem de diversos fluxos (DEPARTAMENTO NACIONAL DE TRÂNSITO, 2015).

O semáforo que será analisado neste estudo é do tipo veicular, não direcionado a ciclistas. Que utiliza três indicações luminosas (verde, amarela e vermelha). A Figura 29 exemplifica a atuação deste tipo de dispositivo.

É importante ressaltar que os dados foram retirados de um documento público, disponibilizado pelo Denatran. O diagrama de temporização foi elaborado pela autora para atender a metodologia e facilitar a compreensão dos dados envolvidos.

Figura 29 – Modelo de um semáforo veicular



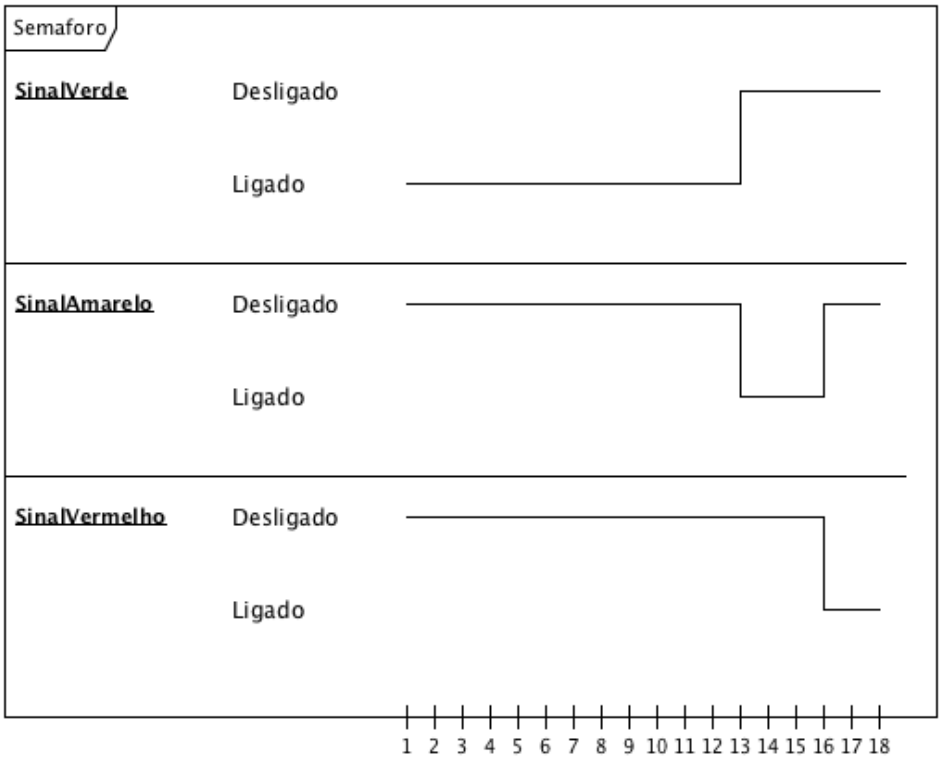
Fonte: (DEPARTAMENTO NACIONAL DE TRÂNSITO, 2015)

A Figura 30 define um diagrama de temporização direcionado para um semáforo veicular, como o da Figura 29.

Os tempos determinados para a criação do diagrama foi de 3 segundos para o vermelho, 3 segundos para o amarelo e 12 segundos para o verde. Esses tempos foram considerados a partir do documento disponibilizado pelo Denatran em um dos grupos determinados pelo órgão. Os cálculos que definem o tempo de um dispositivo não foram considerados relevantes para esta pesquisa, mas se o leitor assim desejar pode acessar diretamente o documento citado.

A execução será feita da seguinte forma, o sinal inicia na ação "Sinal Verde", após 12 tempos ele será alterado para a ação "Sinal Vermelho", após 3 tempos, ele será alterado para a ação "Sinal Amarelo", e assim mantém sua execução.

Figura 30 – Diagrama de temporização de um semáforo de 18s



Fonte: Elaborado pela autora

Considerando o diagrama da Figura 30, as subseções a seguir indicam as verificações que poderão ser impostas, via interface, para um semáforo.

4.3.1 Validações

A seguir serão apresentadas algumas verificações que podem ser aplicadas ao controlador semafórico, com os detalhes determinados pelo Denatran.

4.3.1.1 Existe algum caminho, em que, no futuro, o sinal amarelo e o sinal verde estejam desligados

A descrição considera se existe alguma situação em que, no futuro, os sinais amarelo e verde estarão desligados ao mesmo tempo. O comando CTL∗ para o questionamento indicado é:

SPEC EF (SINALAMARELO = DESLIGADO) & (SINALVERDE = DESLIGADO)

Para esta execução, a resposta do verificador será:

SPECIFICATION (EF SINALAMARELO = DESLIGADO & SINALVERDE = DESLIGADO) IS TRUE

Informando que existe um caminho que atenda a propriedade definida, e não há contradições com relação ao que foi afirmado.

4.3.1.2 Sinal vermelho ligado e temporizador menor que 17

A descrição considera se existe algum caminho, onde o sinal vermelho estará ligado e o temporizador será menor que 17. O comando CTL∗ para o questionamento indicado é:

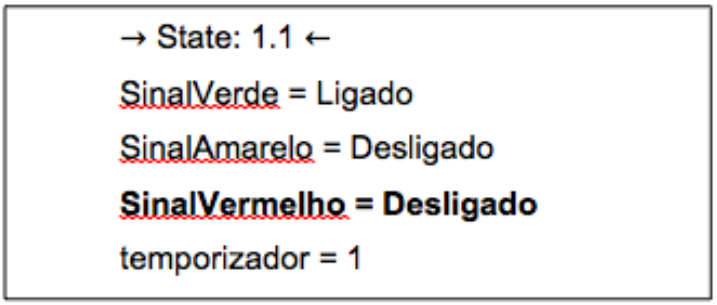
SPEC EX (SINALVERMELHO = LIGADO & TEMPORIZADOR < 17)

Para esta execução, a resposta do verificador será:

SPECIFICATION EX (SINALVERMELHO = LIGADO & TEMPORIZADOR < 17) IS FALSE

Informando que existe um caminho que contradiz o que foi solicitado, e para este caso, ele apresenta o seguinte contraexemplo (ver Figura 31):

Figura 31 – Contraexemplo da propriedade "Sinal vermelho ligado e temporizador menor que 17"



Fonte: Elaborado pela autora

4.3.1.3 Sinal amarelo sempre desligado

A descrição considera se é possível que o sinal amarelo nunca fique ligado. O comando CTL∗ para o questionamento indicado é:

SPEC AG (SINALAMARELO = DESLIGADO)

Para esta execução, a resposta do verificador será:

SPECIFICATION AG SINALAMARELO = DESLIGADO IS FALSE

Informando que existe um caminho que contradiz o que foi solicitado, e para este caso, ele apresenta o seguinte contraexemplo (ver Figura 32):

Figura 32 – Contraexemplo da propriedade "Sinal amarelo sempre desligado"

Fonte: Elaborado pela autora

4.3.1.4 Sinal vermelho desligado implica em sinal verde ligado

A descrição considera se existe algum caminho, em que, no futuro, quando o sinal vermelho estiver desligado, isto implica em que, no próximo evento, o sinal verde estará ligado. O comando CTL∗ para o questionamento indicado é:

SPEC (EF SINALVERMELHO = DESLIGADO) -> (EX SINALVERDE = LIGADO)

Para esta execução, a resposta do verificador será:

SPECIFICATION (EF SINALVERMELHO = DESLIGADO -> EX SINALVERDE = LIGADO) IS TRUE

Informando que existe um caminho que atenda a propriedade definida, e não há contradições com relação ao que foi afirmado.

Este capítulo apresentou dois estudos de caso e algumas possíveis verificações para os modelos descritos.

O próximo capítulo apresentará a conclusão desta pesquisa e os trabalhos futuros, para que haja continuidade e melhoria neste.

Os código XMI e SMV, gerados para as Figuras 25 e 30, estão disponíveis nos Anexos e Apêndice deste documento, respectivamente.

5 CONCLUSÕES

A verificação de modelos tem se mostrado muito eficiente na minimização de erros durante a fase de análise de dados, principalmente quando associada à construção de diagramas. Erros que diversas vezes não são encontrados no início do projeto de *software*, podem ser facilmente percebidos e corrigidos a tempo.

Este trabalho apresenta uma interface que permite que o engenheiro de *software* consiga modelar e automatizar a verificação dos diagramas de temporização da UML, fazendo assim, com que sistemas que tenham o tempo como fator crucial para definição de negócios, sejam validados e avaliados ainda em sua fase inicial.

Considerando que estes profissionais nem sempre possuem o devido conhecimento na aplicação da técnica de verificação, o ambiente permite ampliar o processo e V&V, bem como antecipar as fases de testes.

Os testes demonstrados apresentam a eficácia da ferramenta e justifica os objetivos almejados, permitindo detectar erros de modelagem, e erros na compreensão dos modelos, adequando-os para que atenda a realidade do produto. Assim, a fase de codificação, consequentemente, será mais exata, visto que os modelos retratam e validam o que foi idealizado.

Algumas limitações foram encontradas durante o desenvolvimento e testes do ambiente, dentre elas, podemos citar:

a) O problema de explosão de estados, reconhecidamente, é uma restrição já declarada ao aplicar verificação de modelos. Este trabalho não se preocupou diretamente em resolvê-la, mas reconhece a importância de modelar corretamente e evitar ciclos infinitos;
b) Ter um padrão de modelagem que atenda a ferramenta. Considerando que a UML não possui formalização, e muitos diagramas são desenhados sem a utilização de todos os itens citados neste trabalho. Para que o ambiente possa reconhecer e aplicar a verificação, é necessário que o engenheiro modele-o da forma proposta.

Ressalta-se que, mesmo encontrando trabalhos que são próximos ao que foi apresentado, a inovação ainda se apresenta pois poucos estudos são direcionados aos diagramas de temporização, que são sistemas onde o tempo é primordial para a execução das tarefas.

Outra consideração importante é que os profissionais da área de tecnologia da informação poderão aplicar a verificação de modelos de forma mais amigável, necessitando apenas de um conhecimento básico sobre o assunto.

