UNIVERSIDADE FEDERAL DE SÃO CARLOS

CENTRO DE CIÊNCIAS EXATAS E DE TECNOLOGIA

PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

UMA INVESTIGAÇÃO SOBRE O USO DA UML STATECHART PARA REPRESENTAR O COMPORTAMENTO DE APLICAÇÕES MODELADAS EM MATLAB/SIMULINK

GUILHERME MENDONÇA FREIRE

ORIENTADORA: PROFª. DRª. SANDRA CAMARGO P. F. FABBRI

São Carlos - SP
Novembro/2011

UNIVERSIDADE FEDERAL DE SÃO CARLOS

CENTRO DE CIÊNCIAS EXATAS E DE TECNOLOGIA

PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

UMA INVESTIGAÇÃO SOBRE O USO DA UML STATECHART PARA REPRESENTAR O COMPORTAMENTO DE APLICAÇÕES MODELADAS EM MATLAB/SIMULINK

GUILHERME MENDONÇA FREIRE

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de São Carlos, como parte dos requisitos para a obtenção do título de Mestre em Ciência da Computação, área de concentração: Engenharia de Software.
Orientadora: Dra. Sandra Camargo Pinto Ferraz Fabbri.

São Carlos - SP
Novembro/2011

Ficha catalográfica elaborada pelo DePT da Biblioteca Comunitária da UFSCar

F866iu

Freire, Guilherme Mendonça.
Uma investigação sobre o uso da UML Statechart para representar o comportamento de aplicações modeladas em Matlab/Simulink / Guilherme Mendonça Freire. -- São Carlos : UFSCar, 2013.
132 f.

Dissertação (Mestrado) -- Universidade Federal de São Carlos, 2011.

1. Engenharia de software. 2. Sistemas embarcados. 3. UML Statechart. 4. SIMULINK (Programa de computador). 5. Modelagem. 6. Qualidade de software. I. Título.

CDD: 005.1 (20ª)

UNIVERSIDADE FEDERAL DE SÃO CARLOS

CENTRO DE CIÊNCIAS EXATAS E DE TECNOLOGIA

PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

UMA INVESTIGAÇÃO SOBRE O USO DA UML STATECHART PARA REPRESENTAR O COMPORTAMENTO DE APLICAÇÕES MODELADAS EM MATLAB/SIMULINK

GUILHERME MENDONÇA FREIRE

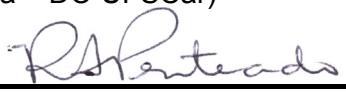
Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de São Carlos, como parte dos requisitos para a obtenção do título de Mestre em Ciência da Computação, área de concentração: Engenharia de Software.

Aprovado em 13 de dezembro de 2011.

Membros da Banca:

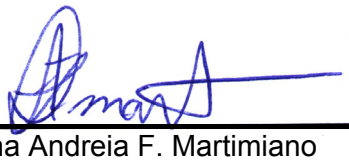
Profª. Drª. Sandra Camargo Pinto Ferraz Fabbri

(Orientadora – DC-UFSCar)



Profª. Drª. Rosângela Aparecida Dellosso Penteado

(DC-UFSCar)



Profª. Drª. Luciana Andreia F. Martimiano

(UEM)

São Carlos - SP
Dezembro/2011

Dedicatória

Dedico esse trabalho a todos aqueles que me ajudaram com amor e carinho. Aos meus pais, meus irmãos, minha namorada em especial, aos meus colegas e amigos e principalmente à Professora Sandra Fabbri.

AGRADECIMENTOS

Agradeço de coração àquelas pessoas que me ajudaram a conquistar esse título.

Agradeço aos colegas que se tornaram amigos André, Fábio, Juciara, Gilmar, Elis, Deyse, Kamilla, Rafael, Alexandre, Gabriel, Carlos Eduardo e Hiro e a todos que estiveram presentes pelo companheirismo e auxílio.

Aos amigos Jefferson, Natália, Aline, Feu, Tácito, Ed, Ponce, Felipe Duarte, Karina Emboaba, Felipe Magalhães e Eduardo pelos bons tempos e conselho moral.

Aos meus pais Evandro Sena Freire e Ana Maria Mendonça Freire por acreditarem, meus irmãos Gustavo e Viviane pelo apoio.

À minha querida namorada Daniele Magnavita pelos conselhos, paciência e perseverança e à professora Sandra Camargo Pinto Ferraz Fabbri.

À CAPES e ao INCT-SEC pelo auxílio financeiro e a bolsa concedida durante a realização desta pesquisa.

Agradeço a todos de coração.

"Tolice é viver uma vida assim, sem aventura".

Luiz Maurício Pragana dos Santos

RESUMO

Contexto: Os sistemas embarcados (SE) têm se tornado cada vez mais presentes na vida das pessoas em decorrência dos avanços tecnológicos e do aumento na diversidade em suas áreas de aplicação. Em geral, os desenvolvedores desse tipo de sistema iniciam o desenvolvimento com o uso de ferramentas do tipo Matlab/Simulink, elaborando modelos em um nível baixo de abstração, que são construídos de forma desorganizada, o que dificulta o entendimento da aplicação. Em decorrência disso, observam-se iniciativas de uso da engenharia de software nesse tipo de sistema. Objetivo: O grupo de pesquisa no qual este trabalho foi desenvolvido está definindo um processo para o desenvolvimento de SEs que dê suporte das fases iniciais até a construção dos modelos Simulink. Em particular, este trabalho tem o objetivo de explorar o uso da UML Statechart para retratar o modelo Simulink em um nível mais alto de abstração, tornando-se então um dos componentes do processo almejado pelo grupo. Método: Para fazer essa investigação, utilizou-se a metodologia Pesquisa-Ação e três ações foram conduzidas, explorando-se a UML Statechart em: (i) uma atividade de reengenharia, partindo do Simulink e reestruturando-o; (ii) em uma atividade de engenharia avante, partindo de um documento de requisitos; e (iii) em um *survey*, caracterizando o uso da UMS Statechart por desenvolvedores de SEs. Resultados: Nas duas primeiras ações, a UML Statechart se mostrou apropriada para representar o comportamento da aplicação, de forma a contribuir na construção do modelo Simulink, facilitando a compreensão do sistema como um todo e permitindo a elaboração de um Simulink mais organizado. Conclusões: Os resultados obtidos nas ações permitem considerar que a UML Statechart é uma forte candidata para compor um dos artefatos a serem construídos durante o processo que está sendo definido pelo grupo de pesquisa. Embora as investigações conduzidas neste trabalho estejam limitadas a duas aplicações, requerendo que outros estudos sejam realizados, pode-se dizer que os modelos Simulink construídos com o apoio do modelo UML Statechart são mais estruturados e mais compreensíveis. Isso caracteriza uma melhora de qualidade no desenvolvimento de SEs.

Palavras-chave: Sistemas Embarcados, UML Statechart, Matlab/Simulink, Modelagem, Qualidade de Software.

ABSTRACT

Context: Embedded systems (ES) has become more important to everyday life due to technology advance and increasing application field. Engineers start ES development using tools like Matlab/Simulink. Usually, Simulink models are low level abstraction models following an ad-hoc design, which difficult the model comprehension. Hence, new trends start to apply software engineering to support ES design. Goal: This work was developed in a research group that is defining a development process to support ES development from initial phases to Simulink models implementation. Particularly, this works goal is to explore UML Statchart technique to depict Simulink models in a high level abstraction view, and become the group's desired component part of the process. Method: The investigation process was conducted in three actions exploring UML Statchart based in the Research Action methodology: (i) a reengineering activity starting from Simulink model; (ii) a forward engineering starting from a requirement document; and (iii) a survey investigating the use of UML Statechart by ES developers. Outcomes: The first and second action showed that UML Statechart is an appropriated technique to depict the application behavior, contributing to implement Simulink models. It also organizes Simulink models and facilitates the system comprehension as a whole. Conclusion: Due to the conducted actions outcomes, UML Statechart can be considered as a candidate to compose one of the artifacts to be implemented during the process that is being defined by the research group. However, this work conducted investigations are limited to two system applications, needing to accomplish more complement studies, we can say that Simulink models implemented with support of UML Statechart, are better structured and more comprehensive. This features an increasing quality in ES design.

Keywords: Embedded Systems, UML Statechart, Matlab/Simulink, Model-based Development, Software Quality.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

ABS – *Antilock Breaking System*

CAPES – *Coordenação de Aperfeiçoamento de Pessoal de Nível Superior*

CBSEC – *Conferência Brasileira de Sistemas Embarcados Críticos*

DOF – *Degree Of Freedom*

DVD – *Digital Versatile Disc*

ETCS – *Electric Train Control System*

GPS – *Global Positioning Systems*

HW – *Hardware*

HRTS – *Hard Real-Time System*

INCT – *Instituto Nacional de Ciência e Tecnologia*

LabVIEW – *Laboratory Virtual Instrument Engineering Workbench*

LaPES – *Laboratório de Pesquisa em Engenharia de Software*

LTL – *Linear-time Temporal Logic*

MARTE – *Modeling and Analysis of Real-Time and Embedded systems*

MATLAB – *MATrix LABoratory*

MBD – *Model-Based Development (Desenvolvimento Baseado em Modelos)*

MDA – *Model-Driven Architecture (Modelagem Orientada a Arquiteturas)*

MEC – *Ministério da Educação*

MEF – *Máquina de Estados Finitos*

MoC – *Model of Computation*

OMG – *Object Management Group*

OO – *Orientação a Objetos*

OOA – *Objtect-Oriented Analysis (Análise Orientada a Objetos)*

OOD – *Object-Oriented Design (Projeto Orientado a Objetos)*

PBD – *Platform-Based Design*

PDAs – *Personal Data Assistants*

RT – *Real-time*

SE – *Sistemas Embarcados*

SEC – *Sistemas Embarcados Críticos*

SEEP – *Sistemas Eletrônicos Embarcados Baseados em Plataforma*

SETR – *Sistemas Embarcados de Tempo Real*

SRTS – *Soft Real-Time Systems*

SPT – *Stochastic Petri Net*

STR – *Sistemas de Tempo Real*

SW – *Software*

SysML – *Systems Modeling Language*

TI – *Tecnologia da Informação*

UFSCar – *Universidade Federal de São Carlos*

UML – *Unified Modeling Language*

VHDL – *VHSIC hardware description language*

VHSIC – *Very-High-Speed Integrated Circuits*

VI – *Virtual Interface*

SUMÁRIO

Capítulo 1

INTRODUÇÃO – DIAGNÓSTICO DA PESQUISA-AÇÃO

Este capítulo corresponde à etapa de Diagnóstico da metodologia de Pesquisa-Ação. Neste capítulo é descrito o problema, o contexto do projeto e o tema de pesquisa.

Sistemas Embarcados (SE) são sistemas computacionais que exercem um conjunto restrito de funcionalidades. O desenvolvimento de um SE envolve uma interação altamente dependente entre hardware e software. Em consequência dessa interação fortemente acoplada, o desenvolvimento de SE é uma atividade bastante complexa, em que os requisitos do sistema necessitam estar bem documentados. Assim como em sistemas de software de propósito geral, a documentação de um projeto de SE irá garantir maior qualidades dos produtos desenvolvidos (EBERT e JONES, 2009).

Entretanto, existem ainda fatores que aumentam a complexidade de desenvolvimento de um SE, como por exemplo:

- Tecnologia – a cada dia a tecnologia sofre avanços que podem tornar complexo o desenvolvimento de sistemas embarcados;
- Aplicação – a aplicação do SE, no que se refere ao ambiente em que irá interagir, tanto quanto o ambiente em que será inserido, sendo uma parte de um todo, pode influenciar na complexidade de desenvolvimento;
- Funcionalidades – O número de funcionalidades pode influenciar diretamente na complexidade de desenvolvimento.

O avanço tecnológico permitiu que a aplicação de sistemas embarcados pudesse ser estendida a áreas que antes não eram aplicadas. Consequentemente, com o aumento da área de aplicação, estenderam-se as funcionalidades que um SE pode desempenhar (EBERT e SALECKER, 2009).

O desenvolvimento de um sistema embarcado pode conter funcionalidades que variam em complexidade e grau de risco. A complexidade das funcionalidades pode ser simples de forma a automatizar e aperfeiçoar uma tarefa diária, antes exercida manualmente, ou complexa que exigiria muito esforço físico ou intelectual, tomando muito tempo para ser concluída. O grau de risco está relacionado com o quanto uma atividade executada por um SE pode provocar risco à saúde de um ser humano. Na área da saúde encontram-se diversos exemplos desse tipo de aplicação.

A partir das características citadas, o desenvolvimento de um SE é uma atividade complexa, pois diversas questões precisam ser consideradas. Para que produtos de melhor qualidade sejam desenvolvidos, tem-se percebido uma crescente tendência de se aplicarem técnicas, métodos e processos da engenharia de software que permitam orientar e qualificar o desenvolvimento de SE. As técnicas e métodos utilizados pela engenharia de software estão bem fundamentados para o desenvolvimento de sistemas de software tradicionais, mas o mesmo não acontece para os SE. Assim, vários trabalhos de pesquisa têm objetivado explorar e adaptar essas técnicas e métodos para o contexto de desenvolvimento de SE.

Com base em uma revisão bibliográfica realizada para iniciar o estudo deste trabalho, observou-se que diversas propostas exploram ferramentas comumente utilizadas para o desenvolvimento de SE, como Simulink (*MathWorks*, 2011), LabVIEW (*National Instruments*, 2011), SystemC (SystemC, 2011), Verilog (Verilog, 2011), entre outras. Em outros trabalhos, a tendência apresentada é o uso de técnicas de modelagem da UML (*Unified Modeling Language*, 2011), utilizando *Profiles*. Os *profiles* da UML são extensões que permitem enriquecer um modelo com notações características do domínio em desenvolvimento. Entretanto, as propostas de trabalho mais interessantes são aquelas que conciliam as duas abordagens descrita anteriormente, ou seja, utilizar ferramentas comumente adotadas para o desenvolvimento de SE com técnicas da UML.

Assim, o projeto do grupo de pesquisa que está sendo desenvolvido no âmbito do projeto INCT-SEC (Instituto Nacional de Ciência e Tecnologia - Sistemas

Embarcados Críticos, 2011), e no qual este trabalho está inserido, tem como objetivo definir um processo de desenvolvimento de SE que contribua para melhorar a qualidade do desenvolvimento desse tipo de aplicação. Esse processo vai contemplar as fases de desenvolvimento que vão desde a elicitação de requisitos até a modelagem de baixo nível, para a qual, pretende-se focar, em um primeiro momento, nos modelos Simulink.

O Matlab/Simulilnk foi escolhido, pois é a ferramenta que a maioria dos desenvolvedores de SE que participam do INCT-SEC utiliza. Em particular, considerando o processo como um todo, o trabalho associado a este mestrado tem o foco na modelagem que deve anteceder ao modelo Simulink, tanto do ponto de vista da engenharia reversa, como da engenharia avante. Além disso, estabeleceu-se investigar, como alternativa para elaboração dessa modelagem, a técnica UML Statechart (*Unified Modeling Language*, 2011).

No entanto, como foi mencionado anteriormente, o uso da engenharia de software nesse domínio de aplicação ainda é restrito e as pesquisas estão se intensificando mais recentemente. Dessa forma, adotou-se como método de pesquisa para realizar este trabalho, a Pesquisa-Ação (STRINGER, 2007), explicada no Apêndice A. Resumidamente, esse método estabelece um objetivo de pesquisa, o qual é investigado de maneira iterativa, por meio da aplicação de ações, que promovem, a cada iteração, um ganho de conhecimento.

A condução deste trabalho, com base na metodologia Pesquisa-Ação, implicou na realização de três iterações relacionadas ao objetivo de identificar uma forma de representar modelos Simulink em um nível mais alto de abstração. Duas dessas iterações, ou seja, duas ações focam essa questão com base em duas aplicações distintas: uma mão robótica e um sistema de controle de refrigeração. Uma terceira ação, que foi executada paralelamente às outras duas, refere-se a um levantamento de como os desenvolvedores de SE constroem suas aplicações. O objetivo dessa terceira ação foi agregar informações para enriquecer as análises e possíveis conclusões que poderiam ser traçadas de uma forma mais geral.

A aplicação utilizada na primeira iteração de ações corresponde a uma mão robótica, cujo modelo Simulink descreve a tarefa de manipulação de objetos por essa mão. Nessa primeira ação o modelo Simulink já tinha sido elaborado e foi aplicada uma engenharia reversa para representar as funcionalidades do modelo no diagrama UML Statechart. Em seguida, a partir do diagrama UML Statechart, o

modelo Simulink foi refeito por meio de uma reengenharia permitindo uma disposição dos blocos mais organizada e, consequentemente, mais legível.

A Tabela 1.1 sintetiza as etapas da Pesquisa-Ação relacionando o capítulo deste trabalho que trata cada uma dessas etapas.

Tabela 1.1 - Organização do Trabalho

Etapas da Pesquisa-Ação	Capítulo no qual a etapa é tratada
1. Diagnóstico	Capítulo 1 Introdução – Diagnóstico da Pesquisa-Ação
a. Descrição do Problema	
b. Contexto do Projeto	
c. Tema da Pesquisa	
2. Planejamento	Capítulo 2 Revisão Bibliográfica – Planejamento da Pesquisa-Ação
a. Revisão Bibliográfica	
i. Estudo Preliminar	
b. Foco da Ação	Capítulo 3 Mão Robótica – Planejamento da Pesquisa-Ação – Ação 1 Capítulo 4 Controle de Refrigeração – Planejamento da Pesquisa-Ação – Ação 2 Capítulo 5 Aplicação do *Survey* – Planejamento da Pesquisa-Ação – Ação 3
i. Objetivos	
ii. Questão da Pesquisa	
iii. Resultados Esperados	
c. Hipótese	
d. Definições Operacionais	
i. Técnicas	
ii. Ferramentas	
iii. Instrumentos	
iv. Projeto de Estudo	
3. Ações	
4. Avaliações e Análises	
5. Reflexões e Aprendizados	Capítulo 5 Conclusões – Reflexões e Aprendizados da Pesquisa-Ação
a. Aprendizados	
b. Reflexões	

Na segunda ação foi utilizado outro objeto de estudo, que é um sistema de controle de refrigeração. A partir de um documento de especificação desse sistema, foi elaborado o diagrama UML Statechart de um de seus módulos. Neste caso foi realizada a engenharia avante, pois a partir do modelo UML Statechart, os modelos Simulink foram construídos.

Assim, em termos da metodologia Pesquisa-Ação, essas duas ações contribuem para a mesma finalidade de avaliar a adequação da UML Statechart para representar o modelo Simulink de uma forma mais abstrata. Além disso, elas foram executadas nessa ordem, pois a primeira ação foi um momento de aprender como abstrair o modelo Simulink. Esse conhecimento foi importante para elaborar o modelo usado na segunda ação.

Na seção seguinte é apresentada a etapa de diagnóstico da metodologia Pesquisa-Ação. Embora essa etapa aborde pontos que já foram mencionados neste

texto, esses pontos são tratados com um pouco mais de detalhe nessas seções. Essa etapa tem por objetivo descrever o problema que deu conteúdo para a realização desta pesquisa, o contexto em que o projeto está inserido e o tema desta pesquisa.

1.1 Descrição do Problema

O desenvolvimento de software está bastante relacionado a computadores pessoais, grandes sistemas de TI e aplicações de internet. Esses sistemas utilizam menos de 2% dos microprocessadores produzidos (EBERT e SALECKER, 2009). Os demais microprocessadores são utilizados no desenvolvimento de SEs. Estes sistemas são constituídos de software e hardware. O software encontrado em um SE (para evitar confusão, a sigla SE só é designada para especificar sistemas embarcados. Software Embarcado será sempre referido pelo nome completo) é denominado de software embarcado, sendo uma categoria especial de software, pois, geralmente, são aplicações críticas que precisam ser operadas de forma segura durante longos períodos de tempo (KONRAD, CHENG e CAMPBELL, 2004) (WANG, ZHOU, *et al.*, 2009).

Como descrito anteriormente, existe uma tendência em aplicar técnicas, métodos e processos tradicionais da engenharia de software para auxiliar o desenvolvimento de SE. Em geral, o desenvolvimento desses sistemas é assistido por ferramentas como Matlab/Simulink e LabVIEW. Essas ferramentas oferecem recursos que permitem auxiliar e facilitar o desenvolvimento de software embarcado a partir de uma interface interativa para implementar modelos que são conjuntos de blocos com fluxo de dados. Os modelos são utilizados para a geração automática de código que será embarcado em um SE.

As ferramentas Matlab/Simulink e LabVIEW são bastante utilizadas no desenvolvimento de sistemas automotivos e aeroespacial, pois oferecem um ambiente de desenvolvimento baseado em modelos e a simulação do sistema de forma intuitiva. Entretanto, apesar de características intuitivas, os modelos dessas ferramentas correspondem a uma abstração de baixo nível, o que dificulta a sua

compreensão. Essa dificuldade pode piorar se a pessoa que precisa compreender os modelos não for um desenvolvedor com conhecimento da ferramenta.

Assim, verificou-se que a implementação do software inicia do baixo nível utilizando ferramentas como Matlab/Simulink e LabVIEW, não sendo acompanhada de um planejamento contendo especificação e documentação do sistema. A ausência de uma documentação para guiar o desenvolvimento do software embarcado pode causar a elaboração de um diagrama ilegível, confuso e de difícil manutenção.

1.2 Contexto do Projeto

O INCT-SEC tem por objetivo elevar o nível de conhecimento, competência e qualidade no que diz respeito ao desenvolvimento de sistemas embarcados críticos (SEC). Para alcançar esse objetivo, o INCT-SEC vem agregando habilidades, competências e infraestrutura necessárias para o desenvolvimento de sistemas embarcados críticos, com ênfase para veículos autônomos móveis, de forma a capacitar a academia e a indústria brasileira no ensino, treinamento, pesquisa e desenvolvimento científico-tecnológicos em aplicações de relevância e de alto impacto econômico-social em áreas estratégicas do país. Dentre as aplicações desse tipo de sistema, é foco no contexto do INCT-SEC, aplicações na agricultura, segurança e defesa nacional, aviação e meio ambiente.

O LaPES (Laboratório de Pesquisa em Engenharia de Software), enquanto membro do projeto INCT-SEC, tem como objetivo identificar uma estratégia de desenvolvimento de sistemas embarcados explorando e estendendo as técnicas, métodos e processos existentes na Engenharia de Software para auxiliar o desenvolvimento desse tipo de sistema. O processo a ser definido deve contemplar as fases iniciais de desenvolvimento, desde a elicitação dos requisitos até a elaboração do modelo Simulink. A definição desse processo tem sido feita partindo-se do modelo Simulink e identificando-se as etapas e artefatos que devem antecedê-lo, aplicando-se, portanto, a engenharia reversa até que se chegue na definição dos requisitos. A cada etapa e artefato identificado, procede-se com a engenharia avante para confirmar as definições e decisões tomadas até aquele momento.

1.3 Tema da Pesquisa

O objetivo específico deste trabalho é identificar uma forma de representar o modelo Simulink em um nível mais alto de abstração, para transformar esse artefato em um componente a ser elaborado no processo almejado pelo LaPES. A técnica UML Statechart foi considerada, a partir da pesquisa descrita neste trabalho, uma forte candidata para representar em mais alto nível um modelo Simulink, descrevendo o comportamento, permitindo compreender as suas funcionalidades.

Assim, neste trabalho, o objetivo é explorar o uso da UML Statechart para representar o comportamento de um SE, de forma que, com base no modelo UML Statechart consiga-se elaborar um modelo em Simulink com maior legibilidade.

1.4 Organização do Trabalho

Embora a organização do trabalho já tenha sido apresentada na Tabela 1.1, em seguida descreve-se com mais detalhes os objetivos de cada capítulo.

Neste capítulo – Capítulo 1 – o objetivo foi informar o leitor de que este trabalho seguiu o método de pesquisa denominado Pesquisa-Ação, que difere de uma pesquisa tradicional, pois ao invés de ter um alvo bem definido a ser alcançado, conforme ele tenha sido idealizado no início do trabalho. Esse método tem um caráter exploratório, que procura adquirir mais conhecimento sobre um determinado assunto por meio de iterações. Além disso, este capítulo mostrou que este trabalho é parte de um projeto maior e que seu objetivo é contribuir apenas com uma parte do objetivo mais amplo. Considerando que o processo da Pesquisa-Ação é constituído de etapas, os capítulos subsequentes representam as outras etapas desse processo.

No Capítulo 2 apresenta-se uma parte da etapa da Pesquisa-Ação, denominada Planejamento, a qual corresponde à revisão bibliográfica realizada para dar suporte ao desenvolvimento do trabalho.

Nos Capítulos 3, 4 e 5 apresentam-se também partes da etapa de Planejamento, as quais correspondem às iterações que são realizadas. Essas

iterações, denominadas Ações, são compostas de várias subetapas, que descrevem todo o planejamento da ação.

Assim, no Capítulo 3 apresenta-se a ação que explorou o uso da UML Statechart para representar o modelo Simulink em um nível mais alto de abstração. O modelo UML Statechart foi elaborado por meio de uma atividade de engenharia reversa e, para verificar sua contribuição no contexto da engenharia avante, o modelo Simulink passou por uma atividade de reestruturação.

No Capítulo 4 apresenta-se outra ação realizada após a ação apresentada no Capítulo 3, a qual explora o uso da UML Statechart no contexto da engenharia avante, mas partindo-se de um documento de requisitos.

No Capítulo 5 descreve-se uma ação que foi realizada paralelamente às outras duas, que caracterizou, por meio de um *survey*, como os desenvolvedores de sistemas embarcados procedem, em geral, para desenvolver esse tipo de aplicação.

No Capítulo 6 apresenta-se a última etapa da Pesquisa-Ação realizada neste trabalho, que trata das reflexões e aprendizados, o que corresponde às conclusões deste trabalho.

Capítulo 2

REVISÃO BIBLIOGRÁFICA – PLANEJAMENTO DA PESQUISA-AÇÃO

Este capítulo corresponde à primeira parte da etapa de Planejamento do método de Pesquisa-Ação, que tem por objetivo apresentar a revisão bibliográfica realizada para este trabalho de pesquisa.

2.1 Considerações Iniciais

Seguindo a metodologia Pesquisa-Ação, neste capítulo é descrito a revisão bibliográfica realizada para esta pesquisa. São apresentadas as definições e os trabalhos relacionados no que se referem a Sistemas Embarcados (SE), às técnicas de modelagem Statechart (HAREL, 1987) e UML Statechart (*Unified Modeling Language*, 2011); Redes de Petri (MURATA, 1989); e Matlab/Simulink (Simulink® *Getting Started Guide*, 2010) e LabVIEW (*National Instruments*, 2011) (KENT, 2002).

O desenvolvimento de SE é uma atividade complexa em que diversos requisitos precisam ser levados em consideração. Durante a fase de desenvolvimento é necessário identificar informações relevantes que permitem determinar o bom funcionamento do SE. Essas informações envolvem propriedades funcionais e não funcionais e o ambiente em que o sistema será inserido, entre outros requisitos. Assim, necessita-se explorar metodologias, que são bastante utilizadas no desenvolvimento de software convencionais, para atingir uma maior qualidade no desenvolvimento de SE.

O desenvolvimento de software convencional baseia-se em modelos, MBD - *Model-Based Development* (KENT, 2002), a partir dos modelos propostos pela UML. A UML é uma coleção de modelos semiformais que permite especificar, visualizar, construir e documentar modelos de sistemas técnicos e de software. O *framework* MDA (*Model Driven Architecture*, 2011) é uma proposta que reúne diversos padrões para modelagem oferecidos pela OMG – *Object Management Group* (omgmarte, 2011).

A MBD permite que os requisitos do sistema sejam especificados a partir de um planejamento para depois iniciar a fase de codificação. Isso reduz o número de erros, falhas no projeto, evitando um retrabalho após a codificação, o que poderia gerar um maior custo no desenvolvimento do sistema.

A partir dos resultados que a MBD vem demonstrando, desenvolvedores de SE têm explorado esse paradigma para melhorar a qualidade dos seus produtos. Assim, diversas pesquisas estão sendo direcionadas na adaptação de técnicas de modelagem para o processo de desenvolvimento de SE. Dentre as técnicas existentes, a mais utilizada é o paradigma orientado a atores (LEE, NEUNDORFFER e WIRTHLIN, 2003), que envolve o uso de ferramentas de domínio específico, tais como Matlab/Simulink, LabVIEW, Ptolomy, SystemC, Verilog, entre outras.

A UML é uma das técnicas de modelagem que vem sendo bastante pesquisada e empregada no desenvolvimento de SE. Essa técnica permite representar um SE através de *Profiles*, que são extensões com estereótipos e anotações para descrever requisitos funcionais e não funcionais de um sistema. Outras duas técnicas gráficas para representar sistemas embarcados, são as técnicas Statechart (HAREL e POLITI, 1998) e Redes de Petri (MURATA, 1989).

Este capítulo está organizado da seguinte forma: Na Seção 2.2 são descritas as definições de Sistemas Embarcados, seus conceitos, caracterização e aplicações; na Seção 2.3 apresenta-se a definição de Statecharts; na Seção 2.4 é descrita a definição de Rede de Petri; na Seção 2.5 é descrito o Matlab/Simulink; na Seção 2.6 apresenta-se a Reengenharia de Software; na Seção 2.7 apresentam-se os trabalhos relacionados à Engenharia de Software e Sistemas Embarcados; na Seção 2.8 apresentam-se os trabalhos relacionados ao uso da UML; na Seção 2.9 apresentam-se os trabalhos relacionados ao uso do paradigma de Desenvolvimento Orientado a Atores; na Seção 2.10 apresentam-se os trabalhos relacionados ao uso de Matlab/Simulink e LabVIEW; na Seção 2.11 são apresentados os trabalhos

relacionados de UML Statecharts e Rede de Petri; na Seção 2.12 as considerações finais.

2.2 Sistemas Embarcados (SE)

Os avanços tecnológicos permitiram empregar SE para garantir maior qualidade nas atividades que, em geral, são executadas manualmente. Esses SE são desenvolvidos com a finalidade de executar tarefas específicas e podem ser aplicados em diversas áreas de atuação, como por exemplo, na saúde, administração, agricultura, segurança, aviação, transporte, dentre outras áreas de atuação. OS SEs são aplicados com o objetivo de automatizar e otimizar tarefas repetitivas, exigindo confiabilidade e agilidade.

A aplicação de SE abrange vasta área, desde sistemas simples, como um controle automático de temperatura de um ar-condicionado, a sistemas mais complexos, que podem envolver riscos à saúde de uma pessoa, como equipamentos que auxiliam na medicina. O crescimento das áreas de aplicação tem exigido SE mais complexos e de maior qualidade (MARWEDEL, 2006).

Em geral um SE é um dispositivo microcontrolado programado para executar um subconjunto de funcionalidades. Algumas definições englobam tanto características de software quanto de hardware de um SE. Assim, Li e Yao (2003) descrevem um SE como um sistema computacional com a integração entre hardware e software desenvolvido para executar uma funcionalidade dedicada.

De acordo com Douglass (1999), um SE controla e monitora um processo físico em um tempo determinado, operando sobre restrições mais severas que sistemas de software convencionais e com desempenho confiável por longos períodos de tempo.

Noergaard (2005) define SE como sistemas limitados em hardware e software, sendo projetados para desempenhar funcionalidades específicas, operando de forma confiável e consistente. O software embarcado é a parte programável do sistema, são linhas de código que descrevem as funcionalidades do SE. O hardware embarcado é a parte física, sendo composto por microcontroladores, sensores e atuadores.

Alguns autores acrescentam como característica, que um SE faça parte de um sistema maior (BARR, 1999). Por exemplo, carros modernos são compostos por um conjunto de SE que controlam diferentes funcionalidades como, sistema de frenagem (ABS), sistema de emissão de poluentes, informações no painel, controlador do retrovisor e controlar limpador do para-brisa. Todos estes SE estão conectados por uma rede de comunicação.

Outra característica atribuída ao SE, é que estes sistemas possuem um conjunto de tarefas e funcionalidades previamente embarcadas que não podem ser modificadas pelo usuário final. O usuário tem capacidade de selecionar as funcionalidades que deseja utilizar, porém, não tem a permissão de substituí-las.

Na próxima seção são apresentados conceitos de Sistemas de Tempo Real. Este tipo de sistema é uma subclasse de SE. Por serem aplicadas em áreas como saúde, segurança e controle, grande parte das pesquisas é destinada a sistemas de tempo real.

2.2.1 Características de Sistemas Embarcados

A partir dos conceitos definidos na seção anterior, outra característica que é bastante importante nesse tipo sistema é o tempo de resposta. A execução correta de um SE não envolve apenas os resultados lógicos computados, mas também, o tempo em que esses resultados são produzidos (STANKOVIC, 1988). O tempo de resposta é considerado tão importante quanto a resposta correta computada.

Sistemas Embarcados de Tempo Real (SETR) podem ser classificados em duas subcategorias: *Hard Real-time* e *Soft Real-time*. *Hard Real-time Systems* (HRTS) são sistemas em que a saída computada precisa estar de acordo com o tempo ou intervalo de tempo estabelecido. Caso contrário, um atraso na resposta pode acarretar em algum prejuízo ao ambiente em que está inserido ou ao usuário que esteja utilizando (STANKOVIC, 1996). Esse tipo de sistema também é conhecido como Sistema Embarcado Crítico (BOWEN e STAVRIDOU, 1993).

Soft Real-time Systems (SRTS) são sistemas que precisam ser executados em tempo determinado, porém, caso a resposta não seja computada no tempo estabelecido, não irá comprometer a integridade do sistema, ambiente ou usuário (GANSSLE, 2008). Exemplos de SRTS são, celulares, PDA, MP3 *players*, entre outros. Muitos dos SE desenvolvidos encontram-se nesta categoria.

A Figura 2.1, apresenta como Li e Yao (2003) classificam a relação de sistemas embarcados com sistemas de tempo real. Li e Yao consideram que SE e STR são categorias diferentes de sistemas computacionais e que um Sistema Embarcado de Tempo Real é considerado ser uma intersecção dessas duas categorias.

Figura 2.1 - Relação entre Sistemas Embarcados e Sistemas de Tempo Real (LI e YAO, 2003).

2.2.2 Aplicação de Sistemas Embarcados

A aplicação de SE abrange diversos níveis de complexidades que vão desde sistemas simples a sistemas altamente sofisticados, complexos e distribuídos. Na Tabela 2.1 são apresentados vários exemplos de aplicação de SE. As áreas em que são aplicadas são diversas, que incluem lazer, saúde, indústria, educação e comunicação. Muitos dos SE são aplicados para otimizar tarefas diárias, oferecendo maior segurança e confiabilidade. Outros são utilizados apenas para lazer, como aparelhos de som e filmes. A forma como os SE são aplicados é que vai definir como é classificado esse tipo de sistema, como descrito na seção anterior.

Tabela 2.1 – Exemplo de sistemas embarcados e sua área de aplicação (NOERGAARD, 2005).

Área	**Dispositivo Embarcado**
Automobilístico	Sistema de ignição
	Controle do motor
	Sistema de freios (ABS - *Antilock Breaking System)*
Dispositivos de Consumo	Televisões analógicas e digitais
	Vídeo cassete, DVD *player*, aparelho de som, etc.
	PDAs (*Personal Data Assistants*)
	Eletrodomésticos de cozinha (geladeira, torradeira, micro-ondas)
	Automóveis
	Jogos e brinquedos
	Telefones, celulares, *pagers*
	Câmeras
	Global Positioning Systems (GPS)
Controle Industrial	Robôs e sistemas de controle (manufatureiros)
Medicina	Bombas de infusão
	Equipamentos de diálise
	Dispositivos de próteses
	Monitores cardíacos
Redes de Comunicação	Roteadores
	Hubs
	Gateways
Material de Escritório	Aparelho de fax
	Aparelho de fotocópia (Xerox)
	Impressoras
	Monitores
	Scanners

2.3 Statechart

Statechart foi desenvolvido por David Harel em 1987, com o propósito de criar uma técnica gráfica e formal para representar o comportamento de Sistemas Reativos. Harel define este tipo de sistema como sendo dirigido a eventos e que precisam estar respondendo ou reagindo às ações externas e internas continuamente (HAREL e PNUELI, 1985).

Statechart é uma extensão de Máquinas de Estados Finitas (MEF) às quais foram adicionados os conceitos de hierarquia, ortogonalidade e comunicação em *broadcast*. A hierarquia permite que os estados de um sistema possam ser decompostos em subestados de forma a representar o comportamento interno; a ortogonalidade permite representar funcionalidades que são executadas concorrentemente; e a comunicação em *broadcast*, que representa a transição em estados concorrentes a partir de um mesmo evento. Em resumo, um statechart pode ser descrito como (HAREL, 1987):

Statechart = Diagrama de Estados + Hierarquia + Ortogonalidade + Comunicação em *Broadcast*.

Hierarquia: Pode ser representada como decomposição *XOR* (ou exclusivo) ou estados aninhados. Essa característica permite que um estado seja composto de outros subestados.

Ortogonalidade: Também conhecido como decomposição *AND*. Com a decomposição *AND* um estado pode ser composto por dois ou mais estados ortogonais que executam independentemente e concorrentemente

Comunicação em *Broadcast*: A comunicação em broadcast permite que estados independentes possam realizar uma transição através de um evento habilitado por outro estado. Os estados não estão concorrentes entre si, mas a ocorrência de um evento em estado implicará na transição de outros estados que aguardam pelo mesmo evento.

2.3.1 Notação dos Statecharts

Statecharts têm como componentes básicos estados com setas direcionadas, as quais representam transições (HAREL e POLITI, 1998). Estados são representados por retângulos com cantos arredondados (Figura 2.2).

Figura 2.2 - Representação Gráfica de um Estado.

A ocorrência de um evento no estado S1, habilita uma transição de S1 para um estado qualquer. Essa transição é representada por uma seta direcionada, contendo um rótulo com o evento que habilitou a transição; uma condição para a transição; e a ação que dever ser executada (Figura 2.3).

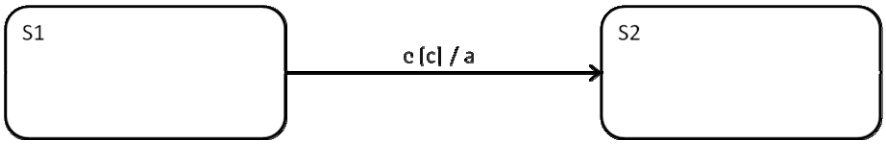


Figura 2.3 – Representação Gráfica de Transição entre Estados.