5.1 Trabalhos futuros

Com o intuito de dar continuidade neste trabalho seguem algumas sugestões que poderão ser desenvolvidas e melhoradas no futuro:

a) Compreender, explorar e evitar o problema da explosão de estados. Identificar casos onde os ciclos infinitos gerarão erros na execução da validação. Tratá –los e retornar ao código;
b) Incluir a tradução dos diagramas desenvolvidos por Fernandes (2011) e Barros (2010), identificados na Figura 2, agrupando as validações em um único ambiente;
c) Associado ao item anterior, incluir um *plugin*, que permita aos profissionais a validação automática dos diagramas;
d) Avaliar a possibilidade de tradução dos diagramas estruturais da UML, pois, os estudos apresentam apenas os diagramas comportamentais;
e) Aplicar a metodologia em um caso real, preferencialmente durante um projeto de desenvolvimento de *software*, permitindo assim acompanhar sua validação. E ao final, avaliar e garantir os resultados;
f) Interpretar os resultados e, em casos de contraexemplos, indicar o local de possível correção a que este se refere.

REFERÊNCIAS

ABES - ASSOCIAÇÃO BRASILEIRA DAS EMPRESAS DE SOFTWARE (Comp.). **MERCADO BRASILIEIRO DE SOFTWARE**: panorama e tendências, 2014. 2014. Disponível em: <http://central.abessoftware.com.br/Content/UploadedFiles/Arquivos/Dados%202011/Publicac%CC%A7ao-mercado-abes-2014.pdf>. Acesso em: 26 set 2014.

AMLA, N. et al. Model checking synchronous timing diagrams. In: FORMAL METHODS IN COMPUTDER-AIDED DESIGN. 3, 2000. Texas: **Proceedings of FMCAD 2000**, 2000. v. 3, p. 320–335.

ASSOCIAÇÃO PROFIBUS (Org.). **DESCRIÇÃO TÉCNICA PROFIBUS 2012**. 2015. Disponível em: <http://www.profibus.org.br/files/DescricaoTecnica/PROFIBUS_DESC_TEC_2012.pdf>. Acesso em: 20 mar 2015.

BAIER, C.; KATOEN, J.-P. **PRINCIPLES OF MODEL CHECKING**. London: The MIT Press, 2008. 984 p.

BARROS, C.; SONG, M. Automatized checking of business rules for activity execution sequence in workflows. **JOURNAL OF SOFTWARE**, v. 7, n. 2, p. 374–381, fev 2012.

BARROS, C. M. **VERIFICAÇÃO AUTOMATIZADA DE REGRAS DE SEQUENCIAMENTO DE EXECUÇÃO EM WORKFLOWS**. 2010. 83 p. Dissertação (Mestrado) — Programa de Pós Graduação em Informática, Pontifícia Universidade Católica de Minas Gerais, Belo Horizonte, 2010.

BHADURI, P.; RAMESH, S. Model checking of statechart models: survey and research directions. **COMPUTING RESERARCH REPOSITORY: CORR**, cs.SE/0407, jul 2008.

BLACK, R.; VEENENDAAL, E. V.; GRAHAM, D. **FOUNDATIONS OF SOFTWARE TESTING**: Istqb certification. 3. ed. Hampshire: Cengage Learning, 2012. 256 p.

BOOCH, G.; RUMBAUGH, J.; JACOBSON, I. **UML**: guia do usuário. 2. ed. Rio de Janeiro: Elsevier, 2012. 521 p.

BRADLEY, A. R. Sat-based model checking without unrolling. In: INTERNATIONAL CONFERENCE ON VERIFICATION, MODEL CHECKING, AND ABSTRACT INTERPRETATION, 12, 2011. Austin: **Proceedings of VMCAI 2011**, 2011. p. 70–87.

BRYANT, R. Graph based algorithms for boolean function manipulation. **IEEE TRANSACTIONS ON COMPUTERS**, c35, n. 8, p. 677–691, ago 1986.

CAVADA, R. et al. **NUSMV 2.5 USER MANUAL**. 2015. Disponível em: <http://nusmv.fbk.eu/NuSMV/userman/v25/nusmv.pdf>. Acesso em: 26 fev 2015.

CHAMA, W.; ELMANSOURI, R.; CHAOUI, A. Model checking and code generation for uml diagrams using graph transformation. **INTERNATIONAL JOURNAL OF SOFTWARE ENGINEERING & APPLICATIONS**, v. 3, n. 6, p. 39–55, nov 2012.

CLARKE, E. M.; GRUMBERG, O. J.; PELED, D. A. **MODEL CHECKING**. London: The MIT Press, 1999. 314 p.

CLARKE, E. M.; LERDA, F. Model checking: Software and beyond. **JOURNAL OF UNIVERSAL COMPUTER SCIENCE**, v. 13, n. 5, p. 639–649, mai 2007.

CLAVEL, M. et al. **MAUDE MANUAL: VERSION 2.6**. 2011. Disponível em: <http://maude.cs.uiuc.edu/maude2-manual/maude-manual.pdf>. Acesso em: 01 mar 2015.

DEPARTAMENTO NACIONAL DE TRÂNSITO (Org.). **MANUAL BRASILEIRO DE SINALIZAÇÃO DE TRÂNSITO**: sinalização semafórica. 2015. Disponível em: <http://www.denatran.gov.br/download/resolucoes/resolu cao4832014_anexo.pdf>. Acesso em: 20 mar 2015.

ERICKSON, J.; SIAU, K. Unified modeling language: The teen years and growing pains. In: INTERNATIONAL CONFERENCE HUMAN COMPUTER INTERACTION, 15, 2013. Las Vegas: **Proceedings of HCI 2013**, 2013. v. 8016, p. 295–304.

FERNANDES, F. G. **UM ARCABOUÇO PARA VERIFICAÇÃO AUTOMÁTICA DE MODELOS UML**. 2011. 95 p. Dissertação (Mestrado) — Programa de Pós Graduação em Informática, Pontifícia Universidade Católica de Minas Gerais, Belo Horizonte, 2011.

FERRARI, A. et al. **FORMS/FORMAT 2010**: Formal methods for automation and safety in railway and automotive systems. In: ____. Oxford: Springer-verlag Berlin Heidelberg, 2010. cap. Model Checking Interlocking Control Tables, p. 107–115.

FOWLER, M. **UML DISTILLED**: a brief guide to the standard object modeling language. 3. ed. Boston: Addison Wesley, 2003. 208 p.

GAGNON, P.; MOKHATI, F.; BADRI, M. Applying model checking to concurrent uml models. **JOURNAL OF OBJECT TECHNOLOGY**, v. 7, p. 59–84, fev 2008.

HARDIN, R. H.; HAR'EL, Z.; KURSHAN, R. P. Cospan. In: COMPUTER AIDED VERIFICATION, 8, 1996. New Jersey: **Proceedings of CAV 1996**, 1996. v. 1102, p. 423–427.

KANER, C.; FALK, J.; NGUYEN, H. Q. **TESTING COMPUTER SOFTWARE**. 2. ed. New York: John Wiley & Sons Inc, 1999. 480 p.

KHAN, S. A. et al. Semantic web specification using z-notation. **LIFE SCIENCE JOURNAL**, v. 9, n. 4, p. 994–1000, 2012.

KWIATKOWSKA, M.; NORMAN, G.; PARKER, D. Prism 4.0: Verication of probabilistic real-time systems. In: COMPUTER AIDED VERIFICATION, 23, 2011. Snowbird: **Proceedings of CAV 2011**, 2011. p. 585–591.

LOMUSCIO, A.; PECHEUR, C.; RAIMONDI, F. Automatic verification of knowledge and time with nusmv. In: INTERNATIONAL JOINT CONFERENCE ON ARTIFICIAL INTELLIGENCE, 20, 2007. Hyderadab: **Proceedings of the IJCAI 2007**, 2007. p. 1384–1389.

MULLER, P.-A. et al. Modeling modeling modeling. **SOFTWARE AND SYSTEMS MODELING**, v. 11, n. 3, p. 347–359, jul 2012.

MYERS, G. J. **THE ART OF SOFTWARE TESTING**. 3. ed. New Jersey: John Wiley & Sons Inc, 2011. 240 p.

OBJECT MANAGEMENT GROUP (ORG.). **DIAGRAM DEFINITION**. 2014. Disponível em: <http://www.omg.org/spec/DD/>. Acesso em: 12 nov 2014.

PENDER, T. **UML, A BÍBLIA**. Rio de Janeiro: Elsevier, 2004. 711 p.

PILONE, D.; PITMAN, N. **UML 2 RÁPIDO E PRÁTICO**. Rio de Janeiro: Alta Books, 2006. 191 p.

PINHEIRO, A. C. **SUBSÍDIOS PARA A APLICAÇÃO DE MÉTODOS DE GERAÇÃO DE CASOS DE TESTES BASEADOS EM MÁQUINAS DE ESTADOS**. 2012. 95 p. Dissertação (Mestrado) — Curso de Ciências de Computação e Matemática Computacional, Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, São Carlos, 2012.

SEGALL, I.; TZORF-BRILL, R.; FARCHI, E. Using binary decision diagrams for combinatorial test design. In: INTERNATIONAL SYMPOSIUM ON SOFTWARE TESTING AND ANALYSIS, 20, 2011. Toronto: **Proceedings of ISSTA 2011**, 2011. p. 254–264.

SHARMA, A. A two step perspective for kripke structure reduction. In: INTERNATIONAL CONFERENCE ON CURRENT TRENDS IN THEORY AND PRACTICE OF COMPUTER SCIENCE, 39, 2013. Spindlerv Mlyn: **Proceedings of SOFSEM 2013**, 2013. v. 1210, p. 04–08.

SOMMERVILLE, I. **ENGENHARIA DE SOFTWARE**. 9. ed. São Paulo: Pearson Prentice Hall, 2011. 529 p.

TAMEIRÃO, A. D. L. de O.; SONG, M. A. J. Symbolic model checking applied to timing diagrams. In: SOFTWARE ENGENEERING RESEARCH AND PRACTICE, 14, 2014. Las Vegas: **Proceedings of SERP 2014**, 2014. p. 167–171.

ZHAO, Y. et al. Quantitative analysis of system based on extended uml state diagrams and probabilistic model checking. **JOURNAL OF SOFTWARE**, v. 5, n. 7, p. 793–800, jul 2010.