A Figura 2.3 descreve a transição entre os estados S1 e S2. A partir do estado S1, uma transição é representada por uma seta direcionada até o estado S2. A transição tem um rótulo representado por ***e[c]/a***. A letra ***e*** representa o evento que precisa ocorrer para que a transição para S2 ocorra. A letra entre colchetes ***[c]***, representa a condição para que um evento ocorra. A letra ***a*** representa a ação que deve ser tomada ao entrar no estado S2.

Na Figura 2.4 observa-se que S1 é um estado composto que contém outros dois subestados, S2 e S3. O pequeno círculo preenchido indica onde statechart inicia. Assim, nessa figura, o statechart tem início no estado S1 e, uma vez entrando nesse estado, inicia-se o estado S2, que por sua vez, através do evento α dispara uma transição para o estado S3.

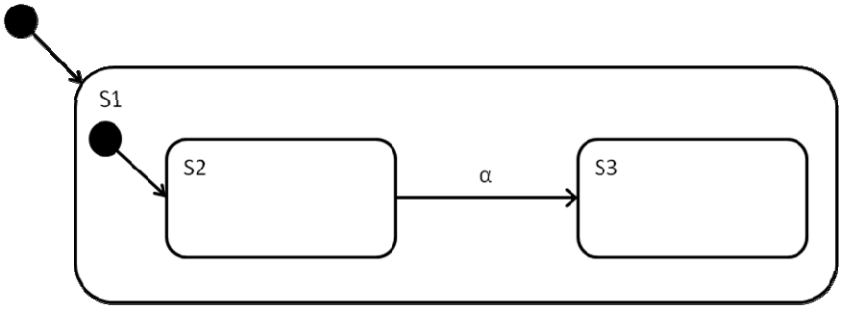


Figura 2.4 - Exemplo de Estado Composto.

A ortogonalidade, também conhecida como decomposição *AND*, representa que, estando em um estado, todos os sistemas contidos dentro deste estado devem ocorrer de forma concorrente. A notação para descrever estados concorrentes é representada pela linha tracejada (Figura 2.5).

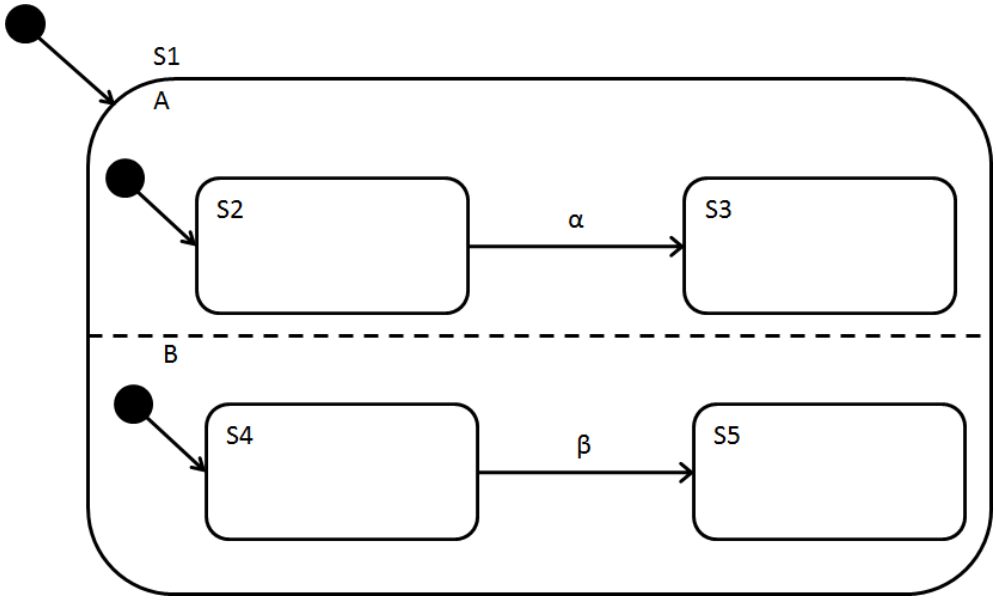


Figura 2.5 - Exemplo de Ortogonalidade.

Na Figura 2.5 o estado composto com duas regiões separado por linhas tracejadas indica que os estados S2 e S3 ocorrem concorrentemente com os estados S4 e S5. Quando entra no estado S1, iniciam-se os estados S2 e S4. Os eventos *α* e *β* disparam uma transição de S2 para S3 e de S4 para S5, respectivamente.

A comunicação *broadcast* é uma característica incorporada ao statechart, em que a ocorrência de um evento permite que duas ou mais transições ocorram. Para que isto ocorra, estas transições precisam estar rotuladas com os mesmos nomes de eventos. Na Figura 2.6 é apresentado um exemplo dessa característica.

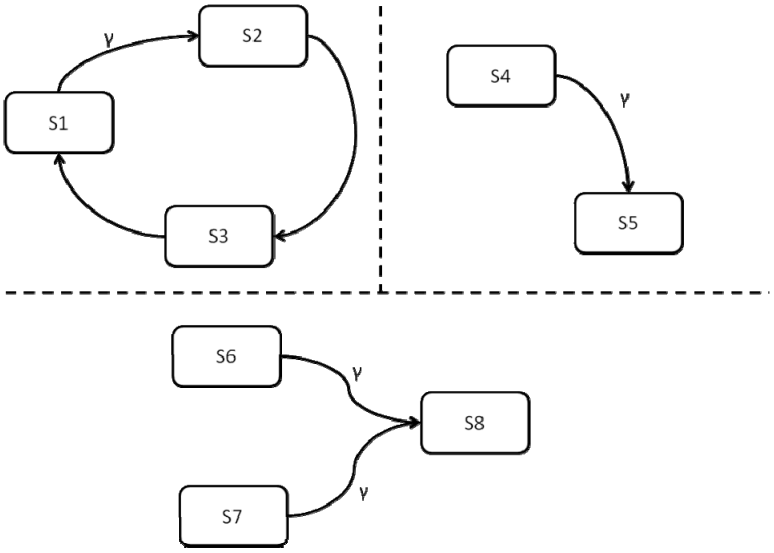


Figura 2.6 - Exemplo de Comunicação em *Broadcast*.

Na Figura 2.6 existem três sistemas executando ortogonalmente. Entretanto, o mecanismo que permite a comunicação em *broadcast* dos três sistemas é a ocorrência do mesmo evento *γ* (gama). A partir do momento que uma das transições rotuladas com símbolo *γ* ocorrer, as outras transições com o mesmo símbolo ocorrerão.

2.4 Redes de Petri

O conceito de Redes de Petri foi introduzido originalmente por Carl Adam Petri em 1962 como uma técnica gráfica e formal para representar e simular o comportamento de sistemas. Permite descrever e analisar sistemas de processamento de informações que são caracterizados como sistemas concorrentes, assíncronos, distribuídos, paralelos, não determinísticos e/ou estocástico. Sendo uma técnica visual bastante utilizada para entender o comportamento do sistema, a Rede de Petri utiliza *tokens* para simular a dinâmica e as atividades concorrentes do sistema (MURATA, 1989).

2.4.1 Notação das Redes de Petri

As Redes de Petri são um tipo de grafo direcionado contendo estados, transições e setas direcionadas. Apesar de assemelhar ao formalismo apresentado Harel (1987), a notação das Redes de Petri difere da notação dos statecharts.

Na Figura 2.7 é apresentada a notação básica dessa técnica, que é composta por lugares, sendo representados por círculos, transições, que são representadas por barras verticais e setas direcionadas, que indicam o fluxo do sistema.

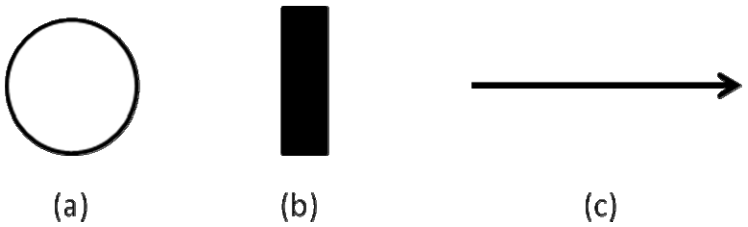


Figura 2.7 - Notações Básicas de Redes de Petri. (a) Lugar; (b) Transição; (c) Seta Direcionada.

Uma rede de Petri é representada por um grafo N direcionado, que contém uma medida e é bipartido contendo dois tipos de nós: *lugar* e *transição*. Os arcos podem ter origem do *lugar* com destino a uma *transição* ou *vice-versa*. Os arcos são rotulados com uma medida, um inteiro positivo *k* qualquer.

Um *lugar* é considerado uma marcação, ou estado, quando possui *k* marcas (círculos preenchidos). Na Figura 2.8 é apresentado um exemplo de marcação, marcas e arcos com medidas. O *lugar* com duas marcas indica uma marcação que dá condição de uma *transição*. Os arcos rotulados com o inteiro positivo 2 indica a quantidade de marcas que são consumidas ao mesmo tempo durante uma *transição*, sendo definidas como pré-condição, arco rotulado saindo do *lugar* para uma *transição*, e pós-condição, arco rotulado saindo de uma *transição* para um *lugar*.

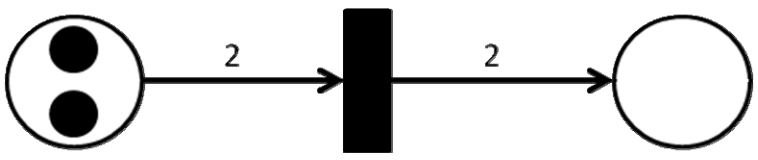


Figura 2.8 - Exemplo de Transição em Redes de Petri.

Na Figura 2.9 é apresentada a regra de transição utilizando o exemplo da reação química: $2H_2 + O_2 \rightarrow 2H_2O$.

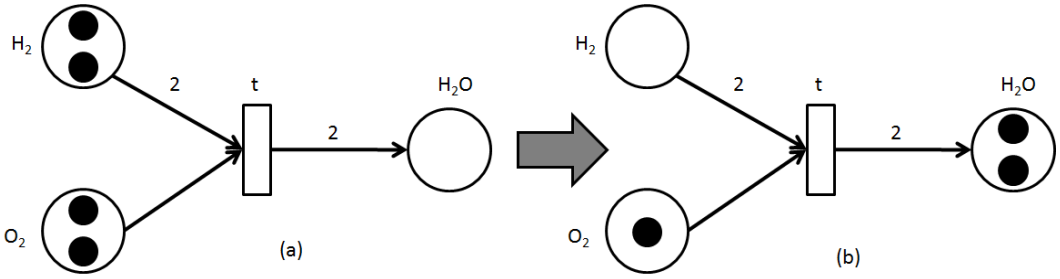


Figura 2.9 - Exemplo da Regra de Transição (MURATA, 1989).

A Figura 2.9(a) mostra que existem duas marcas para cada elemento químico: H_2 e O_2. O arco saindo do *lugar* H_2 está rotulado com o valor de 2 e o arco saindo de O_2 não contém rótulo, que por padrão é definido com o valor 1, definindo a pré-condição da *transição t*. Uma transição é definida como habilitada, se o mínimo de marcas contidas em um *lugar* é igual a pré-condição da *transição t*, ou seja, para H_2 o valor é 2 e para O_2 o valor é 1. A *transição t* vai consumir duas marcas de H_2 e uma marca de O_2. Após *t* consumir as marcas de H_2 e O_2, a marcação irá transitar de acordo com o valor rotulado no arco partido de *t* para H_2O. Como *t* consumiu apenas uma marca de O_2, essa transição não ocorre, de modo que o resultado final fica como mostra a Figura 2.9(b).

2.5 Matlab/Simulink

Matlab/Simulink (*MathWorks*, 2011) é uma ferramenta multidomínio para modelagem e simulação de SE. Essa técnica é bastante utilizada por engenheiros no desenvolvimento de SE, por oferecer um ambiente gráfico interativo com um conjunto customizado de diagramas que permite projetar, simular, implementar e testar diversos sistemas que incluem comunicação, controles, processamento de sinais, de vídeos e de imagens. A partir do conjunto de diagramas, pode-se programar uma sequência sincronizada, em que cada diagrama executa uma ação. Tais diagramas podem ser bibliotecas previamente implementadas na ferramenta, ou implementadas pelo desenvolvedor para realizar as funcionalidades desejadas.

A Figura 2.10 apresenta a interface do Simulink que contém diversas bibliotecas de componentes durante o desenvolvimento de modelos. Os componentes são separados por categorias (Blocos Comumente Utilizados, Blocos de Funções Contínuas, Blocos de Funções Discretas, Operadores Lógicos, etc). Além dessas, podem ser adicionadas bibliotecas próprias desenvolvidas para atender áreas de atuação específicas.

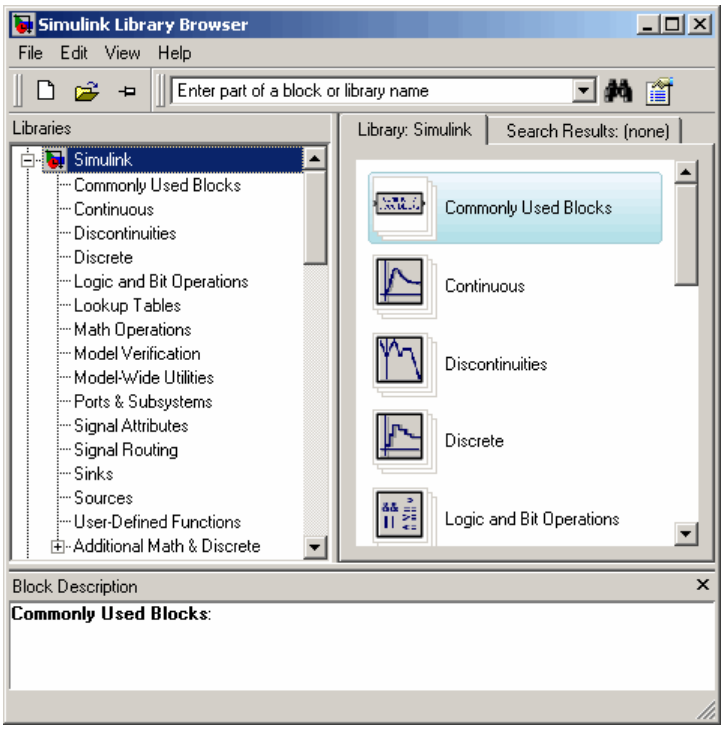


Figura 2.10 - Biblioteca de Componentes do Simulink (Simulink® *Getting Started Guide*, 2010).

Um exemplo simples de diagrama Simulink é apresentado na Figura 2.11 utilizando quatro blocos. O bloco *Sine Wave* gera uma onda senoidal a partir do tempo. O sinal que sai do bloco *Sine Wave* por conectores é bifurcado e passa para um bloco que realiza um cálculo integral. Após sair do bloco *Integrator*, o sinal é multiplexado (barra vertical preenchida) com o sinal que sai do bloco *Sine Wave*. O sinal multiplexado então é conectado a um osciloscópio.

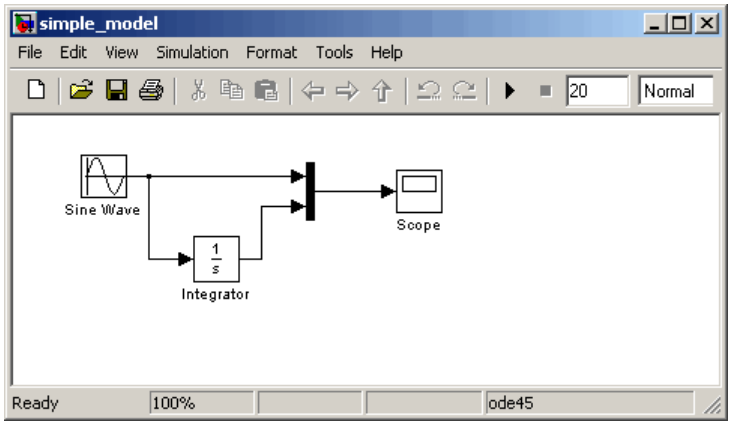


Figura 2.11 - Exemplo Simples de um Modelo Simulink (Simulink® *Getting Started Guide*, 2010).

O modelo permite estabelecer os parâmetros de configuração de cada bloco, inclusive tempo de início e término da simulação; tipo de equação utilizada; importar/exportar dados de configuração; e geração de código C/C++ para ser executado no SE.

2.6 Reengenharia de Software

A reengenharia é aplicada com o objetivo de aprimorar a estrutura e a compreensão de um sistema. A reengenharia é o processo de reimplementação de um sistema com o propósito deixá-lo mais fácil de manter, documentado, organizado, reestruturado e atualizado (SOMMERVILLE, 2003).

A realização da reengenharia segue um processo de desenvolvimento diferente da engenharia avante. A forma mais simples de distinguir a reengenharia da engenharia avante é identificando o ponto de partida do projeto. Na Figura 2.12 são apresentadas as diferenças entre engenharia avante e reengenharia. A engenharia avante parte de uma especificação, enquanto a reengenharia tem como base o software existente. Ao longo do processo de desenvolvimento, a engenharia direta projeta e implementa o novo sistema. A reengenharia realiza a compreensão e transformação com base no software já existente até atingir o software modificado, ou seja, o software que sofreu a reengenharia.

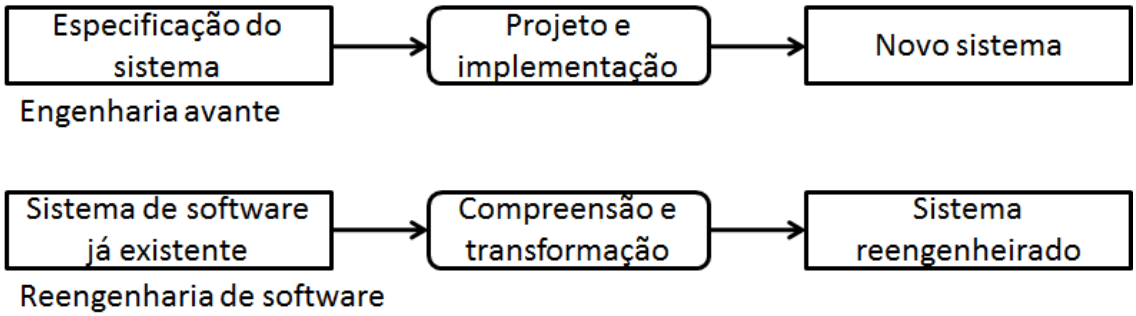


Figura 2.12 - Engenharia Direta e Reengenharia (SOMMERVILLE, 2003).

Para realizar a reengenharia de um sistema é necessária uma estratégia pragmática. Pressman (2006) apresenta um modelo cíclico (Figura 2.13) contendo um conjunto de atividades pragmáticas que fazem parte do processo de reengenharia. O processo cíclico apresentado por Pressman é enfatizado, pois se assemelha com a metodologia de Pesquisa-Ação que foi utilizada neste trabalho de pesquisa. O processo cíclico é composto por atividades que podem ser aperfeiçoadas iterativamente ao longo do processo, ou seja, cada atividade pode ser revisitada quando necessária e concluída quando desejada. Nem todas as atividades são obrigatórias de serem realizadas.