APÊNDICE A - CÓDIGO SMV REFERENTE AO DIAGRAMA DO ESTUDO DE CASO - PROFIBUS-PA

Código 1 – Arquivo SMV gerado via ambiente para o diagrama de temporização do Profibus-PA

```
MODULE main
VAR
  Clock : {1, 0};
  Data : {1, 0};
  CodificacaoBifase : {1, 0};
  temporizador : { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13 };
ASSIGN
  init(Clock) := 1;
  init(Data) := 0;
  init(CodificacaoBifase) := 1;
  init(temporizador) := 1;

  next(temporizador) := case
    temporizador = 1 : 2;
    temporizador = 2 : 3;
    temporizador = 3 : 4;
    temporizador = 4 : 5;
    temporizador = 5 : 6;
    temporizador = 6 : 7;
    temporizador = 7 : 8;
    temporizador = 8 : 9;
    temporizador = 9 : 10;
    temporizador = 10 : 11;
    temporizador = 11 : 12;
    temporizador = 12 : 13;
    TRUE : 1;
  esac;

  next(Clock) := case
    temporizador = 1 : 1;
    temporizador = 2 : 0;
    temporizador = 3 : 1;
    temporizador = 4 : 0;
    temporizador = 5 : 1;
    temporizador = 6 : 0;
    temporizador = 7 : 1;
    temporizador = 8 : 0;
    temporizador = 9 : 1;
    temporizador = 10 : 0;
    temporizador = 11 : 1;
    temporizador = 12 : 0;
    temporizador = 13 : 1;
  esac;
```

```
44
45  next(Data) := case
46    temporizador >= 1 & temporizador <= 2 : 0;
47    temporizador >= 3 & temporizador <= 4 : 1;
48    temporizador >= 5 & temporizador <= 8 : 0;
49    temporizador >= 9 & temporizador <= 13 : 1;
50  esac;
51
52  next(CodificacaoBifase) := case
53    temporizador = 1 : 1;
54    temporizador >= 2 & temporizador <= 3 : 0;
55    temporizador >= 4 & temporizador <= 5 : 1;
56    temporizador = 6 : 0;
57    temporizador = 7 : 1;
58    temporizador >= 8 & temporizador <= 9 : 0;
59    temporizador = 10 : 1;
60    temporizador = 11 : 0;
61    temporizador >= 12 & temporizador <= 13 : 1;
62  esac;
63
64 SPEC AG (CodificacaoBifase = 1 -> Clock = 1)
65 SPEC EG (Clock = 1) -> EF!(Data = 1)
66 SPEC EG !(Clock = 1)
67 SPEC AG (Data = 1) & (temporizador >= 10)
```

APÊNDICE B - CÓDIGO SMV REFERENTE AO DIAGRAMA DO ESTUDO DE CASO - CONTROLADORES SEMAFÓRICOS

Código 1 – Arquivo SMV gerado via ambiente para o diagrama de temporização do Semáforo

```
1  MODULE main
2  VAR
3   SinalVerde : {Desligado, Ligado};
4   SinalAmarelo : {Desligado, Ligado};
5   SinalVermelho : {Desligado, Ligado};
6   temporizador : { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18 };
7  ASSIGN
8   init(SinalVerde) := Ligado;
9   init(SinalAmarelo) := Desligado;
10  init(SinalVermelho) := Desligado;
11  init(temporizador) := 1;
12
13  next(temporizador) := case
14    temporizador = 1 : 2;
15    temporizador = 2 : 3;
16    temporizador = 3 : 4;
17    temporizador = 4 : 5;
18    temporizador = 5 : 6;
19    temporizador = 6 : 7;
20    temporizador = 7 : 8;
21    temporizador = 8 : 9;
22    temporizador = 9 : 10;
23    temporizador = 10 : 11;
24    temporizador = 11 : 12;
25    temporizador = 12 : 13;
26    temporizador = 13 : 14;
27    temporizador = 14 : 15;
28    temporizador = 15 : 16;
29    temporizador = 16 : 17;
30    temporizador = 17 : 18;
31    TRUE : 1;
32  esac;
33
34  next(SinalVerde) := case
35    temporizador >= 1 & temporizador <= 12 : Ligado;
36    temporizador >= 13 & temporizador <= 18 : Desligado;
37  esac;
38
39  next(SinalAmarelo) := case
40    temporizador >= 1 & temporizador <= 12 : Desligado;
41    temporizador >= 13 & temporizador <= 15 : Ligado;
42    temporizador >= 16 & temporizador <= 18 : Desligado;
43  esac;
```

```
44
45   next(SinalVermelho) := case
46     temporizador >= 1 & temporizador <= 15 : Desligado;
47     temporizador >= 16 & temporizador <= 18 : Ligado;
48   esac;
49
50 SPEC (EF SinalVermelho = Desligado) -> (EX SinalVerde = Ligado)
51 SPEC EF (SinalAmarelo = Desligado & SinalVerde = Desligado)
52 SPEC AG (SinalAmarelo = Desligado)
53 SPEC (EX (SinalVerde = Ligado & temporizador < 17))
```

APÊNDICE C - TUTORIAL PARA INSERÇÃO DE PROPRIEDADES VIA AMBIENTE PROPOSTO

O texto a seguir detalha todos os passos necessários para que o engenheiro de *software* utilize o ambiente proposto e escreva as propriedades definidas.

Conforme explicado no Capítulo 3, será necessário desenhar o diagrama idealizado em uma ferramenta, já disponível no mercado, e logo após, exportar para um arquivo tipo XMI ou UML.

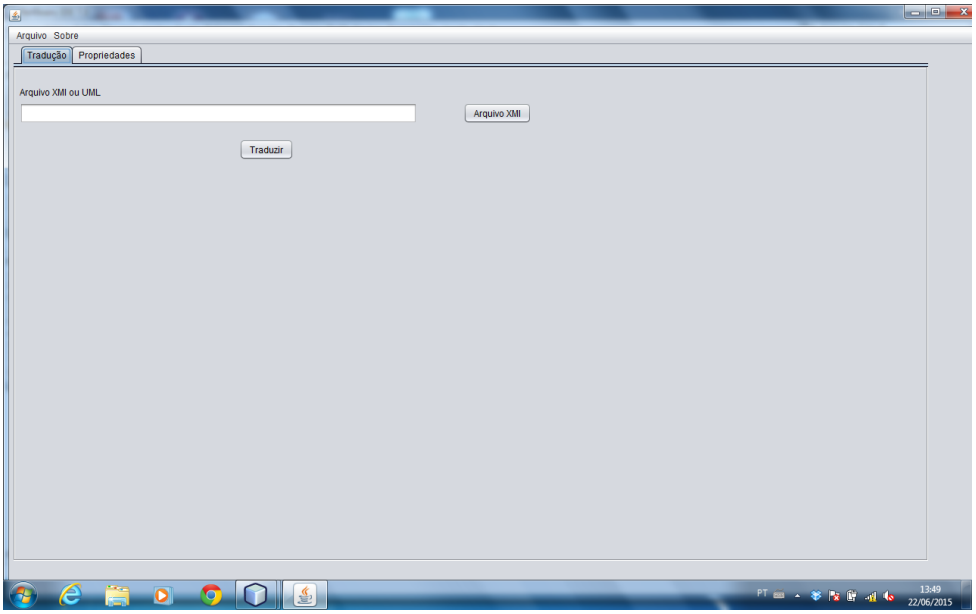
A Figura 33 exibe a tela principal do ambiente, a partir dela, o profissional deverá inserir o arquivo citado anteriormente (XMI ou UML). A busca deste arquivo será efetuada a partir do botão "Arquivo XMI".

Após escolher o arquivo, o profissional deverá clicar no botão "Traduzir", então, todos os diagramas de temporização, contidos no arquivo indicado serão traduzidos para um arquivo SMV, seguindo a metodologia já explicada.

É importante ressaltar que cada diagrama gerará um arquivo distinto, que será criado no mesmo diretório em que está o arquivo XMI indicado.

O nome deste arquivo será criado considerando a seguinte formação:<nome definido para o diagrama> + _TimingDiagram.smv. Exemplo: se o diagrama for criado com o nome ServidorCorreio, o nome do arquivo referente a ele será ServidorCorreio_TimingDiagram.smv.

Figura 33 – Tela principal do ambiente

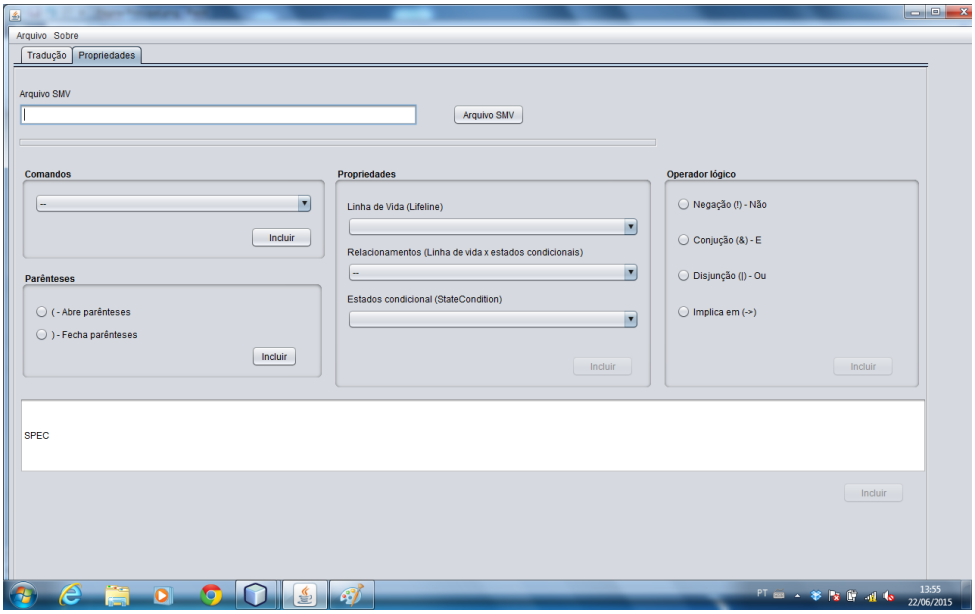


Fonte: Elaborado pela autora

A Figura 34 exibe a tela de inserção de propriedades, a partir dela, o profissional deverá escrever todas as propriedades já determinadas.

O primeiro passo para sua utilização é indicar um arquivo SMV, anteriormente gerado, através do botão "Arquivo SMV". Neste momento, todos os detalhes referentes aquele arquivo serão disponibilizados na tela, e só então o profissional poderá escrever suas propriedades.

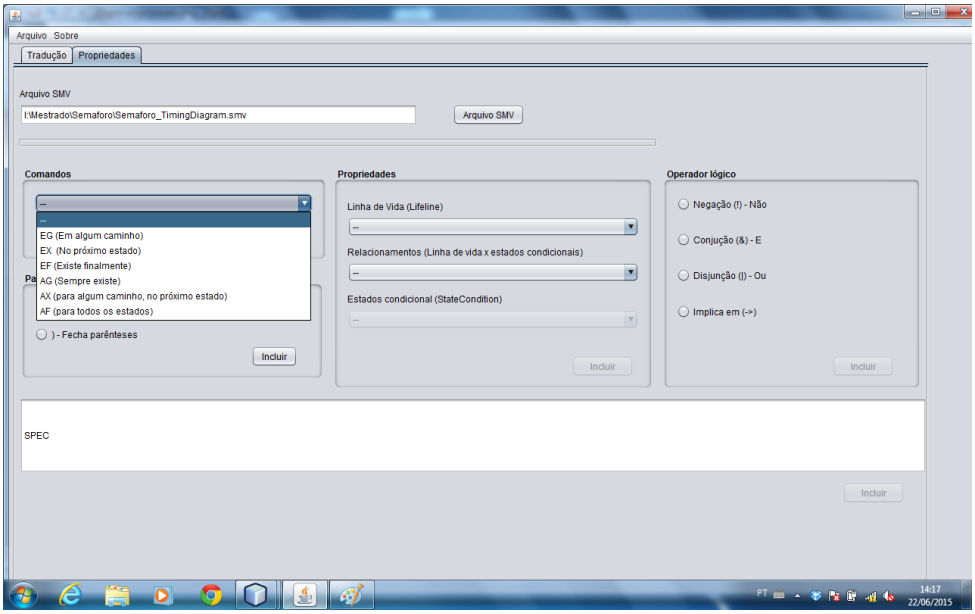
Figura 34 – Tela de Inserção de Propriedades



Fonte: Elaborado pela autora

A Figura 35 exibe como deve ser escrito o início do comando SPEC, todas as combinações de linguagem LTL e CTL.

Figura 35 – Comandos principais para a escrita da propriedade



Fonte: Elaborado pela autora

A inclusão de cada item do comando deve ser gradativa, sempre utilizando o botão "Incluir".

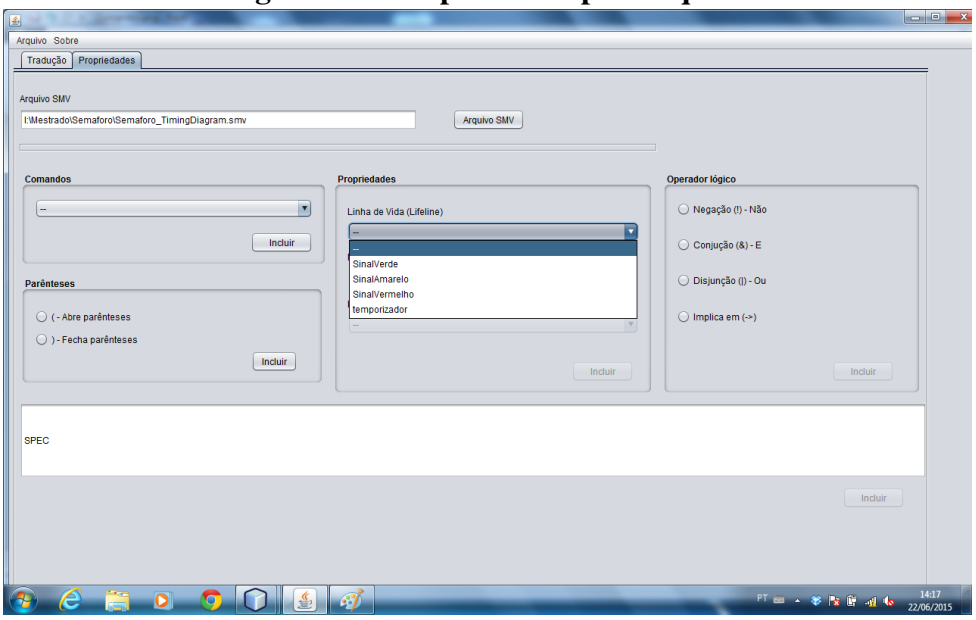
Outra informação importante é referente as propriedades disponíveis, cada arquivo carregará suas propriedades, conforme as variáveis do arquivo SMV.

A Figura 36 exibe o exemplo da lista de variáveis disponíveis para o arquivo do Controladores Semafóricos, citados na Seção 4.3 deste documento.

Os operadores lógicos, que serão permitidos no comando, seguem as possibilidades da linguagem do verificador NuSMV.

A tela não permitirá inclusão incoerente, apenas as opções possíveis ficarão habilitadas.

Figura 36 – Propriedades por Arquivo



Fonte: Elaborado pela autora

ANEXO A - CÓDIGO XMI REFERENTE AO DIAGRAMA DO ESTUDO DE CASO - PROFIBUS-PA

Código 1 – Arquivo XMI referente ao diagrama de temporização do Profibus-PA

```
<packagedElement name="Profibus-PA" xmi:id="wLYCHDKGAqAACgkN"
xmi:type="timingFrame">

   <packagedElement name="Clock" xmi:id="0XMCHDKGAqAACgkR"
    xmi:type="timingFrameLifeline">

   <packagedElement name="1" xmi:id="8M8CHDKGAqAACgkS"
      xmi:type="stateCondition">
        <xmi:Extension extender="Visual Paradigm">
        <qualityScore value="-1"/>
        </xmi:Extension>
      </packagedElement>

      <packagedElement name="0" xmi:id="H.iCHDKGAqAACgkT"
      xmi:type="stateCondition">
        <xmi:Extension extender="Visual Paradigm">
        <qualityScore value="-1"/>
        </xmi:Extension>
      </packagedElement>

      <packagedElement name="" stateCondition="8M8CHDKGAqAACgkS"
      timeUnit="VvpCHDKGAqAACgkc" xmi:id="tvpCHDKGAqAACgkd"
      xmi:type="timeInstance">
        <xmi:Extension extender="Visual Paradigm">
        <qualityScore value="-1"/>
        </xmi:Extension>
      </packagedElement>

      <packagedElement name="" stateCondition="H.iCHDKGAqAACgkT"
      timeUnit="uBZCHDKGAqAACgkg" xmi:id="BBZCHDKGAqAACgkh"
      xmi:type="timeInstance">
        <xmi:Extension extender="Visual Paradigm">
        <qualityScore value="-1"/>
        </xmi:Extension>
      </packagedElement>

      <packagedElement name="" stateCondition="8M8CHDKGAqAACgkS"
      timeUnit="7XZCHDKGAqAACgkk" xmi:id="nXZCHDKGAqAACgkl"
      xmi:type="timeInstance">
        <xmi:Extension extender="Visual Paradigm">
        <qualityScore value="-1"/>
        </xmi:Extension>
      </packagedElement>