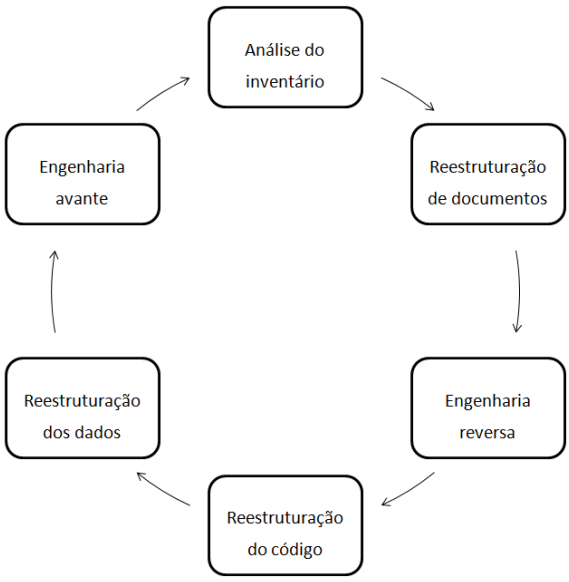


Figura 2.13 - Modelo de Processo de Reengenharia de Software (PRESSMAN, 2006)

As seguintes etapas fazem parte do processo cíclico descrito por Pressman e são apresentadas na Figura 2.13:

Análise de Inventário – A empresa possui um documento detalhado que contém as informações da aplicação que se deseja realizar reengenharia. Nem sempre existe um inventário referente à aplicação, então a estratégia é iniciar o processo através dos documentos da aplicação.

Reestruturação de documentos – Sistemas legados são caracterizados por terem pouca documentação. Nesse caso, a elaboração de nova documentação só é feita quando for essencial. Quando não há uma documentação da aplicação, realiza-se a engenharia reversa para entender a aplicação.

Engenharia reversa – A engenharia reversa de software envolve o processo de análise de um programa, buscando representá-lo em um nível mais alto de abstração. São extraídas informações referentes ao projeto de dados, arquitetura e procedimentos para a recuperação do projeto.

Reestruturação a aplicação – A partir da atividade anterior de engenharia reversa, a aplicação é reestruturada. Ao final, a aplicação resultante é revisada e testada para que não sejam encontradas anomalias em relação às funcionalidades da aplicação anterior. A documentação da aplicação é então atualizada.

Reestruturação de dados – A reestruturação de dados é caracterizada por ser uma abstração de nível mais alto em relação à reestruturação da aplicação. Geralmente, inicia-se na atividade de engenharia reversa definindo os modelos de dados, identificam-se os objetos de dados e atributos e as estruturas de dados existentes são revisadas quanto à qualidade.

Engenharia avante – Essa atividade realiza a reestruturação da aplicação a partir das informações existentes, podendo acrescentar ou melhorar a funcionalidades e o desempenho do sistema.

2.7 Trabalhos Relacionados Engenharia de Software *versus* Sistemas Embarcados

A Engenharia de Software é bastante conhecida por sua aplicação no desenvolvimento de software convencional. Entretanto, são crescentes as pesquisas relacionando a engenharia de software com o desenvolvimento de SE. Nesta seção

são apresentadas algumas tendências da relação entre essas duas áreas de pesquisa.

2.7.1 *Guest Editors' Introduction: Embedded Software – Technologies and Trends*

De acordo com Ebert e Salecker (2009), o desenvolvimento de software se baseia na ideia de que o software é facilmente reparado e rapidamente substituído. Essa ideia cria uma cultura de que o software é implementado considerando a ocorrência de falhas. Nem todos os sistemas permitem ser implementados considerando eventuais falhas que possam ser reparadas facilmente.

Assim, diversos trabalhos de pesquisa vêm explorando a engenharia de software, adaptando métodos, processos e técnicas que podem ser aplicadas durante todo o processo de desenvolvimento SE. A tendência recorrente é representar o SE a partir de um nível mais alto de abstração utilizando a UML em conjunto com ferramentas comumente utilizadas para o desenvolvimento de SE, aumentando, assim, a qualidade do produto final.

2.7.2 Embedded Software: Facts, Figures, and Future

O desenvolvimento de SE apresenta características distintas e complexas quando comparadas com o desenvolvimento de sistemas de software tradicionais. Essas características têm a sua complexidade oriunda do alto nível de interação entre as várias partes do sistema. A seguir são apresentadas as principais características que a engenharia de software deve tratar em um processo de desenvolvimento de SE (EBERT e JONES, 2009):

- Funcionalidades representadas por estados e eventos;
- Comportamento em tempo real dos eventos com ações inesperadas;
- Combinação entre sistemas de hardware e software utilizando software distribuído, computadores, sensores e atuadores;
- Ampla demanda por disponibilidade, segurança de informação e interoperabilidade;
- SE com ciclos de vida longos em que o software embarcado precisa ser executado com alto nível de confiabilidade.

Essas características de alta complexidade podem ser tratadas a partir de algumas práticas de software que garantem uma boa qualidade do produto final:

- Desenvolvimento de atividades para priorizar requisitos e rastreabilidade da qualidade;
- Desenvolvimento baseado em modelos e testes;
- Modelagem matemática para análise de confiabilidade, consumo de energia, temperatura e desempenho;
- Modelagem formal e inspeção de código;
- Análise automática de código para memória, segurança e desempenho;
- Teste automático de cobertura total.

Assim, a partir das boas práticas citadas aplicadas no processo de desenvolvimento de SE, pode-se garantir uma qualidade melhor dos produtos finais, adaptando as técnicas empregadas no desenvolvimento de sistemas de software tradicionais para o contexto de desenvolvimento de SE.

2.7.3 *Trends in Embedded Software Engineering*

Liggesmeyer e Trapp (2009) apresentam algumas vantagens que asseguram a qualidade do sistema, utilizando a abordagem de desenvolvimento baseada em modelos. Durante a fase de desenvolvimento, as propriedades funcionais e não funcionais são asseguradas através da aplicação de técnicas formais de verificação, antes da codificação do software embarcado. Após a codificação, a abordagem de testes dinâmicos pode garantir o funcionamento correto do software e do sistema. Abaixo são descritas aplicações de testes dinâmicos em diversas fases do desenvolvimento:

- *Teste de unidade* – testes aplicados a unidades específicas;
- *Teste de integração* – realiza testes de integração de unidades de software e sistemas de software com o hardware;
- *Teste de validação* – garante que todos os requisitos do sistema foram implementados de acordo com os requisitos;
- *Teste funcional* – garante a conformidade das funcionalidades do sistema.

A aplicação de testes durante a fase desenvolvimento reduzirá eventuais falhas e erros que possam surgir no software embarcado, garantindo a execução correta de forma que não comprometa a segurança de terceiros.

2.8 Trabalhos Relacionados ao uso da UML

Nesta seção são descritos trabalhos relacionados ao uso da UML para apoiar o desenvolvimento de SE. O uso da UML é uma das tendências proposta para o desenvolvimento baseado em modelos aplicado no contexto de SE.

2.8.1 An Object-Oriented Platform-Based Design Process for Embedded Real-Time Systems

No trabalho apresentado por Wehrmeister, *et al.* (2005), é proposto um processo de desenvolvimento orientado a objetos a partir de uma plataforma base. Esse processo propõe uma nova metodologia para o desenvolvimento de sistemas de tempo real. A metodologia SEEP (Sistemas Eletrônicos Embarcados Baseados em Plataforma) descreve todas as fases do processo de desenvolvimento, incluindo modelagem, análise, validação e ferramentas de síntese; visando à reutilização de componentes de software e hardware.

O estudo de caso apresentado por Wehrmeister e outros, consiste na implementação de um sistema de tempo real para a automação de uma cadeira de rodas para deficientes físicos. O principal objetivo desse sistema é o controle da movimentação e, para isso, fez-se necessário um levantamento de requisitos mais detalhado para garantir a segurança do usuário. Nesse projeto foi utilizado o *profile* da UML/RT para modelagem em conjunto com algumas API's Java para transição dos modelos em código-fonte. Outras ferramentas foram utilizadas, como a SASHIMI (ITO, CARRO e JACOBI, 2001) para análise e síntese dos dados, que geram arquivos VHDL (eda, 2011) para customizar o processador FemtoJava.

2.8.2 Object Analysis Patterns for Embedded Systems

Na proposta apresentada no trabalho de Konrad, Cheng e Campbell (2004), foi abordada uma descrição de padrões para ser utilizada durante a fase de análise de objetos, de forma que oriente os desenvolvedores a elaborar modelos conceituais, baseados em objetos para sistemas embarcados, provendo *templates* estruturais e comportamentais do sistema. Nesse trabalho, é proposto um *framework* que dá suporte aos diferentes *profiles* da UML para modelagem de sistemas embarcados. Esses *frameworks* permitem formalizar o subconjunto de diagramas e termos da UML para diferentes linguagens; modelar e visualizar os diagramas, dando suporte às tarefas de verificação, simulação e reconhecimento de falhas.

No estudo de caso do trabalho de Konrad, Cheng e Campbell, foi implementado um SE para automatizar a limpeza de um sistema de filtragem de diesel. A modelagem desse sistema foi feita a partir de um diagrama UML, que representa a estrutura e o comportamento do sistema. Foram utilizadas ferramentas em conjunto para a representação e simulação dos *templates* gerados pela UML; e propriedades específicas em LTL (*Linear-time Temporal Logic*) para tratar requisitos de tempo que não são tratados pela UML.

2.8.3 Challenges in Combining SysML and MARTE for Model-Based Design of Embedded Systems

No trabalho proposto por Espinoza, *et al.* (2009), é apresentado um comparativo entre duas ferramentas de modelagem: a SysML (*System Modeling Language*) e UML/MARTE. A ideia desse trabalho é destacar que uma única ferramenta de modelagem não é o suficiente para capturar todas as características multidisciplinares de um SE.

Cada uma das ferramentas de modelagem aborda um requisito diferente durante a modelagem. A SysML permite elaborar meios para rastrear requisitos, blocos estruturais e comportamentais do sistema, bem como, formalismos paramétricos para especificação de modelos baseados em equações analíticas. O UML/MARTE trata de requisitos relacionados a tempo e recursos e inclui uma taxonomia detalhada de padrões de software em conjunto com atributos não funcionais para análise quantitativa.

Nesse trabalho apresentado por Espinoza e outros, é proposta uma ferramenta para que essas duas técnicas de modelagem possam ser trabalhadas em conjunto de forma complementar.

2.8.4 *OO Techniques Applied to a Real-time, Embedded, Spaceborne Application*

No trabalho descrito por Murray e Shahabuddin (2006), são apresentadas as vantagens em utilizar o paradigma de Orientação a Objetos (OO) para desenvolvimento de SE. As abordagens OOA (*Object-Oriented Analysis*) e OOD (*Object-Oriented Design*) permitem melhorar a comunicação entre os engenheiros desenvolvedores do sistema.

As técnicas oferecidas pela UML permitiram representar a estrutura do sistema separado em pacotes (*Packages*). Esses pacotes são separados em:

- **Requisitos –** contêm os casos de uso do sistema;
- **Verificação –** contendo testes do sistema, análise, tarefas de inspeção, testes de ambiente e configurações;
- **Arquitetura** – descreve o sistema como um todo;
- **Detalhes de projeto –** especifica a estrutura da implementação.

Nessa proposta foi descrito que o uso da UML é muito bem aplicado para representar a estrutura do sistema, buscando demonstrar a interação entre suas diversas funcionalidades, desde requisitos quanto testes e inspeção do sistema.

2.8.5 Modeling C-Based Embedded System Using UML Design

No artigo apresentado por Wang *et al.* (2009), aborda o uso de modelos UML, que são voltados para desenvolvimentos em OO, e o desenvolvimento de SE, que são programados utilizando a linguagem estruturada C. Nesse trabalho foi proposto um mapeamento dos modelos OO para a linguagem C. O mapeamento foi aplicado em um estudo de caso de um cronômetro. O comportamento do cronômetro é representando por um caso de uso, sendo que cada uma das funcionalidades é representado por um arquivo .c e outro arquivo .h. Foi utilizado o diagrama de sequência para representar a ordem das chamadas das funções. Para os estados

de iniciar e parar o cronômetro, a representação é feita utilizando UML Statechart. O diagrama de atividade descreve como a função irá se comportar.

Apesar de ser um exemplo simples, o estudo de caso explora os diagramas oferecidos pela UML e realiza um mapeamento para linguagem C a partir dos modelos realizados para o caso específico.

2.9 Trabalhos Relacionados ao uso do Desenvolvimento Orientado a Atores (*Actor-Oriented Programming*)

O Desenvolvimento Orientado a Atores faz uso de duas abordagens, desenvolvimento baseado em plataforma (PBD – *Platform-Based Design*) e o desenvolvimento baseado em modelos (MBD – *Model-Based Development*). No trabalho apresentado por Lee, Neundorffer e Wirthlin (2003), é descrita a diferença entre essas duas abordagens. O desenvolvimento baseado em plataforma faz uso de softwares de desenvolvimento que formam uma camada de abstração de mais alto nível para o desenvolvimento de SE. As ferramentas que se enquadram nesse grupo são Matlab/Simulink (*MathWorks*, 2011), LabVIEW (*National Instruments*, 2011), SystemC (SystemC, 2011), Verilog (Verilog, 2011) e Ptolomy (Ptolomy Project, 2011). Essas ferramentas utilizam blocos interconectados que se comunicam com fluxo de dados.

O desenvolvimento baseado em modelo é uma visão de mais alto nível do que o PBD. Este é considerado de mais alto nível que a programação em código, assim como a programação em C é um nível mais alto que a programação *assembly*.

O Desenvolvimento Orientado a Atores é assim definido devido aos blocos ou componentes que são utilizados no desenvolvimento. Blocos funcionam como atores que executam alguma funcionalidade e se comunicam com outros atores. Os atores possuem uma interface bem definida que abstrai o comportamento e o estado interno, restringindo como um ator interage com o ambiente. O ator possui portas por onde a comunicação é feita e parâmetros que permitem configurar as operações que precisam ser realizadas.

A seguir, nas próximas subseções, são apresentados trabalhos que buscam relacionar o desenvolvimento orientado a atores com o paradigma de orientação a objetos.

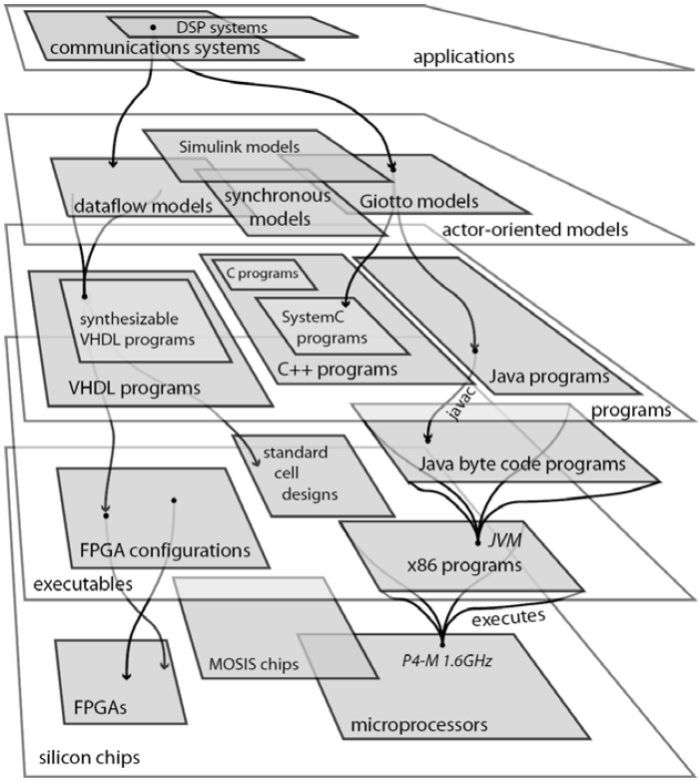


Figura 2.14- Ilustração de Tipos de Plataformas (LEE e NEUENDORFFER, 2004).

Na Figura 2.14, proposta por Lee *et al.* (2003), são apresentados os níveis de abstração das duas abordagens. A camada superior é considerada uma camada de aplicação (*applications*) em que são classificados os sistemas DSP (*Digital Signal Processor*) e de comunicação. Os modelos orientados a atores são constituídos de blocos ou meta-modelos que não sejam códigos de linguagens de alto nível. Na terceira camada estão as linguagens de alto nível como C/C++, Java e VHDL. A quarta camada é composta por códigos executáveis (*executables*). Os códigos executáveis são compostos por programas para processadores x86, JVM (*Java Virtual Machine*) e configurações de placas FPGAs. A camada mais inferior é representada por chips à base de silício (*silicon chips*), FPGAs, MOSIS e microprocessadores em geral.

2.9.1 *Classes and Inheritance in Actor-Oriented Design*

O trabalho apresentado por Lee, Liu e Neuendorffer (2009), descreve como tratar características da programação orientada a objetos. Classes, herança, interface e polimorfismo podem ser representados no desenvolvimento orientado a atores. As ferramentas que utilizam o paradigma de orientação a atores (OA) são melhores para tratar concorrência, ao contrário das linguagens OO como Java, C# e C++ que utilizam threads (apud Lee, 2006). Entretanto, assim como as linguagens orientadas a objetos, as linguagens OA buscam modularizar o desenvolvimento do software.

A adaptação dos conceitos de OO para a OA é demonstrada a partir da ferramenta Ptolomy II. Na Figura 2.15 é apresentado um exemplo simples de classe e herança. O modelo a esquerda no alto possui uma classe definida como *OndaSenoideComRuido*. Uma instância da classe foi feita com o nome *InstanciaDeOndaSenoideComRuido*. No canto esquerdo, abaixo é definida a classe *OndaSenoideComRuido* de forma hierárquica. A hierarquia é representada por um subsistema contendo um conjunto de blocos que podem executar uma funcionalidade, nesse caso uma onda senoidal com ruído. Os subsistemas permitem que esses blocos possam ser reutilizados em outra situação qualquer.

A classe *OndaSenoideComRuido* é uma subclasse de *OndaSenoide*, que é o modelo descrito a direita. Os componentes herdados são circundados com linhas tracejadas de forma a indicar a herança de componentes. A classe *OndaSenoideComRuido* estende a classe *OndaSenoide* adicionando alguns atores, conexões e portas. Os componentes que são adicionados não são marcados por linhas tracejadas, já que estes não foram herdados.

A execução do modelo vai gerar dois sinais, como mostrado no canto direito abaixo da Figura 2.15. Um dos sinais é a onda senóide sem ruído, o outro é a onda senóide com ruído. A onda senóide é gerada pela instância de *OndaSenoide*, ou seja, *InstanciaDeOndaSenoide*. A onda senóide com ruído é produzida por *InstanciaDeOndaSenoideComRuido*, que é uma instância de *OndaSenoideComRuido*. Esta última é uma subclasse de *OndaSenoide*.

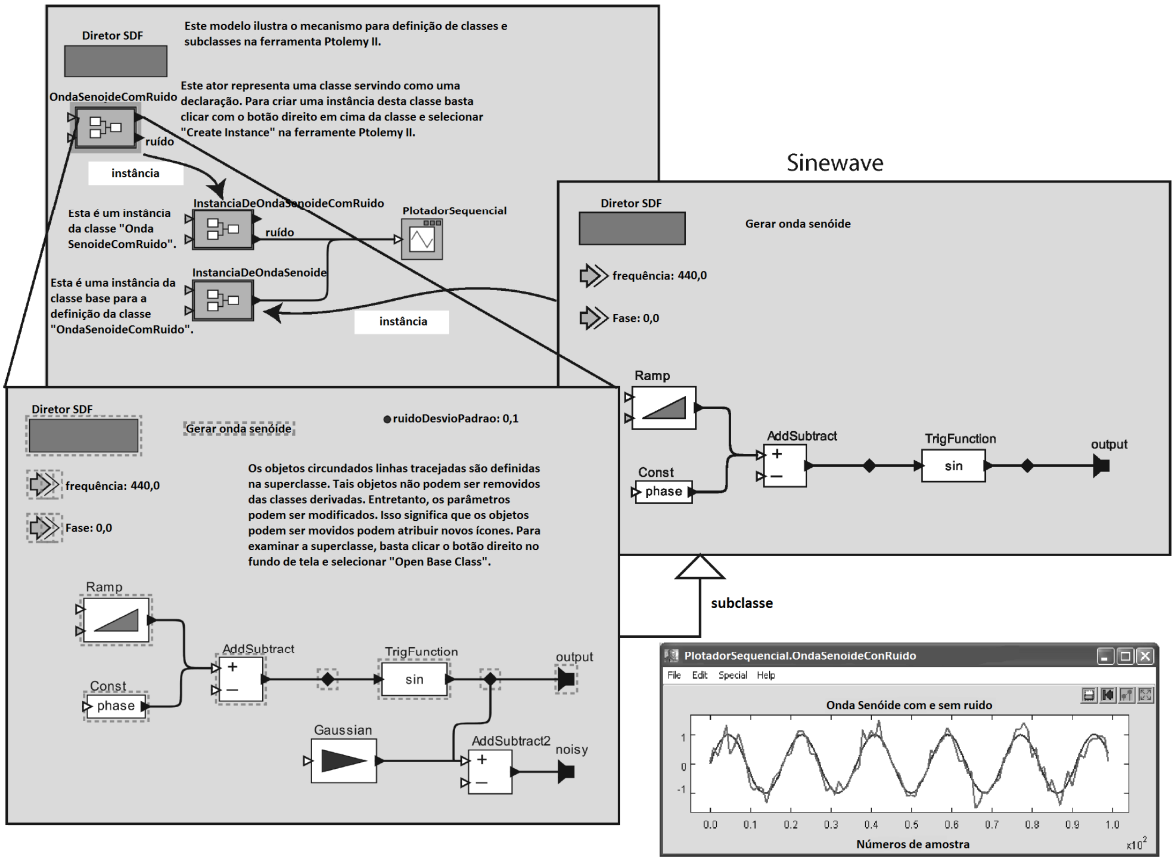


Figura 2.15 - Exemplo Simples de Uso de Classe (LEE, LIU e NEUENDORFFER, 2009).

2.9.2 *Model-Driven Development, From Object-Oriented Design to Actor-Oriented Design*

Neste trabalho de Lee (2003), foi apresentado um exemplo de polimorfismo na programação de atores. O ator *AddSubtract* da Figura 2.16 é um exemplo de que se pode implementar um componente polimórfico, no que se refere a tipos de dados. O ator adiciona ou subtrai sinais de acordo com os dados de entrada. Esses dados de entradas podem ser números (*int*, *double*, *float*), *strings* (concatenação), tipos compostos (vetores, registros, matrizes) e tipos definidos pelo usuário.

Entretanto, Lee, implementa um sistema que possa se comportar de forma polimórfica. Alguns exemplos de comportamento polimórfico:

- O ator *AddSubtract* irá adicionar os sinais apenas quando todas as entradas conectadas possuírem dados;
- O ator *AddSubtract* irá adicionar quando uma das entradas conectadas possuir um dado;

- Uma implementação para um *framework time-triggered*, os dados serão adicionados a cada click do relógio;
- Implementado para um *framework discrete-event*;

Dessa forma, Lee concluiu que uma implementação com comportamento polimórfico permite grandes benefícios para a programação OA da mesma forma que permite para a programação orientada a objetos.

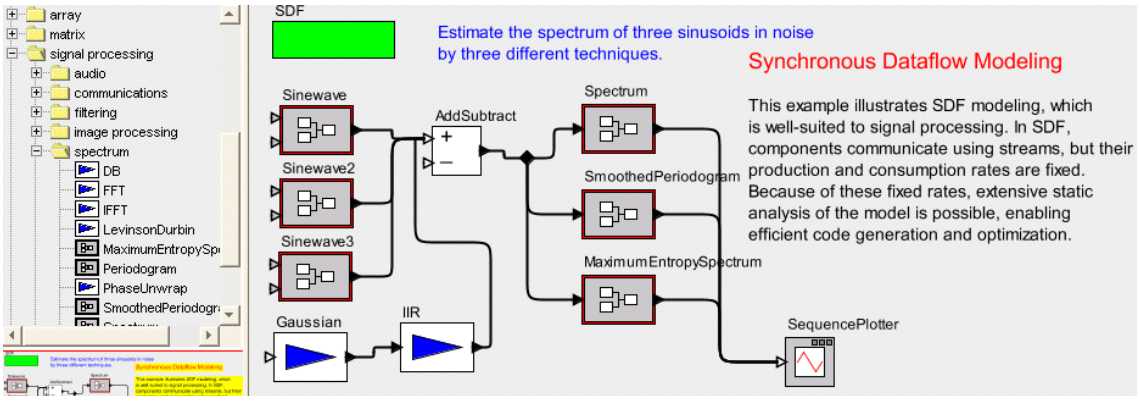


Figura 2.16 - Exemplo de Polimorfismo (LEE, 2003).

2.10 Trabalhos Relacionados ao Uso de Matlab/Simulink e LabVIEW

Nesta seção são descritos os trabalhos relacionados que utilizam as ferramentas Simulink e LabVIEW para o desenvolvimento de SE. Estas ferramentas possuem como principal característica o desenvolvimento baseado em modelo, permitindo a geração automática de código.

2.10.1 *An Integrative Approach for Embedded Software Design with UML and Simulink*

O trabalho realizado por Farkas, Neumann e Hinnerichs (2009), apresenta um estudo de caso em que MATLAB/Simulink e UML são utilizadas em conjunto para que possam ser supridas as deficiências de cada técnica de modelagem. O Simulink é bastante utilizado para modelar complexos processamentos de sinais e fluxo de dados, porém possui restrições para modelagens relacionadas à arquitetura do

sistema. Em contra partida, a UML permite a modelagem da arquitetura, comportamento e integração dos aspectos do software embarcado. Entretanto, a UML é limitada para representar aspectos específicos de um SE como, aspectos físicos, processamento de sinais e simulações. O estudo de caso apresentado por Farkas, Neumann e Hinnerichs foi conduzido na indústria automotiva para desenvolver um sistema de controle de trava de portas para carros. A integração dessas duas ferramentas permitiu o desenvolvimento do projeto a partir do paradigma de OO e orientado à arquitetura do sistema. Segundo os autores, essa abordagem garante facilitar o desenvolvimento, a depuração e o reuso dos componentes.

2.10.2 *A Rapid Prototyping Tool for Embedded, Real-Time Hierarchical Control Systems*

No trabalho descrito por Rajagopal, *et al.* (2008) é proposto um *framework* para ser utilizado integrando a ferramenta LabVIEW RT e o ambiente de hardware, permitindo um desenvolvimento de SE a partir de princípios básicos, evitando, assim, questões que comprometem o desempenho destes sistemas em tempo real de execução.

O estudo de caso conduzido por Rajagopal e outros, esboça a arquitetura de um sistema móvel que busca tratar as questões de especificações de alto e baixo nível. Um exemplo de especificação de alto nível para esse sistema é um comando para que um robô se locomova de um ponto A para um ponto B sem colidir com nenhum obstáculo. Um exemplo de especificação de baixo nível são questões referentes ao tempo de resposta de um sensor atuador. Nesse estudo de caso foi constatado que a ferramenta LabVIEW permitiu uma redução no tempo de desenvolvimento em consequência do ambiente integrado de simulação e execução. Outra questão é a linguagem de programação G, da ferramenta LabVIEW, que oferece suporte a diferentes modelos computacionais.

2.10.3 *Using Uml as a Front-End for an Efficient Simulink-Based Multithread Code Generation Targeting MPSoCs*

Uma proposta apresentada por Brisolara, *et al.* (2007), descreve um mapeamento de modelos UML para modelos Simulink, permitindo que o uso inicial de UML possa apoiar os modelos Simulink. Essa proposta também apoia a integração das duas modelagens, explorando os benefícios da UML para especificação de requisito e projeto de software, assim como a geração de código executável, a partir da especificação de alto nível. Os modelos Simulink dão suporte a diversos modelos computacionais, tais como processamento de sinais e controle.

O estudo de caso descrito por Brisolara e outros, é um sistema de controle de uma cadeira de rodas automática em que o SE foi projetado a partir do fluxo de dados da UML para gerar os modelos Simulink correspondentes. A ferramenta também permite que sejam inseridos, automaticamente, requisitos de tempo nos modelos Simulink.

2.11 UML Statecharts *versus* Redes de Petri

Nesta seção são apresentados os trabalhos que envolvem a transformação de UML Statecharts para Redes de Petri. Esses trabalhos foram estudos desenvolvidos por Trowitzsch, Zimmermann, e Hommel (2005) que buscam realizar provas formais em modelagens UML Statecharts, já que estes consideram a UML como uma técnica semiformal. Assim, são dois os métodos para verificação de desempenho de um modelo UML. O método direto é feito por meio da aplicação de métodos de análise que operam diretamente na especificação UML, e o método indireto é feito por meio da transformação do modelo UML para um modelo Redes de Petri.

2.11.1 *Towards Quantitative Analysis of Real-Time UML Using Stochastic Petri Nets*

Nesse primeiro trabalho apresentado por Trowitzsch, Zimmermann e Hommel (2005), são descritas as regras de transformação de elementos básicos da UML Statechart. Neste caso envolvem a transformação de um estado para redes de Petri e a transição de um estado para outro.

Todo estado, quando iniciado, consome um determinado tempo, mesmo que desprezível. Dentro do estado, algumas vezes, atividades são executadas antes que um evento provoque a saída do estado. Toda a execução, que vai desde a entrada do estado, executando as atividades internas até o momento do disparo de uma ação, que realizará a saída do estado, precisa ser representada para que o modelo em Rede de Petri possa realizar a simulação real do comportamento do sistema. A Figura 2.17 descreve como é feita uma transformação de UML Statechart para Redes de Petri. As atividades internas de um estado, ou seja, *entry*, *do* e *exit*, precisam ser representadas, já que estas consomem algum tempo para serem executadas.

O tipo de Rede de Petri utilizado é a definida como Estocástica. Esse tipo de Rede de Petri permite definir transições em que o tempo de execução pode ser imediato (retângulo preto estreito), determinístico (retângulo preto largo) e não determinístico (retângulo branco).

As transições de um estado para outro também podem consumir algum tempo, por isso na representação do modelo UML foi utilizado o *profile* SPT (*Schedulability, Performance and Time*). O *profile* SPT possui estereótipos, *annotations* e *tags*, que permitem identificar as propriedades não funcionais de um sistema. Então, as transições em Redes de Petri contêm um valor identificando quanto tempo uma atividade ou transição de estado levaria para ser executada.

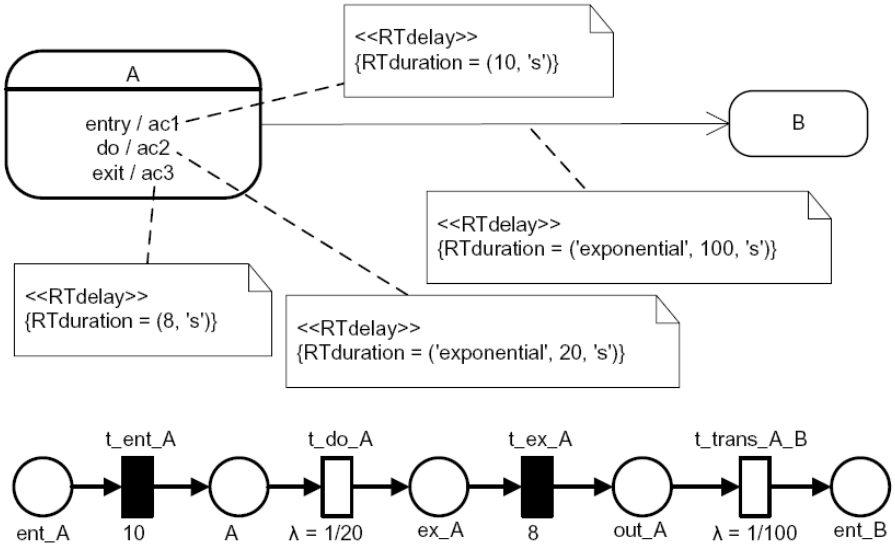


Figura 2.17 - Transformação Simples de UML Statechart para Redes de Petri (TROWITZSCH, ZIMMERMANN e HOMMEL, 2005).

A Figura 2.17 descreve como é a representação de um estado da UML Statechart em Redes de Petri. Em Redes de Petri, os círculos representam posições ou estados estáticos e os retângulos representam transições ou estados dinâmicos. Então, descrevendo o modelo gerado em Redes de Petri, o círculo rotulado *ent_A* descreve a entrada no estado A. Em seguida, uma transição é feita com um tempo de 10 segundos, rotulado de *t_ent_A*. Assim, resumindo, os círculos estão representando o estado atual: *ex_A* descreve a situação de saída do estado A; *out_A* descreve a situação fora do estado A; e *ent_B* descreve a situação de entrada do estado B. Os retângulos descrevem as transições entre as posições (círculos): *t_do_A* descreve a situação em que a atividade *do* está sendo executada e transitando para a atividade *exit*; *t_ex_A* descreve a situação em que a atividade *exit* está sendo executada, ou seja, a última atividade antes de sair do estado A; e *t_trans_A_B* descreve a transição do estado A para o estado B.

2.11.2 *Real-Time UML State Machines: An Analysis Approach*

Neste outro trabalho apresentado por Trowitzsch e Zimmermann (2005), as transformações são estendidas para os pseudo-estados. Estes são representados pelos estados iniciais (círculo preto preenchido) e finais (círculo preto preenchido envolto por um círculo maior). Entretanto, a UML Statechart adicionou algumas

anotações como complemento para melhor representar a Orientação a Objetos, são eles: *fork*, *join*, *choice* e *junction*. Estes quatro últimos pseudo-estados são as notações passivas de transformações nesse trabalho.

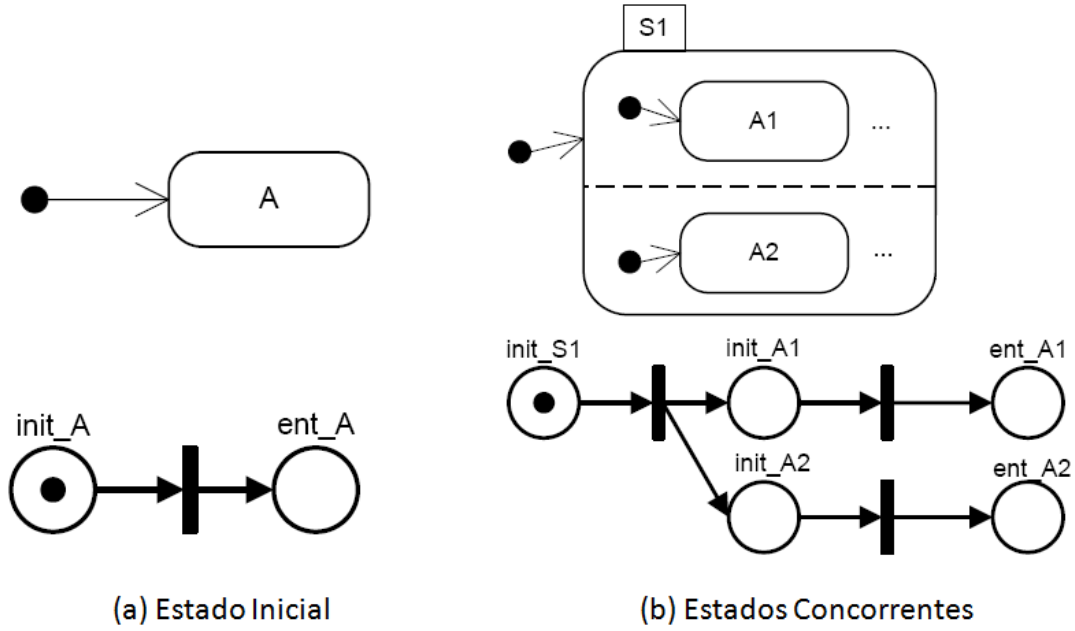


Figura 2.18 - Exemplo de Transformação de Estado Inicial (a) e Estados Concorrentes (b) (TROWITZSCH e ZIMMERMANN, 2005).

Na Figura 2.18 é apresentado um exemplo simples de estado inicial representado pela Figura 2.18 (a) e a representação de estados concorrentes representado pela Figura 2.18 (b). A concorrência é uma das principais características de Statechart e a sua representação precisa descrever cada etapa, que vai desde o estado inicial até a entrada dos estados concorrentes. Então, o estado inicial é equivalente a posição *init_S1* em Redes de Petri. Logo em seguida, existe a transição para os estados iniciais dentro de cada estado concorrente. Estando nas posições *init_A1* e *init_A2*, que representam os estados iniciais dos subestados concorrentes A1 e A2, é feita a transição para as posições *ent_A1* e *ent_A2*.

2.11.3 *Using UML State Machines and Petri Nets for the Quantitative Investigation of ETCS*

Este trabalho apresentado por Trowitzsch e Zimmermann (2006), é uma continuação dos trabalhos anteriores de transformação da UML Statechart para

redes de Petri. Aqui são apresentadas as transformações referentes à comunicação *broadcast* da UML Statechart.

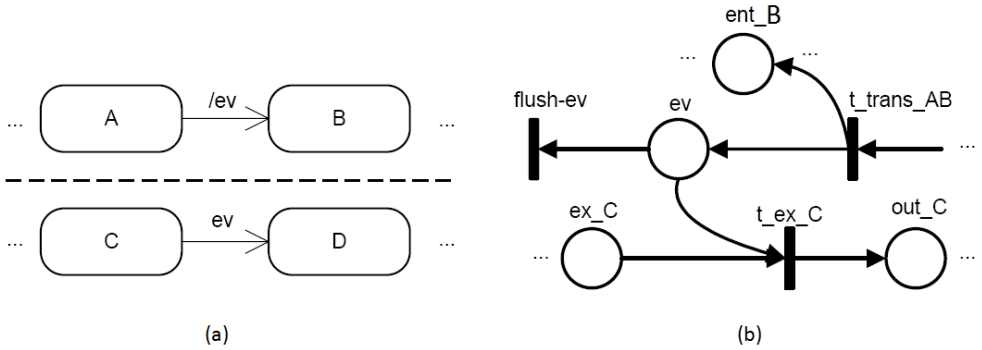


Figura 2.19 - Exemplo de Transformação de Comunicação *Broadcast* (TROWITZSCH e ZIMMERMANN, 2006).