```

```
45        <packagedElement name="" stateCondition="H.iCHDKGAqAACgkT"
46        timeUnit="_C5CHDKGAqAACgko" xmi:id="gi5CHDKGAqAACgkp"
47        xmi:type="timeInstance">
48          <xmi:Extension extender="Visual Paradigm">
49          <qualityScore value="-1"/>
50          </xmi:Extension>
51        </packagedElement>
52
53        <packagedElement name="" stateCondition="8M8CHDKGAqAACgkS"
54        timeUnit="yj5CHDKGAqAACgks" xmi:id="qj5CHDKGAqAACgkt"
55        xmi:type="timeInstance">
56          <xmi:Extension extender="Visual Paradigm">
57          <qualityScore value="-1"/>
58          </xmi:Extension>
59        </packagedElement>
60
61        <packagedElement name="" stateCondition="H.iCHDKGAqAACgkT"
62        timeUnit="ABVCHDKGAqAACgkw" xmi:id="QBVCHDKGAqAACgkx"
63        xmi:type="timeInstance">
64          <xmi:Extension extender="Visual Paradigm">
65          <qualityScore value="-1"/>
66          </xmi:Extension>
67        </packagedElement>
68
69        <packagedElement name="" stateCondition="8M8CHDKGAqAACgkS"
70        timeUnit="RE1CHDKGAqAACgk0" xmi:id="JE1CHDKGAqAACgk1"
71        xmi:type="timeInstance">
72          <xmi:Extension extender="Visual Paradigm">
73          <qualityScore value="-1"/>
74          </xmi:Extension>
75        </packagedElement>
76
77        <packagedElement name="" stateCondition="H.iCHDKGAqAACgkT"
78        timeUnit="BB1CHDKGAqAACgk4" xmi:id="xB1CHDKGAqAACgk5"
79        xmi:type="timeInstance">
80          <xmi:Extension extender="Visual Paradigm">
81          <qualityScore value="-1"/>
82          </xmi:Extension>
83        </packagedElement>
84
85        <packagedElement name="" stateCondition="8M8CHDKGAqAACgkS"
86        timeUnit="1L1CHDKGAqAACgk8" xmi:id="tL1CHDKGAqAACgk9"
87        xmi:type="timeInstance">
88          <xmi:Extension extender="Visual Paradigm">
89          <qualityScore value="-1"/>
90          </xmi:Extension>
91        </packagedElement>
```

```
92
93        <packagedElement name="" stateCondition="H.iCHDKGAqAACgkT"
94        timeUnit="v4NCHDKGAqAACglA" xmi:id="_4NCHDKGAqAACglB"
95        xmi:type="timeInstance">
96          <xmi:Extension extender="Visual Paradigm">
97          <qualityScore value="-1"/>
98          </xmi:Extension>
99        </packagedElement>
100
101       <packagedElement name="" stateCondition="8M8CHDKGAqAACgkS"
102       timeUnit="UeNCHDKGAqAACglE" xmi:id="MeNCHDKGAqAACglF"
103       xmi:type="timeInstance">
104         <xmi:Extension extender="Visual Paradigm">
105         <qualityScore value="-1"/>
106         </xmi:Extension>
107       </packagedElement>
108
109       <packagedElement name="" stateCondition="H.iCHDKGAqAACgkT"
110       timeUnit="tDNCHDKGAqAACglI" xmi:id="9DNCHDKGAqAACglJ"
111       xmi:type="timeInstance">
112         <xmi:Extension extender="Visual Paradigm">
113         <qualityScore value="-1"/>
114         </xmi:Extension>
115       </packagedElement>
116
117       <packagedElement name="" stateCondition="8M8CHDKGAqAACgkS"
118       timeUnit="fwtCHDKGAqAACglM" xmi:id="_wtCHDKGAqAACglN"
119       xmi:type="timeInstance">
120         <xmi:Extension extender="Visual Paradigm">
121         <qualityScore value="-1"/>
122         </xmi:Extension>
123       </packagedElement>
124     </packagedElement>
125
126     <packagedElement name="Data" xmi:id="i8yCHDKGAqAACgkU"
127      xmi:type="timingFrameLifeline">
128
129       <packagedElement name="1" xmi:id="Gp6CHDKGAqAACgkX"
130       xmi:type="stateCondition">
131         <xmi:Extension extender="Visual Paradigm">
132         <qualityScore value="-1"/>
133         </xmi:Extension>
134       </packagedElement>
135
136       <packagedElement name="0" xmi:id="oYmCHDKGAqAACgkY"
137       xmi:type="stateCondition">
138         <xmi:Extension extender="Visual Paradigm">
```

```
139         <qualityScore value="-1"/>
140         </xmi:Extension>
141       </packagedElement>
142
143       <packagedElement name="" stateCondition="oYmCHDKGAqAACgkY"
144       timeUnit="VvpCHDKGAqAACgkc" xmi:id="DvpCHDKGAqAACgke"
145       xmi:type="timeInstance">
146         <xmi:Extension extender="Visual Paradigm">
147         <qualityScore value="-1"/>
148         </xmi:Extension>
149       </packagedElement>
150
151       <packagedElement name="" stateCondition="oYmCHDKGAqAACgkY"
152       timeUnit="uBZCHDKGAqAACgkg" xmi:id="xBZCHDKGAqAACgki"
153       xmi:type="timeInstance">
154         <xmi:Extension extender="Visual Paradigm">
155         <qualityScore value="-1"/>
156         </xmi:Extension>
157       </packagedElement>
158
159       <packagedElement name="" stateCondition="Gp6CHDKGAqAACgkX"
160       timeUnit="7XZCHDKGAqAACgkk" xmi:id="3XZCHDKGAqAACgkm"
161       xmi:type="timeInstance">
162         <xmi:Extension extender="Visual Paradigm">
163         <qualityScore value="-1"/>
164         </xmi:Extension>
165       </packagedElement>
166
167       <packagedElement name="" stateCondition="Gp6CHDKGAqAACgkX"
168       timeUnit="_C5CHDKGAqAACgko" xmi:id="wi5CHDKGAqAACgkq"
169       xmi:type="timeInstance">
170         <xmi:Extension extender="Visual Paradigm">
171         <qualityScore value="-1"/>
172         </xmi:Extension>
173       </packagedElement>
174
175       <packagedElement name="" stateCondition="oYmCHDKGAqAACgkY"
176       timeUnit="yj5CHDKGAqAACgks" xmi:id="6j5CHDKGAqAACgku"
177       xmi:type="timeInstance">
178         <xmi:Extension extender="Visual Paradigm">
179         <qualityScore value="-1"/>
180         </xmi:Extension>
181       </packagedElement>
182
183       <packagedElement name="" stateCondition="oYmCHDKGAqAACgkY"
184       timeUnit="ABVCHDKGAqAACgkw" xmi:id="oBVCHDKGAqAACgky"
185       xmi:type="timeInstance">
```

```
186         <xmi:Extension extender="Visual Paradigm">
187         <qualityScore value="-1"/>
188         </xmi:Extension>
189       </packagedElement>
190
191       <packagedElement name="" stateCondition="oYmCHDKGAqAACgkY"
192       timeUnit="RE1CHDKGAqAACgk0" xmi:id="ZE1CHDKGAqAACgk2"
193       xmi:type="timeInstance">
194         <xmi:Extension extender="Visual Paradigm">
195         <qualityScore value="-1"/>
196         </xmi:Extension>
197       </packagedElement>
198
199       <packagedElement name="" stateCondition="oYmCHDKGAqAACgkY"
200       timeUnit="BB1CHDKGAqAACgk4" xmi:id="JB1CHDKGAqAACgk6"
201       xmi:type="timeInstance">
202         <xmi:Extension extender="Visual Paradigm">
203         <qualityScore value="-1"/>
204         </xmi:Extension>
205       </packagedElement>
206
207       <packagedElement name="" stateCondition="Gp6CHDKGAqAACgkX"
208       timeUnit="1L1CHDKGAqAACgk8" xmi:id="9L1CHDKGAqAACgk."
209       xmi:type="timeInstance">
210         <xmi:Extension extender="Visual Paradigm">
211         <qualityScore value="-1"/>
212         </xmi:Extension>
213       </packagedElement>
214
215       <packagedElement name="" stateCondition="Gp6CHDKGAqAACgkX"
216       timeUnit="v4NCHDKGAqAACglA" xmi:id="gENCHDKGAqAACglC"
217       xmi:type="timeInstance">
218         <xmi:Extension extender="Visual Paradigm">
219         <qualityScore value="-1"/>
220         </xmi:Extension>
221       </packagedElement>
222
223       <packagedElement name="" stateCondition="Gp6CHDKGAqAACgkX"
224       timeUnit="UeNCHDKGAqAACglE" xmi:id="seNCHDKGAqAACglG"
225       xmi:type="timeInstance">
226         <xmi:Extension extender="Visual Paradigm">
227         <qualityScore value="-1"/>
228         </xmi:Extension>
229       </packagedElement>
230
231       <packagedElement name="" stateCondition="Gp6CHDKGAqAACgkX"
232       timeUnit="tDNCHDKGAqAACglI" xmi:id="jDNCHDKGAqAACglK"
```

```
233       xmi:type="timeInstance">
234         <xmi:Extension extender="Visual Paradigm">
235         <qualityScore value="-1"/>
236         </xmi:Extension>
237       </packagedElement>
238
239       <packagedElement name="" stateCondition="Gp6CHDKGAqAACgkX"
240       timeUnit="fwtCHDKGAqAACglM" xmi:id="gItCHDKGAqAACglO"
241       xmi:type="timeInstance">
242         <xmi:Extension extender="Visual Paradigm">
243         <qualityScore value="-1"/>
244         </xmi:Extension>
245       </packagedElement>
246     </packagedElement>
247
248     <packagedElement name="CodificacaoBifase" xmi:id="QYWCHDKGAqAACgkZ"