A Figura 2.19 descreve como seria representada a situação em que uma ação, que ocorre durante uma transição entre os estados A e B, provoca um evento entre os estados C e D. Depois que efetuada a transformação para Redes de Petri, a Figura 2.19 (b) apresenta uma transição *t_trans_AB* que irá ocorrer para a posição *ent_B* e para a posição *ev*, que indica ação gerada pela transição entre os estados A e B. A posição *ex_C* para a transição *t_ex_C* só irá acontecer caso a ação *ev* tenha sido gerada durante a transição de A para B, ou seja, caso esta contenha um *token*. Pelo contrário, se a posição *ev* contenha um *token* e a posição *ex_C* não o contenha, o *token* contido na posição *ev* é eliminado pela transição *flush-ev*.

2.12 Considerações Finais

Neste capítulo foi apresentada a primeira parte do Planejamento da etapa de Pesquisa-Ação, descrevendo a revisão bibliográfica realizada para conduzir este estudo. O levantamento bibliográfico realizado apresentou os conceitos e trabalhos relacionados das técnicas, ferramentas e área de conhecimento para dar embasamento ao estudo.

Com base nos trabalhos apresentados referentes à aplicação da engenharia de software para auxiliar o desenvolvimento de sistemas embarcados, foram apresentadas características comuns no desenvolvimento de um SE e quais práticas

de desenvolvimento de software podem ser adotadas para garantir a qualidade do produto final. Entre as práticas apresentadas, a elaboração de modelos para auxiliar o desenvolvimento de SE tem sido bastante investigada no meio acadêmico. O desenvolvimento baseado em modelos é uma abordagem que é muito bem empregada no desenvolvimento de software de propósito geral (BOOCH, RUMBAUGH e JACOBSON, 1999). A aplicação desta abordagem para o desenvolvimento de SE, exigiu uma adaptação das técnicas provenientes da UML.

Diversos trabalhos foram publicados utilizando *profiles* da UML para representar SE. Entretanto, através do levantamento bibliográfico conduzido neste trabalho, foi verificado que as técnicas oferecidas pela UML não são suficientes para representar algumas características de um SE, tais como, fluxo de sinais de dados, sincronismo, assincronismo e simulações dinâmicas. Assim, a proposta aqui abordada é o uso conjunto da UML com ferramentas que são geralmente empregadas para o desenvolvimento de SE.

O desenvolvimento de SE é comumente assistido por ferramentas que são classificadas como desenvolvimento orientado a atores, por fazerem uso de modelos compostos por blocos com fluxo de dados. O estudo conduzido neste trabalho investigou o uso da técnica UML Statechart em conjunto com a ferramenta Simulink.

A técnica Statechart foi a base do estudo deste trabalho com intuito de ser aplicada como complemento no desenvolvimento de modelo Simulink. O Statechart, no contexto deste trabalho, é aplicado com o intuito de identificar as funcionalidades e representar o comportamento do modelo Simulink.

A ferramenta Simulink é bastante utilizada para desenvolvimento de SE por engenheiros, devido a sua interface gráfica e geração automática de código embarcado. Contudo, o desenvolvimento de SE utilizando essa ferramenta nem sempre é documentado. Em consequência dessa necessidade, decidiu-se realizar um estudo combinando a técnica statechart e a ferramenta Simulink.

Assim, por meio da metodologia Pesquisa-Ação, foram conduzidas três iterações de ação relacionando a técnica UML Statechart e a ferramenta Simulink para comporem um processo de engenharia avante para desenvolvimento de SE. A UML Statechart e a ferramenta Simulink são avaliadas em duas iterações inicias de forma que a primeira é conduzida para investigar a viabilidade de usar a técnica UML Statechart através de um processo de engenharia reversa e reengenharia. A segunda iteração é uma ação para complementar os resultados obtidos por meio da

primeira iteração. A terceira iteração diz respeito à aplicação de um *survey* entre os desenvolvedores de SE. O *survey* tem objetivo de investigar a cultura de desenvolvimento e se nesta cultura é aplicada alguma técnica para documentar e auxiliar o processo de elaboração de um SE. Os Capítulos 3, 4 e 5 apresentam como foram conduzidas as iterações a partir da ação da mão robótica, do sistema de controle de refrigeração e do *survey*, respectivamente.

Capítulo 3

MÃO ROBÓTICA – PLANEJAMENTO DA PESQUISA-AÇÃO – AÇÃO 1

Neste capítulo é descrita a segunda parte da etapa de Planejamento, no que se refere à Definição da Ação; as etapas de Ação; e Avaliações e Análises da metodologia Pesquisa-Ação, que descrevem a pesquisa realizada tendo a aplicação da Mão Robótica como objeto de estudo.

3.1 Considerações Iniciais

Com base no contexto apresentado no Capítulo 1, descrevendo as três iterações de ação conduzida neste trabalho, o objetivo deste capítulo é apresentar a primeira iteração de ação da mão robótica. O escopo deste trabalho, que permite dar o foco exato a ação da metodologia Pesquisa-Ação, equivale ao primeiro passo de construção do processo de engenharia avante para desenvolvimento de SE que está sendo elaborado pelo LaPES. Esta primeira ação corresponde a identificar uma forma de representar a informação contida em um modelo Simulink em um nível mais alto de abstração. Essa representação deveria ser feita utilizando uma técnica capaz de retratar os requisitos contidos no modelo Simulink, de tal maneira que, quando utilizada no processo de engenharia avante, viesse a contribuir para elaborar um modelo Simulink mais legível, mais estruturado e mais organizado do que aqueles encontrados na prática. Lembrando que esses modelos por serem, em geral, o único artefato construído pelo desenvolvedor de SEs, são construídos iterativamente e não apresenta essas propriedades mencionadas.

A partir da revisão bibliográfica, apresentada no Capítulo 2, referente à primeira etapa de planejamento da Pesquisa-Ação, identificou-se que uma possível técnica que poderia contribuir para a representação mais alto nível do modelo Simulink seria a UML Statechart. Partindo dessa premissa, o foco da ação deste trabalho foi definido e o planejamento desta etapa foi especificado conforme apresentado neste capítulo.

Assim, na Seção 3.2 descreve-se a Definição da Ação, a partir do qual são estabelecidos os meios para conduzir a pesquisa. Na Seção 3.3 é descrita a Ação, apresentando a aplicação selecionada como objeto de estudo e a descrição da ação realizada. A Seção 3.4 apresenta as Avaliações e Análises do trabalho e, por fim, na Seção 3.5, as Considerações Finais.

3.2 Definição da Ação

Esta parte do processo de Pesquisa-Ação é fundamental para organizar todo o trabalho subsequente. Nela são descritos os objetivos, as questões de pesquisa, os resultados esperados, as hipóteses e as definições operacionais, os quais estão apresentados a seguir.

3.2.1 Foco da Ação

O foco da ação tem por objetivo caracterizar e delinear o escopo da ação definindo o objetivo, as questões de pesquisa e os resultados esperados, com base na aplicação da mão robótica.

Objetivo:

O foco da ação realizada neste trabalho teve como objetivo explorar a adequação da técnica UML Statechart para representar as funcionalidades de um modelo Simulink em um nível mais alto de abstração.

Partindo-se do princípio de que as técnicas Simulink e UML Statechart têm propósitos diferentes, era esperado que a UML Statechart não fosse capaz de representar toda informação contida no Simulink. Assim, a intenção foi representar o

máximo possível no que diz respeito às funcionalidades, tempos de execução e comportamento da aplicação modelada em Simulink.

Questões:

Com base nos objetivos, estabelecem-se as questões, que definem com maior precisão o que se deseja investigar. No caso deste trabalho as questões podem ser escritas da seguinte forma:

Questão 1: A técnica UML Statechart é uma alternativa viável para representar um modelo Simulink em um nível mais alto de abstração?

Questão 2: A técnica UML Statechart representa todos os detalhes contidos no modelo Simulink?

Questão 3: A representação do modelo Simulink por meio da UML Statechart contribui para uma reengenharia desse modelo, com o objetivo de organizá-lo ou melhorar o seu entendimento?

Resultados:

Resultados esperados para a questão 1:

R.1.1 – Representar o comportamento da aplicação modelada no Simulink em um nível mais alto de abstração.

R.1.2 – Representar as funcionalidades da aplicação modelada no Simulink em um nível mais alto de abstração.

R.1.3 – Representar características de hierarquia e concorrência da aplicação modelada no Simulink.

R.1.4 – Representar requisitos não funcionais da aplicação modelada no Simulink

Resultados esperados para a questão 2:

R.2.1 – Identificar se existem informações do modelo Simulink que não são representadas na técnica UML Statechart.

Resultados esperados para a questão 3:

R.3.1 – Permitir criar um modelo Simulink mais organizado e legível a partir do modelo UML Statechart.

R.3.2 – Facilitar atividades de manutenção do modelo Simulink decorrente de sua reengenharia.
R.3.3 – Proporcionar uma forma de documentar o modelo Simulink.

3.2.2 Hipóteses

Considerando o objetivo deste trabalho, foram levantadas as questões que se esperam responder no decorrer da pesquisa. A partir das questões, foram elaborados os resultados esperados com intuito de conduzir o estudo sem desviar do objetivo. As hipóteses são formuladas combinando as questões levantadas e os resultados esperados, como estão apresentadas a seguir:

- A técnica UML Statechart é uma alternativa para representar de forma mais alto nível um modelo Simulink, identificando os requisitos funcionais e não funcionais;
- A técnica UML Statechart permite organizar o modelo Simulink, representando características como hierarquia e concorrência das funcionalidades;
- A técnica UML Statechart permite representar todas as características do modelo Simulink a partir de artefatos contidos na própria técnica.

3.2.3 Definições Operacionais

As definições operacionais descrevem as técnicas, instrumentos e ferramentas utilizadas para conduzir esta ação do trabalho.

i. **Instrumentos –** Nesta ação do trabalho não foi utilizada nenhuma forma de instrumentação.

ii. **Técnicas –** Nesta ação do trabalho foram utilizadas as seguintes técnicas:
    a. **UML Statechart –** Técnica para modelar sistemas reativos. Permite representar hierarquia, concorrência e comunicação broadcast.

iii. **Ferramentas**

a. **MagicDraw –** ferramenta para modelagem de sistemas, que facilita a análise e desenvolvimento de sistemas orientados a objetos e banco de dados (*MagicDraw*[1], 2011);
b. **Matlab/Simulink –** ferramenta baseada em modelos para desenvolvimento de SEs (*MathWorks*[2], 2011);
c. **Understand –** ferramenta de análise estática para manutenção, medição e análise do código fonte (*Scitools*[3], 2011);

3.3 Descrição da Ação

Nesta seção as ações descrevem todas as etapas realizadas envolvendo a aplicação da mão robótica como objeto de estudo, apresentando o contexto da aplicação e o processo da ação realizada.

3.3.1 Descrição da Aplicação Mão Robótica

A mão Kanguera (BENANTE, PEDRO, *et al.*, 2007) é uma mão antropomórfica mecânica controlada por cabos, sendo composta por cinco dedos híbridos independentes. A ideia de dedos híbridos está relacionada aos motores que atuam nas juntas dos dedos. Cada dedo possui quatro juntas que são controladas por quatro motores. Estes motores foram implementados utilizando a ferramenta Simulink, sendo que três deles são motores simulados, chamados de motores virtuais, e um motor que recebe os dados de um motor físico, chamado de motor real.

A tarefa de manipulação de objetos por meio de mãos robóticas é um dos maiores desafios na área da robótica (BICCHI e KUMAR, 2001). O desenvolvimento de uma manipulação habilidosa envolve várias áreas de conhecimento da ciência que precisam ser trabalhadas em paralelo.

[1] MagicDraw – versão demo, licença de 30 dias.

[2] Matlab/Simulink – versão gratuita para estudante.

[3] Scitools/Undestand – versão demo, licença de 30 dias.

A mão robótica antropomórfica é um projeto mecânico complexo que permite níveis de liberdade a cada dedo (*Degree Of Freedom* - DOF). Os DOF's significam que cada dedo possui trajetórias diferentes e atuação independente, fazendo com que a mão robótica alcance uma alta capacidade de manipulação.

Um modelo Simulink foi desenvolvido para executar a tarefa de manipulação de um lápis que exige o movimento independente de apenas três dedos (CAURIN e PEDRO, 2009). A Figura 3.1 ilustra uma posição inicial (a) e final (b) para que a mão realize a manipulação do lápis, as quais são previamente fornecidas como referência para o controle do motor. Durante a tarefa de manipulação, o movimento de cada articulação precisa ser combinado com os torques e forças apropriados de forma síncrona para a execução dos movimentos desejados do objeto.

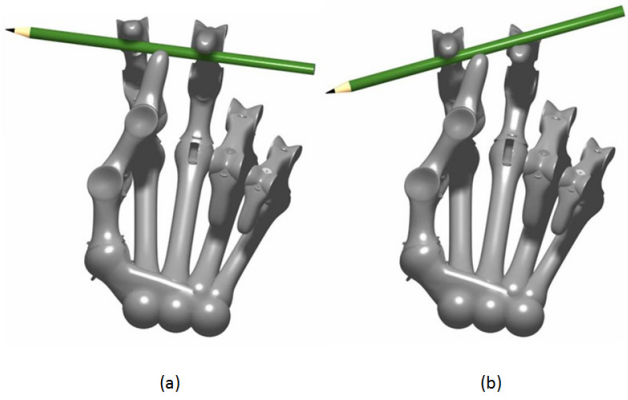


Figura 3.1 - Tarefa de Manipulação do Lápis: (a) Posição Inicial; (b) Posição Final (CAURIN, ALBUQUERQUE e MIRANDOLA, 2004).

3.3.2 Descrição da Ação Realizada

No contexto de SEs, em geral, os desenvolvedores iniciam o desenvolvimento a partir de um nível baixo de abstração, utilizando ferramentas de modelagem como Simulink e LabVIEW. Como dito anteriormente, o objetivo deste trabalho é contribuir com uma parte da definição de um processo de engenharia reversa e avante para os SEs, tendo o modelo Simulink como o artefato de mais baixo nível, anteriormente à geração de código. Assim, nessa primeira ação, a intenção é verificar se a UML Statechart é adequada para compor uma representação mais alto nível do modelo Simulink.

O estudo teve início a partir de um modelo Simulink que é parte de uma tarefa de manipulação de objetos pela mão robótica. No modelo estão representadas as

funcionalidades para a execução do movimento de fechar um dedo da mão. Durante o desenvolvimento inicial do modelo Simulink não foram elaborados modelos de mais alto nível ou qualquer documento para especificação do sistema que pudessem auxiliar a fase de implementação. O modelo Simulink original está representado na Figura 3.2, tendo sido elaborado iterativamente à medida que o engenheiro desenvolvedor simulava e testava o sistema. Assim, o modelo foi evoluindo à medida que novos requisitos foram surgindo e defeitos corrigidos. Com base nesse modelo, as etapas dessa ação podem ser representadas por meio da Figura 3.3, que retrata a atividade de engenharia reversa, construindo o modelo UML Statechart, e a atividade de reengenharia, que retrata a reengenharia do modelo criando um novo Simulink.

Para realizar a reengenharia do modelo Simulink, a primeira tentativa aplicada foi reorganizar o modelo, apenas reestruturando o diagrama, de forma a representá-lo de um modo mais legível. Entretanto, os resultados não ajudaram no entendimento do sistema. Após essa primeira tentativa, requisitou-se a presença do engenheiro desenvolvedor do modelo original, para auxiliar na extração dos requisitos.

Na segunda tentativa, aplicaram-se os passos da Figura 3.3, sendo que durante a etapa de Extração da Informação o objetivo foi identificar tarefas concorrentes e o conjunto de requisitos funcionais e não funcionais. Ressalta-se que, mesmo para o próprio desenvolvedor dessa aplicação, o entendimento do modelo não foi uma tarefa trivial em virtude do tempo decorrido desde que a aplicação foi desenvolvida. Durante essa etapa, à medida, que se buscava o entendimento do sistema, o engenheiro desenvolvedor adotou a estratégia de identificar conjuntos semelhantes de subsistemas no modelo Simulink, os quais deram subsídios para a definição dos estados no modelo UML Statecharts, conforme ilustrado na Figura 3.4. No caso desse exemplo, estes subsistemas representam as articulações do dedo. O movimento do dedo é controlado por quatro articulações, cada uma correspondendo a um motor. As cores especificadas na Figura 3.4 são apenas para distinguir atividades diferentes e que cada uma é representada por um estado distinto.

Então, para se realizar o processo de engenharia reversa foram utilizados os conhecimentos técnicos do engenheiro desenvolvedor com relação ao Simulink e a equipe de engenharia de software com relação à técnica UML Statechart. Uma

primeira diretriz proposta pela equipe de engenharia de software, foi identificar quais funcionalidades ocorriam concorrentemente. Assim, o engenheiro desenvolvedor identificou quatro conjuntos de blocos que executavam a movimentação da articulação de um dedo. A proposta de iniciar a engenharia reversa do modelo Simulink para o modelo UML Statechart identificando atividades paralelas pode ser uma boa abordagem para realizar o mapeamento. Entretanto, caso o modelo Simulink não apresente nenhuma atividade concorrente, pode-se identificar uma atividade inicial básica ou uma atividade de escolha do desenvolvedor.

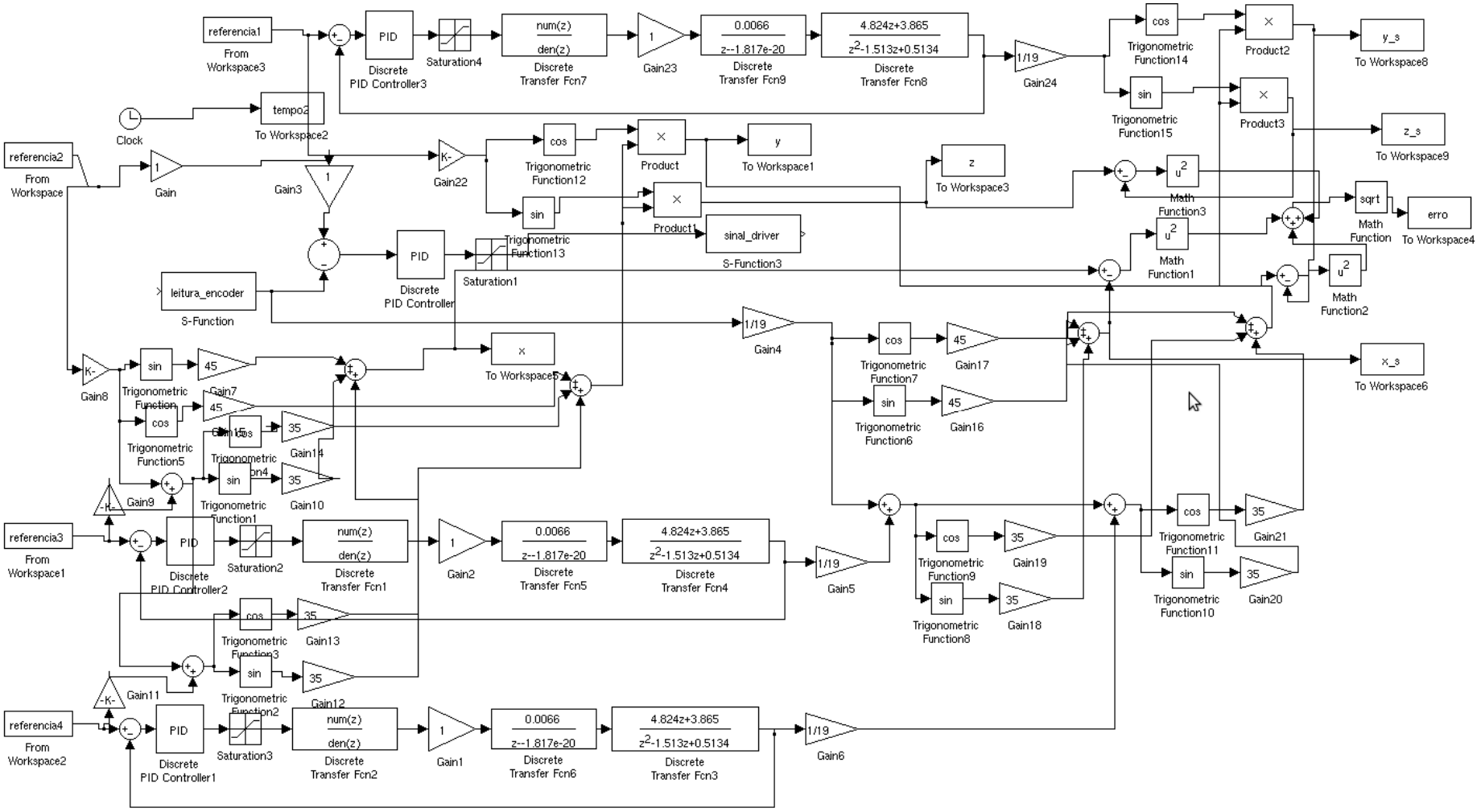


Figura 3.2 - Modelo Simulink Original para a Mão Kanguera (CAURIN e PEDRO, 2009).