249      xmi:type="timingFrameLifeline">
250
251       <packagedElement name="1" xmi:id="OIBCHDKGAqAACgka" xmi:type="stateCondition">
252         <xmi:Extension extender="Visual Paradigm">
253         <qualityScore value="-1"/>
254         </xmi:Extension>
255       </packagedElement>
256
257       <packagedElement name="0" xmi:id="YvhCHDKGAqAACgkb" xmi:type="stateCondition">
258         <xmi:Extension extender="Visual Paradigm">
259         <qualityScore value="-1"/>
260         </xmi:Extension>
261       </packagedElement>
262
263       <packagedElement name="" stateCondition="OIBCHDKGAqAACgka"
264       timeUnit="VvpCHDKGAqAACgkc" xmi:id="jvpCHDKGAqAACgkf"
265       xmi:type="timeInstance">
266         <xmi:Extension extender="Visual Paradigm">
267         <qualityScore value="-1"/>
268         </xmi:Extension>
269       </packagedElement>
270
271       <packagedElement name="" stateCondition="YvhCHDKGAqAACgkb"
272       timeUnit="uBZCHDKGAqAACgkg" xmi:id="pBZCHDKGAqAACgkj"
273       xmi:type="timeInstance">
274         <xmi:Extension extender="Visual Paradigm">
275         <qualityScore value="-1"/>
276         </xmi:Extension>
277       </packagedElement>
278
279       <packagedElement name="" stateCondition="YvhCHDKGAqAACgkb"
```

```
280         timeUnit="7XZCHDKGAqAACgkk" xmi:id="vXZCHDKGAqAACgkn"
281         xmi:type="timeInstance">
282           <xmi:Extension extender="Visual Paradigm">
283           <qualityScore value="-1"/>
284           </xmi:Extension>
285         </packagedElement>
286
287         <packagedElement name="" stateCondition="OIBCHDKGAqAACgka"
288         timeUnit="_C5CHDKGAqAACgko" xmi:id="oi5CHDKGAqAACgkr"
289         xmi:type="timeInstance">
290           <xmi:Extension extender="Visual Paradigm">
291           <qualityScore value="-1"/>
292           </xmi:Extension>
293         </packagedElement>
294
295         <packagedElement name="" stateCondition="OIBCHDKGAqAACgka"
296         timeUnit="yj5CHDKGAqAACgks" xmi:id="mj5CHDKGAqAACgkv"
297         xmi:type="timeInstance">
298           <xmi:Extension extender="Visual Paradigm">
299           <qualityScore value="-1"/>
300           </xmi:Extension>
301         </packagedElement>
302
303         <packagedElement name="" stateCondition="YvhCHDKGAqAACgkb"
304         timeUnit="ABVCHDKGAqAACgkw" xmi:id="YBVCHDKGAqAACgkz"
305         xmi:type="timeInstance">
306           <xmi:Extension extender="Visual Paradigm">
307           <qualityScore value="-1"/>
308           </xmi:Extension>
309         </packagedElement>
310
311         <packagedElement name="" stateCondition="OIBCHDKGAqAACgka"
312         timeUnit="RE1CHDKGAqAACgk0" xmi:id="FE1CHDKGAqAACgk3"
313         xmi:type="timeInstance">
314           <xmi:Extension extender="Visual Paradigm">
315           <qualityScore value="-1"/>
316           </xmi:Extension>
317         </packagedElement>
318
319         <packagedElement name="" stateCondition="YvhCHDKGAqAACgkb"
320         timeUnit="BB1CHDKGAqAACgk4" xmi:id="ZB1CHDKGAqAACgk7"
321         xmi:type="timeInstance">
322           <xmi:Extension extender="Visual Paradigm">
323           <qualityScore value="-1"/>
324           </xmi:Extension>
325         </packagedElement>
326
```

```
327       <packagedElement name="" stateCondition="YvhCHDKGAqAACgkb"
328       timeUnit="1L1CHDKGAqAACgk8" xmi:id="DL1CHDKGAqAACgk_"
329       xmi:type="timeInstance">
330         <xmi:Extension extender="Visual Paradigm">
331         <qualityScore value="-1"/>
332         </xmi:Extension>
333       </packagedElement>
334
335       <packagedElement name="" stateCondition="OIBCHDKGAqAACgka"
336       timeUnit="v4NCHDKGAqAACglA" xmi:id="QENCHDKGAqAACglD"
337       xmi:type="timeInstance">
338         <xmi:Extension extender="Visual Paradigm">
339         <qualityScore value="-1"/>
340         </xmi:Extension>
341       </packagedElement>
342
343       <packagedElement name="" stateCondition="YvhCHDKGAqAACgkb"
344       timeUnit="UeNCHDKGAqAACglE" xmi:id="ceNCHDKGAqAACglH"
345       xmi:type="timeInstance">
346         <xmi:Extension extender="Visual Paradigm">
347         <qualityScore value="-1"/>
348         </xmi:Extension>
349       </packagedElement>
350
351       <packagedElement name="" stateCondition="OIBCHDKGAqAACgka"
352       timeUnit="tDNCHDKGAqAACglI" xmi:id="TDNCHDKGAqAACglL"
353       xmi:type="timeInstance">
354         <xmi:Extension extender="Visual Paradigm">
355         <qualityScore value="-1"/>
356         </xmi:Extension>
357       </packagedElement>
358
359       <packagedElement name="" stateCondition="OIBCHDKGAqAACgka"
360       timeUnit="fwtCHDKGAqAACglM" xmi:id="QItCHDKGAqAACglP"
361       xmi:type="timeInstance">
362         <xmi:Extension extender="Visual Paradigm">
363         <qualityScore value="-1"/>
364         </xmi:Extension>
365     </packagedElement>
366   </packagedElement>
367
368   <packagedElement name="1" xmi:id="VvpCHDKGAqAACgkc" xmi:type="timeUnit">
369     <xmi:Extension extender="Visual Paradigm">
370     <qualityScore value="-1"/>
371     </xmi:Extension>
372   </packagedElement>
373
```

```
374     <packagedElement name="2" xmi:id="uBZCHDKGAqAACgkg" xmi:type="timeUnit">
375        <xmi:Extension extender="Visual Paradigm">
376        <qualityScore value="-1"/>
377        </xmi:Extension>
378     </packagedElement>
379
380     <packagedElement name="3" xmi:id="7XZCHDKGAqAACgkk" xmi:type="timeUnit">
381        <xmi:Extension extender="Visual Paradigm">
382        <qualityScore value="-1"/>
383        </xmi:Extension>
384     </packagedElement>
385
386     <packagedElement name="4" xmi:id="_C5CHDKGAqAACgko" xmi:type="timeUnit">
387        <xmi:Extension extender="Visual Paradigm">
388        <qualityScore value="-1"/>
389        </xmi:Extension>
390     </packagedElement>
391
392     <packagedElement name="5" xmi:id="yj5CHDKGAqAACgks" xmi:type="timeUnit">
393        <xmi:Extension extender="Visual Paradigm">
394        <qualityScore value="-1"/>
395        </xmi:Extension>
396     </packagedElement>
397
398     <packagedElement name="6" xmi:id="ABVCHDKGAqAACgkw" xmi:type="timeUnit">
399        <xmi:Extension extender="Visual Paradigm">
400        <qualityScore value="-1"/>
401        </xmi:Extension>
402     </packagedElement>
403
404      <packagedElement name="7" xmi:id="RE1CHDKGAqAACgk0" xmi:type="timeUnit">
405        <xmi:Extension extender="Visual Paradigm">
406        <qualityScore value="-1"/>
407        </xmi:Extension>
408      </packagedElement>
409
410      <packagedElement name="8" xmi:id="BB1CHDKGAqAACgk4" xmi:type="timeUnit">
411        <xmi:Extension extender="Visual Paradigm">
412        <qualityScore value="-1"/>
413        </xmi:Extension>
414      </packagedElement>
415
416      <packagedElement name="9" xmi:id="1L1CHDKGAqAACgk8" xmi:type="timeUnit">
417        <xmi:Extension extender="Visual Paradigm">
418        <qualityScore value="-1"/>
419        </xmi:Extension>
420      </packagedElement>
```

```
421
422     <packagedElement name="10" xmi:id="v4NCHDKGAqAACglA" xmi:type="timeUnit">
423       <xmi:Extension extender="Visual Paradigm">
424       <qualityScore value="-1"/>
425       </xmi:Extension>
426     </packagedElement>
427
428     <packagedElement name="11" xmi:id="UeNCHDKGAqAACglE" xmi:type="timeUnit">
429       <xmi:Extension extender="Visual Paradigm">
430       <qualityScore value="-1"/>
431       </xmi:Extension>
432     </packagedElement>
433
434     <packagedElement name="12" xmi:id="tDNCHDKGAqAACglI" xmi:type="timeUnit">
435       <xmi:Extension extender="Visual Paradigm">
436       <qualityScore value="-1"/>
437       </xmi:Extension>
438     </packagedElement>
439
440     <packagedElement name="13" xmi:id="fwtCHDKGAqAACglM" xmi:type="timeUnit">
441       <xmi:Extension extender="Visual Paradigm">
442       <qualityScore value="-1"/>
443       </xmi:Extension>
444     </packagedElement>
445   </packagedElement>
446
447   <packagedElement from="wLYCHDKGAqAACgkN" to=".A_iHDKGAqAACgmW"
448   xmi:id="HA_iHDKGAqAACgma" xmi:type="anchor">
449     <xmi:Extension extender="Visual Paradigm">
450     <qualityScore value="-1"/>
451     </xmi:Extension>
452   </packagedElement>
453 </packagedElement>
```