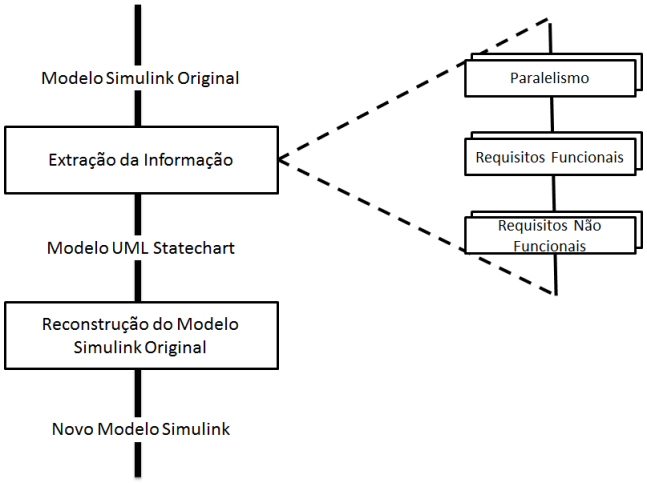


Figura 3.3 - Processo de Reengenharia do Modelo Simulink. Adaptado de Pressman (2006).

A articulação do motor é representada na UML Statechart por um superestado denominado *Articulation Motor One* (Figura 3.5), composto por dois subestados – *Controlling Voltage* e *Moving Articulation* – que executam em quatro ciclos de cinco milisegundos. Essa restrição de tempo é tratada como um requisito não funcional na Figura 3.5, em que é identificada como uma condição de *timeout* para a transição entre os estados *Articulation Motor* e *Calculating Direct Cinematic*. As quatro articulações (*Articulation Motor One*, *Two*, *Three* e *Four*), são representadas como estados paralelos no modelo UML Statechart, pois a execução dessas articulações precisa ocorrer simultaneamente. Uma vez que esses estados foram identificados, bem como a condição de paralelismo existente entre eles, foi possível projetá-los como sendo subestados de um estado ortogonal *Moving Finger* (Figura 3.5). A partir da identificação desse contexto existente na aplicação da mão robótica, o restante dos estados foi identificado de forma natural. Observe, na Figura 3.5, que após o término de quatro ciclos de cinco milissegundos das quatro articulações, o *timeout* dispara uma transição para o estado *Calculating Direct Cinematic*. Nesse estado são executados cálculos para correção da trajetória do dedo.

Além dos estados mencionados, no diagrama existem dois outros estados que representam tarefas específicas de simulação e plano de trajetória (Figura 3.5). O estado *Planning Trajectory* obtém a informação enviada pelo estado *Calculating Direct Cinematic* para identificar a trajetória executada.

Assim, é verificada se a trajetória desejada foi alcançada. Caso tenha sido, ocorre uma transição para fora do superestado *Finger*, caso contrário, a trajetória é recalculada e enviada para o estado *Calculating Inverse Cinematic*. Este último estado irá calcular o movimento de cada articulação que é composto por dados de velocidade, força e torque apropriados e enviados para o superestado *Moving Finger*.

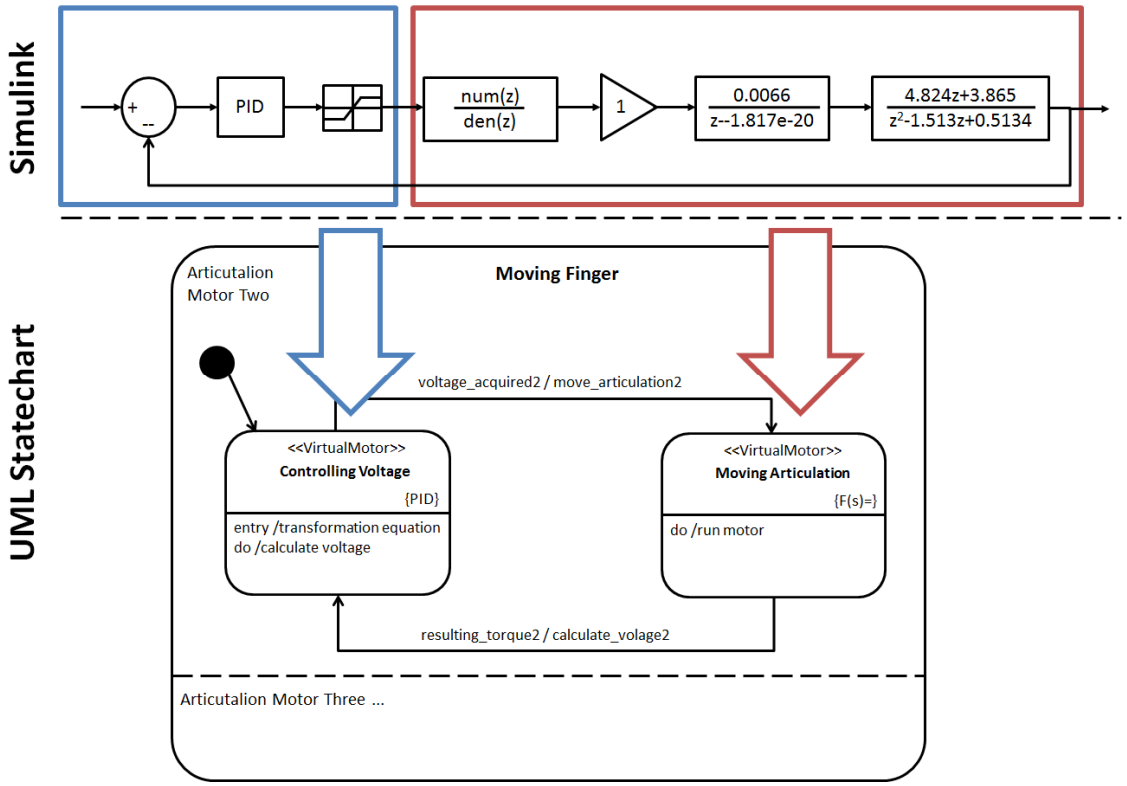


Figura 3.4 - Estados Identificados a partir do Modelo Simulink Original.

Assim, por meio de uma atividade iterativa e interativa entre os integrantes do grupo de engenharia de software envolvidos no projeto e o engenheiro desenvolvedor do modelo Simulink, foi construído o modelo UML Statecharts apresentado na Figura 3.5.

Após finalizar o modelo UML Statechart, ele foi entregue ao engenheiro desenvolvedor do modelo Simulink sistema, ao qual foi solicitada a tarefa de refazê-lo. A nova modelagem do Simulink é ilustrada na Figura 3.6, cuja melhora é notória, tornando-se um modelo mais claro, o que pode facilitar atividades de manutenção.

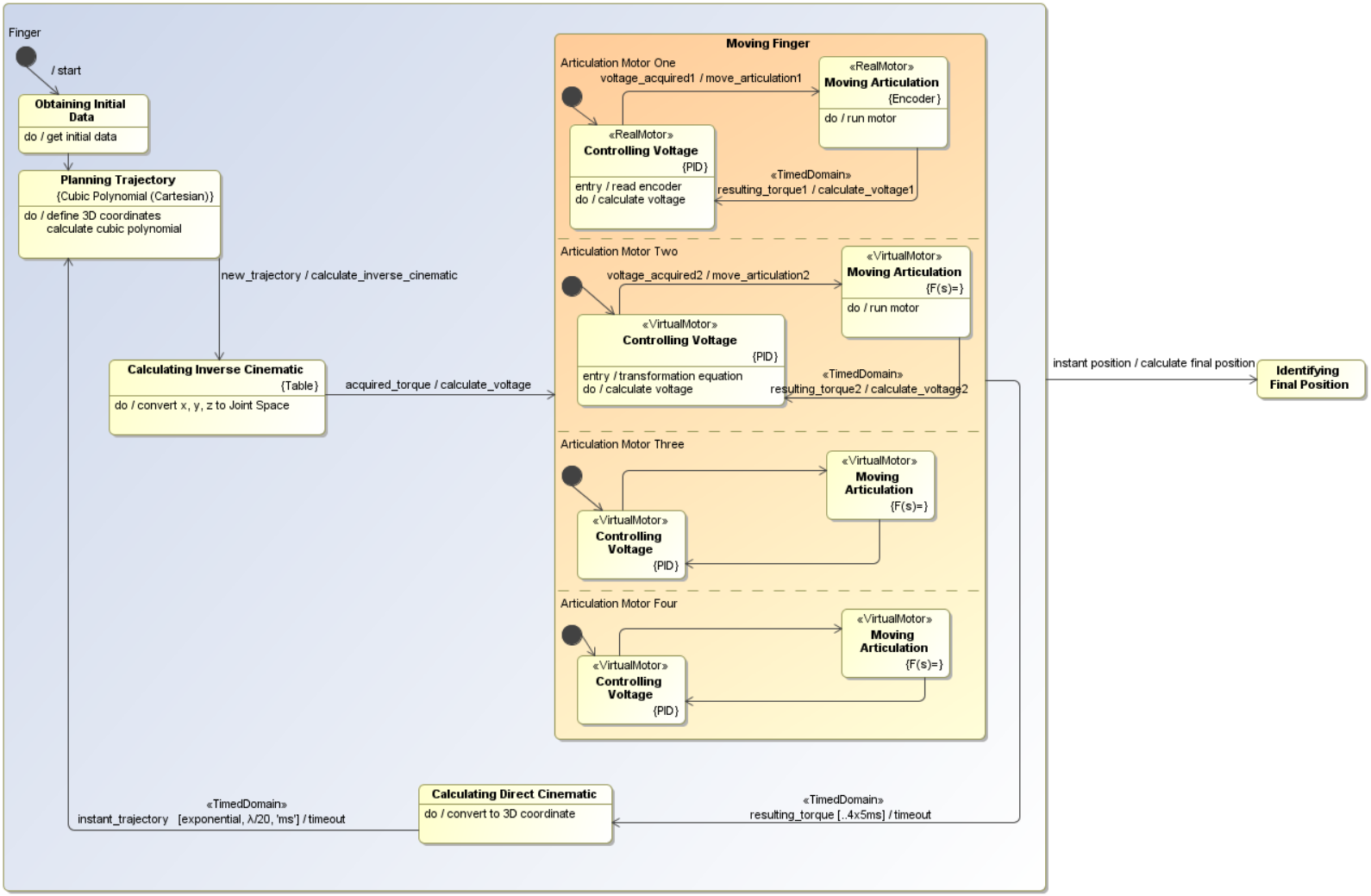


Figura 3.5 - Modelo UML Statechart da Mão Kanguera.

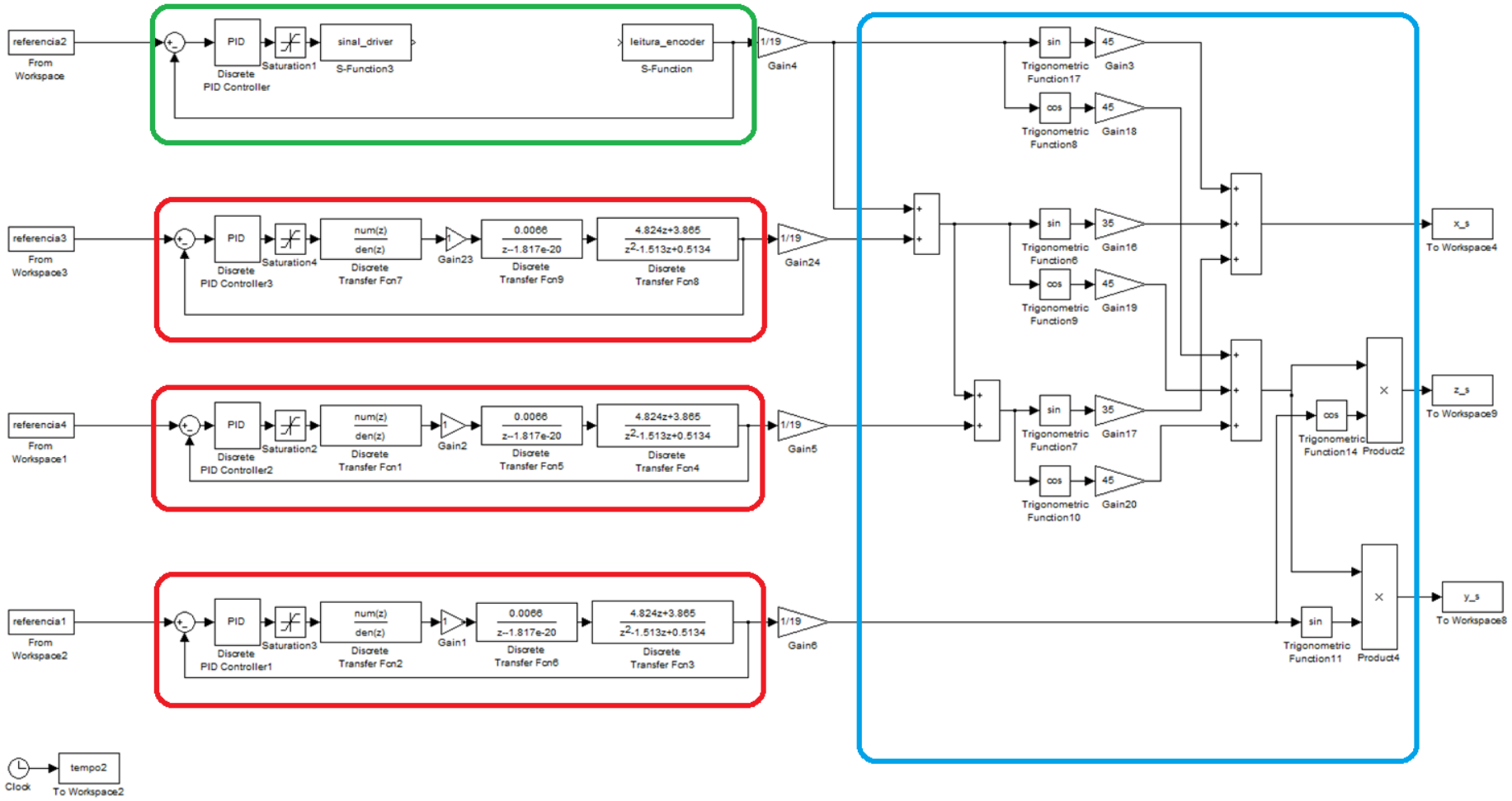


Figura 3.6 - Modelo Simulink Reformulado.

Observou-se que o processo de engenharia reversa aplicado ao modelo Simulink, permitiu identificar as funcionalidades e representá-las em um nível mais alto de abstração por meio da técnica UML Statechart. O modelo UML Statechart permitiu especificar o tempo de execução do motor, que no modelo Simulink é ocultado por ser um parâmetro do bloco que compõe o motor, podendo ser visualizado executando a ferramenta Simulink.

Após o processo de engenharia reversa, o processo de refatoração do modelo Simulink original permitiu organizar as funcionalidades, o que tornou a visualização intuitiva do fluxo de execução do modelo. Assim, a técnica UML Statechart pode ser considerada uma técnica complementar que pode ser utilizada objetivando melhorar a compreensão no desenvolvimento de SEs e, consequentemente, reduzir a sua complexidade.

3.4 Avaliações e Análises

Como descrito anteriormente, o objetivo desta ação foi explorar a técnica UML Statechart como forma de representar um modelo Simulink em um nível mais alto de abstração. Os resultados desta pesquisa foram publicados na 1ª Conferência Brasileira de Sistemas Embarcados Críticos – CBSEC-2011 (FREIRE, PEDRO, *et al.*, 2011). A seguir são descritas as ponderações que podem ser elaboradas como resultados esperados para as questões elaboradas para esta ação, as quais foram apresentadas na Seção 3.2:

Resultados referentes à questão 1:

Como apresentado na etapa de planejamento, a questão 1 tem o objetivo de verificar se a UML Statechart é uma alternativa para representar o modelo Simulink em um nível mais alto de abstração. Para avaliar essa questão, observou-se se a UML foi capaz de representar os seguintes itens:

a) R 1.1 – comportamento:

Considera-se que o comportamento da mão robótica ficou representado adequadamente por meio da UML Statechart, pois é possível identificar os estados pelos quais o movimento da mão deve passar. Mesmo sem

conhecimento detalhado da técnica UML Statechart, a leitura desse modelo é quase intuitiva. Por exemplo, observando a Figura 3.5 identifica-se, com certo conhecimento do domínio, que a aplicação deve se referir a uma mão, pois existe um estado rotulado *Moving Finger*, que já induz a existência de um dedo da mão. Além disso, para mover esse dedo, não é difícil observar que são precisos quatro motores, e assim por diante.

b) R.1.2 – funcionalidades:

Considera-se que as funcionalidades da mão robótica podem ser deduzidas do modelo UML Statechart, desde que haja um conhecimento do domínio como requisito para aplicação do processo. Assim, algumas funcionalidades necessárias para movimentar a mão robótica seriam: movimentar o dedo, calcular a cinemática direta, planejar a trajetória, e assim por diante, como pode ser deduzido observando-se a Figura 3.5.

c) R.1.3 – hierarquia e concorrência:

A representação hierárquica de um modelo estabelece uma organização do modelo, simplificando a representação e facilitando a compreensão como um todo. A técnica UML Statechart permite representar características de hierarquia através de estados compostos. A Figura 3.5 ilustra dois exemplos de estados compostos. O superestado *Finger*, por exemplo, é composto pelos estados que descrevem as funcionalidades necessárias para movimentar um dedo.