ANEXO B - CÓDIGO XMI REFERENTE AO DIAGRAMA DO ESTUDO DE CASO - CONTROLADORES SEMAFÓRICOS

Código 1 – Arquivo XMI referente ao diagrama de temporização do Semáforo

```
<packagedElement name="Semaforo" xmi:id="zQpfnDKGAqAACgxh" xmi:type="timingFrame">

  <packagedElement name="1" xmi:id="wMLfnDKGAqAACgxw" xmi:type="timeUnit">
    <xmi:Extension extender="Visual Paradigm">
    <qualityScore value="-1"/>   </xmi:Extension>
  </packagedElement>

  <packagedElement name="2" xmi:id="QHrfnDKGAqAACgx0" xmi:type="timeUnit">
    <xmi:Extension extender="Visual Paradigm">
    <qualityScore value="-1"/>   </xmi:Extension>
  </packagedElement>

  <packagedElement name="3" xmi:id="UCbfnDKGAqAACgx4" xmi:type="timeUnit">
    <xmi:Extension extender="Visual Paradigm">
    <qualityScore value="-1"/>   </xmi:Extension>
  </packagedElement>

  <packagedElement name="4" xmi:id="KxbfnDKGAqAACgx8" xmi:type="timeUnit">
    <xmi:Extension extender="Visual Paradigm">
    <qualityScore value="-1"/>   </xmi:Extension>
  </packagedElement>

  <packagedElement name="5" xmi:id="9w7fnDKGAqAACgyA" xmi:type="timeUnit">
    <xmi:Extension extender="Visual Paradigm">
    <qualityScore value="-1"/>   </xmi:Extension>
  </packagedElement>

  <packagedElement name="6" xmi:id="qK7fnDKGAqAACgyE" xmi:type="timeUnit">
    <xmi:Extension extender="Visual Paradigm">
    <qualityScore value="-1"/>   </xmi:Extension>
  </packagedElement>

  <packagedElement name="7" xmi:id="Ej7fnDKGAqAACgyI" xmi:type="timeUnit">
    <xmi:Extension extender="Visual Paradigm">
    <qualityScore value="-1"/>   </xmi:Extension>
  </packagedElement>

  <packagedElement name="8" xmi:id="VQHfnDKGAqAACgyM" xmi:type="timeUnit">
    <xmi:Extension extender="Visual Paradigm">
    <qualityScore value="-1"/>   </xmi:Extension>
  </packagedElement>

  <packagedElement name="9" xmi:id="mRHfnDKGAqAACgyQ" xmi:type="timeUnit">
    <xmi:Extension extender="Visual Paradigm">
```

```
45      <qualityScore value="-1"/>    </xmi:Extension>
46    </packagedElement>
47
48    <packagedElement name="10" xmi:id="3THfnDKGAqAACgyU" xmi:type="timeUnit">
49      <xmi:Extension extender="Visual Paradigm">
50      <qualityScore value="-1"/>    </xmi:Extension>
51    </packagedElement>
52
53    <packagedElement name="11" xmi:id="QMnfnDKGAqAACgyY" xmi:type="timeUnit">
54      <xmi:Extension extender="Visual Paradigm">
55      <qualityScore value="-1"/>    </xmi:Extension>
56    </packagedElement>
57
58    <packagedElement name="12" xmi:id="LenfnDKGAqAACgyc" xmi:type="timeUnit">
59      <xmi:Extension extender="Visual Paradigm">
60      <qualityScore value="-1"/>    </xmi:Extension>
61    </packagedElement>
62
63    <packagedElement name="13" xmi:id="fjnfnDKGAqAACgyg" xmi:type="timeUnit">
64      <xmi:Extension extender="Visual Paradigm">
65      <qualityScore value="-1"/>    </xmi:Extension>
66    </packagedElement>
67
68    <packagedElement name="14" xmi:id="qMXfnDKGAqAACgyk" xmi:type="timeUnit">
69      <xmi:Extension extender="Visual Paradigm">
70      <qualityScore value="-1"/>    </xmi:Extension>
71    </packagedElement>
72
73    <packagedElement name="15" xmi:id="ahXfnDKGAqAACgyo" xmi:type="timeUnit">
74      <xmi:Extension extender="Visual Paradigm">
75      <qualityScore value="-1"/>    </xmi:Extension>
76    </packagedElement>
77
78    <packagedElement name="16" xmi:id="AbXfnDKGAqAACgys" xmi:type="timeUnit">
79      <xmi:Extension extender="Visual Paradigm">
80      <qualityScore value="-1"/>    </xmi:Extension>
81    </packagedElement>
82
83    <packagedElement name="17" xmi:id="j23fnDKGAqAACgyw" xmi:type="timeUnit">
84      <xmi:Extension extender="Visual Paradigm">
85      <qualityScore value="-1"/>    </xmi:Extension>
86    </packagedElement>
87
88    <packagedElement name="18" xmi:id="oP3fnDKGAqAACgy0" xmi:type="timeUnit">
89      <xmi:Extension extender="Visual Paradigm">
90      <qualityScore value="-1"/>    </xmi:Extension>
91    </packagedElement>
```

```
92
93    <packagedElement name="SinalVerde" xmi:id="MYDfnDKGAqAACgxt"
94    xmi:type="timingFrameLifeline">
95      <packagedElement name="Desligado" xmi:id="s8TfnDKGAqAACgxv"
96      xmi:type="stateCondition">
97        <xmi:Extension extender="Visual Paradigm">
98       <qualityScore value="-1"/>    </xmi:Extension>
99      </packagedElement>
100
101     <packagedElement name="Ligado" xmi:id="I1jfnDKGAqAACgxu"
102     xmi:type="stateCondition">
103       <xmi:Extension extender="Visual Paradigm">
104       <qualityScore value="-1"/>    </xmi:Extension>
105     </packagedElement>
106
107     <packagedElement name="" stateCondition="I1jfnDKGAqAACgxu"
108     timeUnit="wMLfnDKGAqAACgxw" xmi:id="kMLfnDKGAqAACgxz"
109     xmi:type="timeInstance">
110       <xmi:Extension extender="Visual Paradigm">
111       <qualityScore value="-1"/>    </xmi:Extension>
112     </packagedElement>
113
114     <packagedElement name="" stateCondition="I1jfnDKGAqAACgxu"
115     timeUnit="QHrfnDKGAqAACgx0" xmi:id="YHrfnDKGAqAACgx3"
116     xmi:type="timeInstance">
117       <xmi:Extension extender="Visual Paradigm">
118       <qualityScore value="-1"/>    </xmi:Extension>
119     </packagedElement>
120
121     <packagedElement name="" stateCondition="I1jfnDKGAqAACgxu"
122     timeUnit="UCbfnDKGAqAACgx4" xmi:id="8CbfnDKGAqAACgx7"
123     xmi:type="timeInstance">
124       <xmi:Extension extender="Visual Paradigm">
125       <qualityScore value="-1"/>    </xmi:Extension>
126     </packagedElement>
127
128     <packagedElement name="" stateCondition="I1jfnDKGAqAACgxu"
129     timeUnit="KxbfnDKGAqAACgx8" xmi:id="mxbfnDKGAqAACgx_"
130     xmi:type="timeInstance">
131       <xmi:Extension extender="Visual Paradigm">
132       <qualityScore value="-1"/>    </xmi:Extension>
133     </packagedElement>
134
135     <packagedElement name="" stateCondition="I1jfnDKGAqAACgxu"
136     timeUnit="9w7fnDKGAqAACgyA" xmi:id="zw7fnDKGAqAACgyD"
137     xmi:type="timeInstance">
138       <xmi:Extension extender="Visual Paradigm">
```

```
139       <qualityScore value="-1"/>    </xmi:Extension>
140     </packagedElement>
141
142     <packagedElement name="" stateCondition="I1jfnDKGAqAACgxu"
143     timeUnit="qK7fnDKGAqAACgyE" xmi:id="WK7fnDKGAqAACgyH"
144     xmi:type="timeInstance">
145       <xmi:Extension extender="Visual Paradigm">
146       <qualityScore value="-1"/>    </xmi:Extension>
147     </packagedElement>
148
149     <packagedElement name="" stateCondition="I1jfnDKGAqAACgxu"
150     timeUnit="Ej7fnDKGAqAACgyI" xmi:id="sj7fnDKGAqAACgyL"
151     xmi:type="timeInstance">
152       <xmi:Extension extender="Visual Paradigm">
153       <qualityScore value="-1"/>    </xmi:Extension>
154     </packagedElement>
155
156     <packagedElement name="" stateCondition="I1jfnDKGAqAACgxu"
157     timeUnit="VQHfnDKGAqAACgyM" xmi:id="tQHfnDKGAqAACgyP"
158     xmi:type="timeInstance">
159       <xmi:Extension extender="Visual Paradigm">
160       <qualityScore value="-1"/>    </xmi:Extension>
161     </packagedElement>
162
163     <packagedElement name="" stateCondition="I1jfnDKGAqAACgxu"
164     timeUnit="mRHfnDKGAqAACgyQ" xmi:id="eRHfnDKGAqAACgyT"
165     xmi:type="timeInstance">
166       <xmi:Extension extender="Visual Paradigm">
167       <qualityScore value="-1"/>    </xmi:Extension>
168     </packagedElement>
169
170     <packagedElement name="" stateCondition="I1jfnDKGAqAACgxu"
171     timeUnit="3THfnDKGAqAACgyU" xmi:id="AzHfnDKGAqAACgyX"
172     xmi:type="timeInstance">
173       <xmi:Extension extender="Visual Paradigm">
174       <qualityScore value="-1"/>    </xmi:Extension>
175     </packagedElement>
176
177     <packagedElement name="" stateCondition="I1jfnDKGAqAACgxu"
178     timeUnit="QMnfnDKGAqAACgyY" xmi:id="4MnfnDKGAqAACgyb"
179     xmi:type="timeInstance">
180       <xmi:Extension extender="Visual Paradigm">
181       <qualityScore value="-1"/>    </xmi:Extension>
182     </packagedElement>
183
184     <packagedElement name="" stateCondition="I1jfnDKGAqAACgxu"
185     timeUnit="LenfnDKGAqAACgyc" xmi:id="nenfnDKGAqAACgyf"
```

```
186     xmi:type="timeInstance">
187       <xmi:Extension extender="Visual Paradigm">
188       <qualityScore value="-1"/>    </xmi:Extension>
189     </packagedElement>
190
191     <packagedElement name="" stateCondition="s8TfnDKGAqAACgxv"
192     timeUnit="fjnfnDKGAqAACgyg" xmi:id="ITnfnDKGAqAACgyj"
193     xmi:type="timeInstance">
194       <xmi:Extension extender="Visual Paradigm">
195       <qualityScore value="-1"/>    </xmi:Extension>
196     </packagedElement>
197
198     <packagedElement name="" stateCondition="s8TfnDKGAqAACgxv"
199     timeUnit="qMXfnDKGAqAACgyk" xmi:id="mMXfnDKGAqAACgyn"
200     xmi:type="timeInstance">
201       <xmi:Extension extender="Visual Paradigm">
202       <qualityScore value="-1"/>    </xmi:Extension>
203     </packagedElement>
204
205     <packagedElement name="" stateCondition="s8TfnDKGAqAACgxv"
206     timeUnit="ahXfnDKGAqAACgyo" xmi:id="mhXfnDKGAqAACgyr"
207     xmi:type="timeInstance">
208       <xmi:Extension extender="Visual Paradigm">
209       <qualityScore value="-1"/>    </xmi:Extension>
210     </packagedElement>
211
212     <packagedElement name="" stateCondition="s8TfnDKGAqAACgxv"
213     timeUnit="AbXfnDKGAqAACgys" xmi:id="IbXfnDKGAqAACgyv"
214     xmi:type="timeInstance">
215       <xmi:Extension extender="Visual Paradigm">
216       <qualityScore value="-1"/>    </xmi:Extension>
217     </packagedElement>
218
219     <packagedElement name="" stateCondition="s8TfnDKGAqAACgxv"
220     timeUnit="j23fnDKGAqAACgyw" xmi:id="b23fnDKGAqAACgyz"
221     xmi:type="timeInstance">
222       <xmi:Extension extender="Visual Paradigm">
223       <qualityScore value="-1"/>    </xmi:Extension>
224     </packagedElement>
225
226     <packagedElement name="" stateCondition="s8TfnDKGAqAACgxv"
227     timeUnit="oP3fnDKGAqAACgy0" xmi:id="sP3fnDKGAqAACgy3"
228     xmi:type="timeInstance">
229       <xmi:Extension extender="Visual Paradigm">
230       <qualityScore value="-1"/>    </xmi:Extension>
231     </packagedElement>
232   </packagedElement>
```

```
233
234   <packagedElement name="SinalAmarelo" xmi:id="9etfnDKGAqAACgxq"
235   xmi:type="timingFrameLifeline">
236     <packagedElement name="Desligado" xmi:id="Ga9fnDKGAqAACgxs"
237     xmi:type="stateCondition">
238       <xmi:Extension extender="Visual Paradigm">
239       <qualityScore value="-1"/>    </xmi:Extension>
240     </packagedElement>
241
242     <packagedElement name="Ligado" xmi:id="EbdfnDKGAqAACgxr"
243     xmi:type="stateCondition">
244       <xmi:Extension extender="Visual Paradigm">
245       <qualityScore value="-1"/>    </xmi:Extension>
246     </packagedElement>
247
248     <packagedElement name="" stateCondition="Ga9fnDKGAqAACgxs"
249     timeUnit="wMLfnDKGAqAACgxw" xmi:id="EMLfnDKGAqAACgxy"
250     xmi:type="timeInstance">
251       <xmi:Extension extender="Visual Paradigm">
252       <qualityScore value="-1"/>    </xmi:Extension>
253     </packagedElement>
254
255     <packagedElement name="" stateCondition="Ga9fnDKGAqAACgxs"
256     timeUnit="QHrfnDKGAqAACgx0" xmi:id="oHrfnDKGAqAACgx2"
257     xmi:type="timeInstance">
258       <xmi:Extension extender="Visual Paradigm">
259       <qualityScore value="-1"/>    </xmi:Extension>
260     </packagedElement>
261
262     <packagedElement name="" stateCondition="Ga9fnDKGAqAACgxs"
263     timeUnit="UCbfnDKGAqAACgx4" xmi:id="cCbfnDKGAqAACgx6"
264     xmi:type="timeInstance">
265       <xmi:Extension extender="Visual Paradigm">
266       <qualityScore value="-1"/>    </xmi:Extension>
267     </packagedElement>
268
269     <packagedElement name="" stateCondition="Ga9fnDKGAqAACgxs"
270     timeUnit="KxbfnDKGAqAACgx8" xmi:id="GxbfnDKGAqAACgx."
271     xmi:type="timeInstance">
272       <xmi:Extension extender="Visual Paradigm">
273       <qualityScore value="-1"/>    </xmi:Extension>
274     </packagedElement>
275
276     <packagedElement name="" stateCondition="Ga9fnDKGAqAACgxs"
277     timeUnit="9w7fnDKGAqAACgyA" xmi:id="jw7fnDKGAqAACgyC"
278     xmi:type="timeInstance">
279       <xmi:Extension extender="Visual Paradigm">
```

```
280       <qualityScore value="-1"/>    </xmi:Extension>
281     </packagedElement>
282
283     <packagedElement name="" stateCondition="Ga9fnDKGAqAACgxs"
284     timeUnit="qK7fnDKGAqAACgyE" xmi:id="mK7fnDKGAqAACgyG"
285     xmi:type="timeInstance">
286       <xmi:Extension extender="Visual Paradigm">
287       <qualityScore value="-1"/>    </xmi:Extension>
288     </packagedElement>
289
290     <packagedElement name="" stateCondition="Ga9fnDKGAqAACgxs"
291     timeUnit="Ej7fnDKGAqAACgyI" xmi:id="Mj7fnDKGAqAACgyK"
292     xmi:type="timeInstance">
293       <xmi:Extension extender="Visual Paradigm">
294       <qualityScore value="-1"/>    </xmi:Extension>
295     </packagedElement>
296
297     <packagedElement name="" stateCondition="Ga9fnDKGAqAACgxs"
298     timeUnit="VQHfnDKGAqAACgyM" xmi:id="NQHfnDKGAqAACgyO"
299     xmi:type="timeInstance">
300       <xmi:Extension extender="Visual Paradigm">
301       <qualityScore value="-1"/>    </xmi:Extension>
302     </packagedElement>
303
304     <packagedElement name="" stateCondition="Ga9fnDKGAqAACgxs"
305     timeUnit="mRHfnDKGAqAACgyQ" xmi:id="uRHfnDKGAqAACgyS"
306     xmi:type="timeInstance">
307       <xmi:Extension extender="Visual Paradigm">
308       <qualityScore value="-1"/>    </xmi:Extension>
309     </packagedElement>
310
311     <packagedElement name="" stateCondition="Ga9fnDKGAqAACgxs"
312     timeUnit="3THfnDKGAqAACgyU" xmi:id="_THfnDKGAqAACgyW"
313     xmi:type="timeInstance">
314       <xmi:Extension extender="Visual Paradigm">
315       <qualityScore value="-1"/>    </xmi:Extension>
316     </packagedElement>
317
318     <packagedElement name="" stateCondition="Ga9fnDKGAqAACgxs"
319     timeUnit="QMnfnDKGAqAACgyY" xmi:id="YMnfnDKGAqAACgya"
320     xmi:type="timeInstance">
321       <xmi:Extension extender="Visual Paradigm">
322       <qualityScore value="-1"/>    </xmi:Extension>
323     </packagedElement>
324
325     <packagedElement name="" stateCondition="Ga9fnDKGAqAACgxs"
326     timeUnit="LenfnDKGAqAACgyc" xmi:id="HenfnDKGAqAACgye"
```

```
327     xmi:type="timeInstance">
328       <xmi:Extension extender="Visual Paradigm">
329       <qualityScore value="-1"/>    </xmi:Extension>
330     </packagedElement>
331
332     <packagedElement name="" stateCondition="EbdfnDKGAqAACgxr"
333     timeUnit="fjnfnDKGAqAACgyg" xmi:id="wTnfnDKGAqAACgyi"
334     xmi:type="timeInstance">
335       <xmi:Extension extender="Visual Paradigm">
336       <qualityScore value="-1"/>    </xmi:Extension>
337     </packagedElement>
338
339     <packagedElement name="" stateCondition="EbdfnDKGAqAACgxr"
340     timeUnit="qMXfnDKGAqAACgyk" xmi:id="GMXfnDKGAqAACgym"
341     xmi:type="timeInstance">
342       <xmi:Extension extender="Visual Paradigm">
343       <qualityScore value="-1"/>    </xmi:Extension>
344     </packagedElement>
345
346     <packagedElement name="" stateCondition="EbdfnDKGAqAACgxr"
347     timeUnit="ahXfnDKGAqAACgyo" xmi:id="GhXfnDKGAqAACgyq"
348     xmi:type="timeInstance">
349       <xmi:Extension extender="Visual Paradigm">
350       <qualityScore value="-1"/>    </xmi:Extension>
351     </packagedElement>
352
353     <packagedElement name="" stateCondition="Ga9fnDKGAqAACgxs"
354     timeUnit="AbXfnDKGAqAACgys" xmi:id="wbXfnDKGAqAACgyu"
355     xmi:type="timeInstance">
356       <xmi:Extension extender="Visual Paradigm">
357       <qualityScore value="-1"/>    </xmi:Extension>
358     </packagedElement>
359
360     <packagedElement name="" stateCondition="Ga9fnDKGAqAACgxs"
361     timeUnit="j23fnDKGAqAACgyw" xmi:id="r23fnDKGAqAACgyy"
362     xmi:type="timeInstance">
363       <xmi:Extension extender="Visual Paradigm">
364       <qualityScore value="-1"/>    </xmi:Extension>
365     </packagedElement>
366
367     <packagedElement name="" stateCondition="Ga9fnDKGAqAACgxs"
368     timeUnit="oP3fnDKGAqAACgy0" xmi:id="MP3fnDKGAqAACgy2"
369     xmi:type="timeInstance">
370       <xmi:Extension extender="Visual Paradigm">
371       <qualityScore value="-1"/>    </xmi:Extension>
372     </packagedElement>
373   </packagedElement>
```

```
374
375   <packagedElement name="SinalVermelho" xmi:id="0EVfnDKGAqAACgxn"
376   xmi:type="timingFrameLifeline">
377     <packagedElement name="Desligado" xmi:id="R.NfnDKGAqAACgxp"
378     xmi:type="stateCondition">
379       <xmi:Extension extender="Visual Paradigm">
380       <qualityScore value="-1"/>   </xmi:Extension>
381     </packagedElement>
382
383     <packagedElement name="Ligado" xmi:id="zT1fnDKGAqAACgxo"
384     xmi:type="stateCondition">
385       <xmi:Extension extender="Visual Paradigm">
386       <qualityScore value="-1"/>   </xmi:Extension>
387     </packagedElement>
388
389     <packagedElement name="" stateCondition="R.NfnDKGAqAACgxp"
390     timeUnit="wMLfnDKGAqAACgxw" xmi:id="4MLfnDKGAqAACgxx"
391     xmi:type="timeInstance">
392       <xmi:Extension extender="Visual Paradigm">
393       <qualityScore value="-1"/>   </xmi:Extension>
394     </packagedElement>
395
396     <packagedElement name="" stateCondition="R.NfnDKGAqAACgxp"
397     timeUnit="QHrfnDKGAqAACgx0" xmi:id="IHrfnDKGAqAACgx1"
398     xmi:type="timeInstance">
399       <xmi:Extension extender="Visual Paradigm">
400       <qualityScore value="-1"/>   </xmi:Extension>
401     </packagedElement>
402
403     <packagedElement name="" stateCondition="R.NfnDKGAqAACgxp"
404     timeUnit="UCbfnDKGAqAACgx4" xmi:id="MCbfnDKGAqAACgx5"
405     xmi:type="timeInstance">
406       <xmi:Extension extender="Visual Paradigm">
407       <qualityScore value="-1"/>   </xmi:Extension>
408     </packagedElement>
409
410     <packagedElement name="" stateCondition="R.NfnDKGAqAACgxp"
411     timeUnit="KxbfnDKGAqAACgx8" xmi:id="axbfnDKGAqAACgx9"
412     xmi:type="timeInstance">
413       <xmi:Extension extender="Visual Paradigm">
414       <qualityScore value="-1"/>   </xmi:Extension>
415     </packagedElement>
416
417     <packagedElement name="" stateCondition="R.NfnDKGAqAACgxp"
418     timeUnit="9w7fnDKGAqAACgyA" xmi:id="Dw7fnDKGAqAACgyB"
419     xmi:type="timeInstance">
420       <xmi:Extension extender="Visual Paradigm">
```

```
421       <qualityScore value="-1"/>    </xmi:Extension>
422     </packagedElement>
423
424     <packagedElement name="" stateCondition="R.NfnDKGAqAACgxp"
425     timeUnit="qK7fnDKGAqAACgyE" xmi:id="6K7fnDKGAqAACgyF"
426     xmi:type="timeInstance">
427       <xmi:Extension extender="Visual Paradigm">
428       <qualityScore value="-1"/>    </xmi:Extension>
429     </packagedElement>
430
431     <packagedElement name="" stateCondition="R.NfnDKGAqAACgxp"
432     timeUnit="Ej7fnDKGAqAACgyI" xmi:id="Uj7fnDKGAqAACgyJ"
433     xmi:type="timeInstance">
434       <xmi:Extension extender="Visual Paradigm">
435       <qualityScore value="-1"/>    </xmi:Extension>
436     </packagedElement>
437
438     <packagedElement name="" stateCondition="R.NfnDKGAqAACgxp"
439     timeUnit="VQHfnDKGAqAACgyM" xmi:id="1QHfnDKGAqAACgyN"
440     xmi:type="timeInstance">
441       <xmi:Extension extender="Visual Paradigm">
442       <qualityScore value="-1"/>    </xmi:Extension>
443     </packagedElement>
444
445     <packagedElement name="" stateCondition="R.NfnDKGAqAACgxp"
446     timeUnit="mRHfnDKGAqAACgyQ" xmi:id="ORHfnDKGAqAACgyR"
447     xmi:type="timeInstance">
448       <xmi:Extension extender="Visual Paradigm">
449       <qualityScore value="-1"/>    </xmi:Extension>
450     </packagedElement>
451
452     <packagedElement name="" stateCondition="R.NfnDKGAqAACgxp"
453     timeUnit="3THfnDKGAqAACgyU" xmi:id="vTHfnDKGAqAACgyV"
454     xmi:type="timeInstance">
455       <xmi:Extension extender="Visual Paradigm">
456       <qualityScore value="-1"/>    </xmi:Extension>
457     </packagedElement>
458
459     <packagedElement name="" stateCondition="R.NfnDKGAqAACgxp"
460     timeUnit="QMnfnDKGAqAACgyY" xmi:id="IMnfnDKGAqAACgyZ"
461     xmi:type="timeInstance">
462       <xmi:Extension extender="Visual Paradigm">
463       <qualityScore value="-1"/>    </xmi:Extension>
464     </packagedElement>
465
466     <packagedElement name="" stateCondition="R.NfnDKGAqAACgxp"
467     timeUnit="LenfnDKGAqAACgyc" xmi:id="benfnDKGAqAACgyd"
```

```
468     xmi:type="timeInstance">
469       <xmi:Extension extender="Visual Paradigm">
470       <qualityScore value="-1"/>    </xmi:Extension>
471     </packagedElement>
472
473     <packagedElement name="" stateCondition="R.NfnDKGAqAACgxp"
474     timeUnit="fjnfnDKGAqAACgyg" xmi:id="gTnfnDKGAqAACgyh"
475     xmi:type="timeInstance">
476       <xmi:Extension extender="Visual Paradigm">
477       <qualityScore value="-1"/>    </xmi:Extension>
478     </packagedElement>
479
480     <packagedElement name="" stateCondition="R.NfnDKGAqAACgxp"
481     timeUnit="qMXfnDKGAqAACgyk" xmi:id="6MXfnDKGAqAACgyl"
482     xmi:type="timeInstance">
483       <xmi:Extension extender="Visual Paradigm">
484       <qualityScore value="-1"/>    </xmi:Extension>
485     </packagedElement>
486
487     <packagedElement name="" stateCondition="R.NfnDKGAqAACgxp"
488     timeUnit="ahXfnDKGAqAACgyo" xmi:id="6hXfnDKGAqAACgyp"
489     xmi:type="timeInstance">
490       <xmi:Extension extender="Visual Paradigm">
491       <qualityScore value="-1"/>    </xmi:Extension>
492     </packagedElement>
493
494     <packagedElement name="" stateCondition="zT1fnDKGAqAACgxo"
495     timeUnit="AbXfnDKGAqAACgys" xmi:id="QbXfnDKGAqAACgyt"
496     xmi:type="timeInstance">
497       <xmi:Extension extender="Visual Paradigm">
498       <qualityScore value="-1"/>    </xmi:Extension>
499     </packagedElement>
500
501     <packagedElement name="" stateCondition="zT1fnDKGAqAACgxo"
502     timeUnit="j23fnDKGAqAACgyw" xmi:id="z23fnDKGAqAACgyx"
503     xmi:type="timeInstance">
504       <xmi:Extension extender="Visual Paradigm">
505       <qualityScore value="-1"/>    </xmi:Extension>
506     </packagedElement>
507
508     <packagedElement name="" stateCondition="zT1fnDKGAqAACgxo"
509     timeUnit="oP3fnDKGAqAACgy0" xmi:id="UP3fnDKGAqAACgy1"
510     xmi:type="timeInstance">
511       <xmi:Extension extender="Visual Paradigm">
512       <qualityScore value="-1"/>    </xmi:Extension>
513     </packagedElement>
514 </packagedElement>
```