O segundo exemplo de hierarquia da Figura 3.5 é o estado *Moving Finger* que é composto por quatro estados chamados de *Articulation Motor* (*One, Two, Three* e *Four*). Esse estado, além de representar hierarquia, representa também a característica de concorrência, pois, por meio das linhas tracejadas, fica denotado que para movimentar um dedo é preciso que os quatro motores funcionem simultaneamente.

d) R.1.4 – requisitos não funcionais:

As tarefas executadas pela mão robótica obedecem a um tempo de resposta, com objetivo de ter um *feedback* sobre a ação executada. São estabelecidos tempos para execução dos dedos da mão e para cálculos de

avaliação de trajetória. Para a execução da ação dos dedos, as posições inicial e final são passadas para o plano de trajetória, este plano converte as posições inicial e final em dados do espaço cartesiano e em seguida para os dados do espaço de juntas. Toda essa atividade descrita precisa ocorrer em 20 milissegundos. Após esses cálculos os dados de velocidade, torque e potência são passados para o motor. O motor executa quatro ciclos de cinco milissegundos. Terminado esse ciclo, uma ação de *timeout* dispara uma transição saindo do estado *Articulation Motor* para o estado *Calculating Direct Cinematic* como está representado na Figura 3.5.

Resultados referentes à questão 2:

Como descrito no planejamento, essa questão tinha o objetivo de investigar se é possível descrever na UML Statechart todas as informações contidas no modelo Simulink.

No entanto, após elaborar o modelo UML Statechart, notou-se que a representação da mão robótica em UML Statechart não completa todas as informações que o modelo Simulink apresenta. Essas informações foram identificadas com o apoio do engenheiro desenvolvedor e descritas de forma textual. A descrição textual, por ora, é utilizada como complemento para enriquecer o modelo. Contudo, em outro trabalho de mestrado deste grupo de pesquisa estão sendo investigadas formas de representar esse tipo de informação. A seguir, apresentam-se as informações que não foram representadas no modelo UML Statechart, para cada estado:

1. *Planning Trajectory*
Tipo: Polinomial Cúbica no Espaço Cartesiano;
Entrada: Posições inicial e final (θ_i; θ_f);
Saída: Posições no espaço cartesiano 3D, ou seja, x, y e z;
Obs.: Caso o planejamento de trajetória esteja no Espaço de Juntas não é necessário o cálculo das cinemáticas inversa e direta. Sendo necessário apenas se estiver no Espaço Cartesiano.

2. *Inverse Cinematic*
Tipo: Tabela (Outros: Algébrico ou Jacobiana);
Entrada: Posições no espaço cartesiano 3D (x, y, z);
Saída: Transforma os dados do espaço cartesiano para o Espaço de Juntas.

3. *Voltage Control*
Tipo: PID (Outros: Derivativo, Proporcional);
Obs.: O tipo de controle pode ser distinto nos diferentes controles implementados.

4. *Moving Articulation* (Real Motor "Hardware")
Tipo: DC (Outro: AC);
Entrada: Encoder;
Saída: Analógico (Outras: USB, Serial ou CAN);
Tensão: Definir tensão máxima e mínima (Especificação do motor).

5. *Moving Articulation* (Virtual Motor "Software")
Tipo: DC (Outro: AC);
Entrada: Função de transferência;
Saída: Analógico, USB, Serial, CAN;
Tensão: Definir tensão máxima e mínima (Especificação do motor).

6. *Direct Cinematic*
Entrada: Posição no Espaço de Juntas;
Saída: Posição no Espaço Cartesiano.

Assim, para cada estado da Figura 3.5, identificaram-se as informações contidas no modelo Simulink que não ficam explícitas no modelo UML Statechart e que são importantes de serem documentadas. Tais informações são, por exemplo, qual a atividade que deve ser executada internamente ao estado, quais os dados de entrada e saída e o tipo que caracteriza a atividade, por exemplo: um motor pode ser AC (*Alternating Current*) ou DC (*Discrete Current*). Em alguns casos não há informação a se acrescentar, pois, segundo o engenheiro desenvolvedor, a funcionalidade de alguns estados é simples, bem estabelecida e executada a partir de uma equação matemática, como por exemplo, o estado *Calculating Direct Cinematic*.

Além disso, como a intenção desta questão foi avaliar o quanto a UML Statechart era capaz de representar os requisitos do modelo Simulink, investigou-se a possibilidade de expandir a representação feita, utilizando-se:

- *Profile* UML MARTE – extensão da UML que enriquece o modelo com informações por meio de estereótipos, anotações e *tags*;
- *Constraints* – notação contida na UML Statechart para representar as restrições contidas em um estado;

- *Atividades* – as atividades descrevem de forma textual o que precisa ser executado dentro do estado. As notações que descrevem as atividades internas a um estado foram descritas no Capítulo 2.

Com base nesses elementos foram incorporados *contraints* e estereótipos que acrescentaram informações extras ao modelo. A partir do diagrama da Figura 3.5, acrescentaram-se aos estados restrições (*contraints*) e estereótipos que informam como aquela funcionalidade foi implementada. Por exemplo:

- ***Planning Trajectory –*** passou a apresentar um *constraint* informando que aquela funcionalidade é implementada por meio de um cálculo polinomial cúbico no espaço cartesiano;
- ***Calculating Inverse Cinematic –*** passou a apresentar um *constraint* chamado *table*, informando que é utilizada uma tabela para a conversão das posições no espaço cartesiano para o espaço de junta (torque, potência e velocidade);
- ***Moving Articulation –*** passou a apresentar uma instância de um estereótipo do tipo *HwDevice* da UML MARTE com o nome de <<RealMotor>>. Esse estereótipo informa que um dos motores que acionam a articulação foi implementado a partir de um motor real (hardware). Os outros motores são representados com estereótipos do tipo *SwDevice* com nome de <<VirtualMotor>>. Esse estereótipo informa que os motores são simulados por meio de uma equação de transformação do tempo para frequência.

A Figura 3.7 ilustra as informações adicionais. Observe que os estados *Planning Trajectory* e *Calculating Inverse Cinematic* apresentam os *constraints Cubic Polynomial* (a) e *Table* (b), respectivamente. Os estados *Moving Articulation*, além de apresentarem diferentes *constraints*, *Encoder* e *F(s)* (função transformada), possuem estereótipos identificando que um estado é composto por um motor físico (c) e outro por um motor simulado (d). Todos os quatro estados apresentam atividades internas descrevendo o que precisa ser feito dentro do estado.

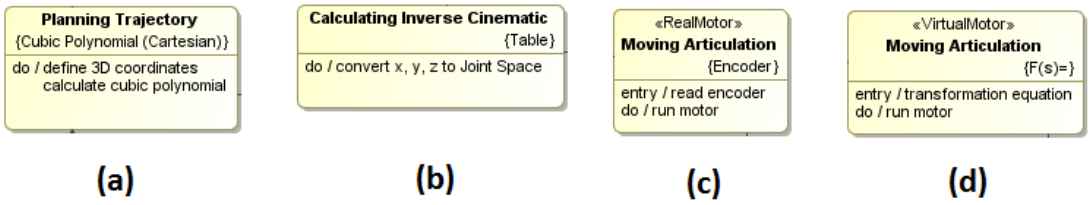


Figura 3.7 - Exemplo de uso de *profiles* e *constraints* para enriquecer a notação da UML Statechart.

Contudo, no contexto da aplicação da mão robótica, informações detalhadas e específicas da área da robótica não foram representadas no modelo. Por exemplo, a Figura 3.7-a do estado *Planning Trajectory* contém um *constraint* que informa se a funcionalidade foi implementada utilizando uma equação polinomial cúbica. Porém, a equação propriamente dita não foi especificada.

Resultados referentes à questão 3:

Como foi apresentada no planejamento, essa questão tem o objetivo de avaliar se o modelo UML Statechart ajuda a realizar uma reengenharia do modelo Simulink, com o objetivo de torná-lo mais compreensível. Assim, para responder essa questão, foi observado se a UML Statechart contribuiu para:

a) R.3.1 – criar um Simulink mais organizado e legível:

Uma vez elaborado o modelo UML Statechart, foi solicitado para o engenheiro desenvolvedor, que reimplementasse o modelo Simulink original, levando em consideração a abstração modelada na UML Statechart. A Figura 3.6 ilustra como ficou o modelo Simulink após a reestruturação. A organização e a legibilidade do novo modelo Simulink são visíveis, mesmo por um leigo em Simulink. Além disso, fica muito mais clara a informação de que há elementos que devem executar simultaneamente.

b) R.3.2 – facilitar atividades de manutenção:

No contexto deste trabalho não foi possível realizar atividades de manutenção para poder avaliar o grau de manutenibilidade do modelo Simulink reestruturado (Figura 3.6). Entretanto, a organização permitiu maior legibilidade e, certamente, uma maior compreensão do modelo. Assim, espera-se que, por meio dessa organização, a manutenção do modelo possa ser facilitada.

c) R.3.3 – documentar o modelo Simulink

O modelo UML Statechart, gerado a partir do modelo Simulink, por meio de uma atividade de engenharia reversa, serve como uma forma de documentar a aplicação, bem como de entender o modelo Simulink. Além disso, na atividade de reengenharia do modelo Simulink, o modelo UML Statechart foi fundamental para sua reorganização. Isso dá indícios que, de fato, a UML Statechart é uma alternativa viável para documentar o modelo Simulink.

3.4.1 Avaliação da Reengenharia do Modelo Simulink no Código Gerado

O modelo Simulink, intuitivamente, apresentou significativa melhoria considerando fatores como organização e legibilidade. Porém, para realizar uma avaliação quantitativa, foram analisados os códigos gerados de ambos os modelos Simulink (modelo original e modelo reestruturado). Por meio da aplicação da ferramenta Understand (Scitools, 2011), para geração de métricas de código, elaborou-se um gráfico para comparar o valor das métricas resultantes de ambos os códigos. As métricas selecionadas para avaliar ambos os modelos são aquelas que mensuram quantitativamente as linhas de código gerado e o comportamento do fluxo de programa através da métrica de complexidade ciclomática proposta por McCabe (1976).

A Tabela 3.1 apresenta os valores das métricas e a Figura 3.8 apresenta o gráfico em que são comparados os códigos gerados de ambos os modelos. Inicialmente, é descrito o significado de cada métrica:

***AvgEssential* –** Média da complexidade ciclomática para todas as funções aninhadas.
***CountLine* –** Número de linhas (NL).
***CountLineBlank* –** Número de linhas em branco (BLOC).
***CountLineCode* –** Número de linhas contendo código-fonte (LOC).
***CountLineComment* –** Número de linhas contendo comentários.
***CountLineCodeExe* –** Número de linhas contendo código-fonte executável.

***SumCyclomatic* –** Soma da complexidade ciclomática de todas as funções aninhadas (WMC).

Analisando os resultados apresentados na Figura 3.8, as métricas AltCountLineCode, CountLine, CountLineCode e CountLineCodeExe são as que apresentam valores com diferenças significativas se comparadas aos valores das demais métricas. Essas métricas avaliam o número de linhas de código a partir de diferentes parâmetros, conforme a explicação das métricas na Tabela 3.1.

A razão dessas métricas obterem resultados mais significativos em relação às demais é decorrente da redução de excessos de blocos utilizados no modelo Simulink original.

Tabela 3.1 - Comparação das Métricas (Modelo Original e Modelo Reestruturado).

Métricas	Modelo Original	Modelo Reestruturado	Diferença
AvgEssential	1,67	1,58	0,09
***CountLine* (NL)**	1.827	1.685	142
***CountLineBlank* (BLOC)**	249	232	17
***CountLineCode* (LOC)**	1.317	1.212	105
***CountLineComment* (CLOC)**	267	247	20
***SumCyclomatic* (WMC)**	20	19	1

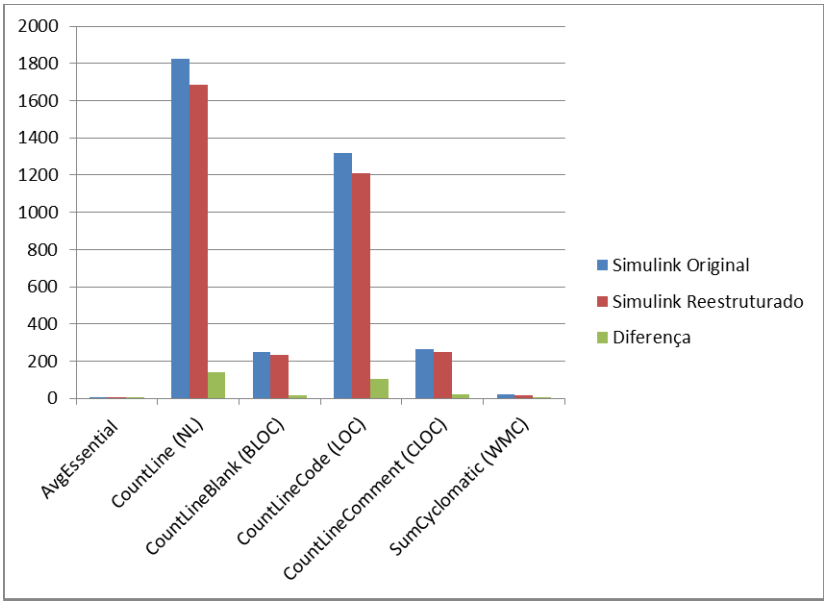


Figura 3.8 - Comparação das métricas (Original *versus* Reestruturado).

Por outro lado, a métrica MaxCyclomatic, que avalia a quantidade de desvios linearmente independentes no código, não apresenta uma diferença significativa entre os dois modelos. Ambos os modelos (original e reestruturado) não apresentam diferença relevante entre os valores da complexidade ciclomática.

Os códigos gerados pelos dois modelos Simulink foram analisados detalhadamente pelo autor e por um doutorando que faz parte do projeto. As ocorrências relatadas a seguir podem ter causado a diferença do número de linhas de código, como mencionado anteriormente:

- **Remoção de Blocos:** Foram removidas as referências do bloco *Workspace* que representam as variáveis e dados de entrada de um arquivo ".dat". Esse arquivo é usado para calibrar a mão robótica;
- **Redundância de Blocos:** O modelo Simulink original é composto por diversos blocos de ganho e adição que foram utilizados para realizar testes e simulações. Com a reestruturação do modelo, a redundância de blocos foi removida, permitindo um modelo mais enxuto;
- **Método de Log:** O método de log é utilizado no modelo Simulink original para analisar e detectar erros no fluxo de dados do modelo. Com a reestruturação esse método foi removido porque era utilizado apenas para depurar a execução do sistema durante os testes;
- **Composição de Variáveis:** Em algumas situações, os nomes de variáveis influenciam na quantidade de linhas de código. Algumas declarações seguidas de execuções ocupam mais de uma linha devido ao nome da variável e por ser uma geração automática de código.
  Exemplo:

```
controlepid_fecha_dedo_inicial2_B.Gain10 =
  controlepid_fecha_dedo_inicial2_B.Gain7 +
  controlepid_fecha_dedo_inicial2_B.Gain10;
```

A disposição de uma instrução como apresentada acima depende da geração automática do código, podendo ser disposta em uma ou mais linhas. A

Figura 3.9 apresenta a comparação realizada entre os códigos utilizando a ferramenta Understand.

A ferramenta Understand automaticamente demarca as regiões em que houve uma modificação de código. Nas partes demarcadas mostram a ausência de linhas de código do modelo original (lado esquerdo) em relação ao código do modelo reestruturado (lado direito). As linhas de códigos removidas referem-se às redundâncias de blocos que foram enxutos do modelo reestruturado.

A reestruturação do modelo Simulink ocasionou um impacto visual entre os dois modelos. A organização do modelo reestruturado pode facilitar a compreensão e futuras atividades de manutenção. Entretanto, mesmo que a melhora seja bastante intuitiva no que se refere ao modelo, a reestruturação não refletiu em uma melhora do código gerado. A organização do modelo provocou a redução de algumas linhas de código relacionadas aos blocos redundantes. Assim, as funcionalidades básicas se mantiveram as mesmas que, consequentemente, não apresentaram significativa diferença entre os códigos gerados com base na métrica de complexidade ciclomática.

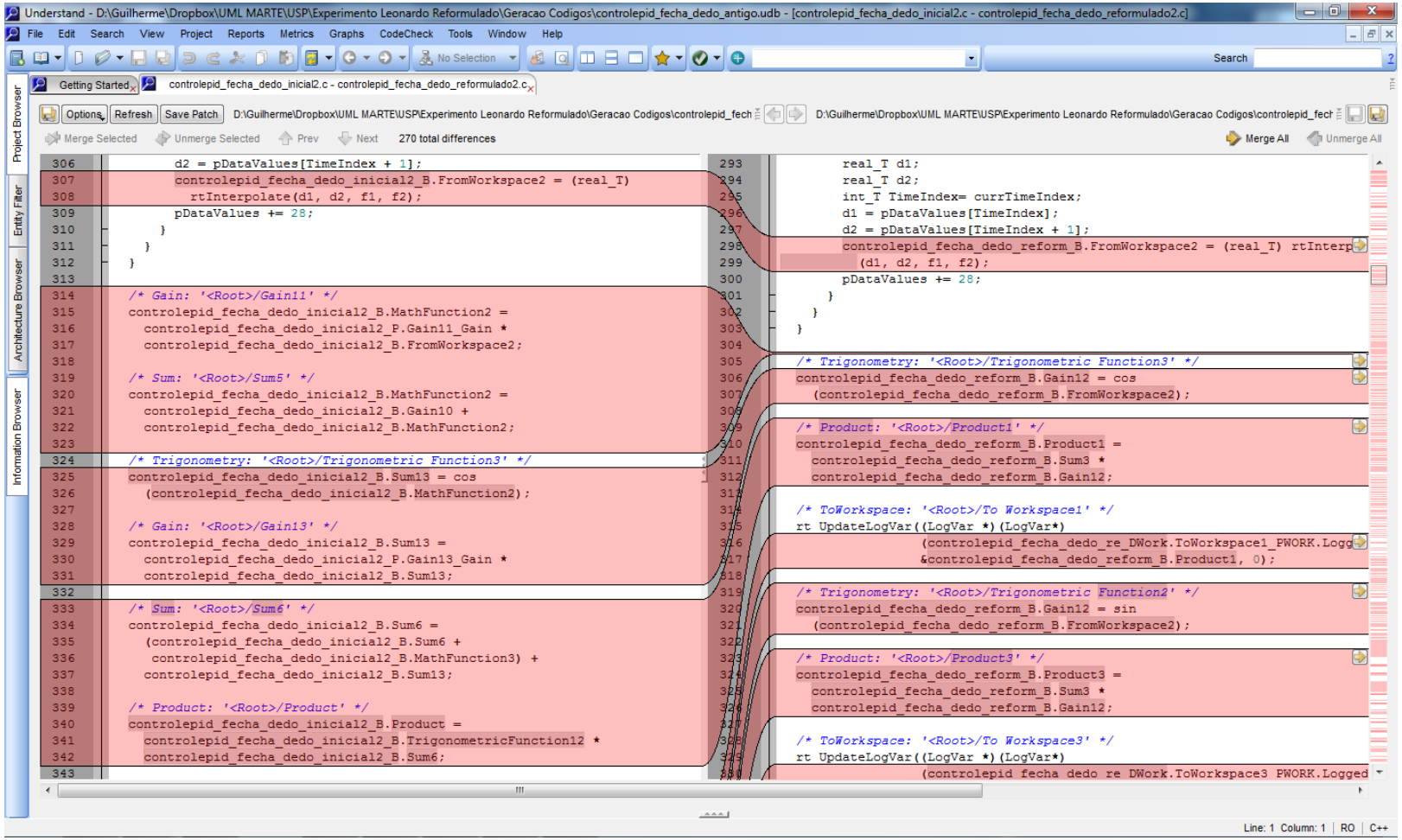


Figura 3.9 - Comparação entre os códigos dos modelos original e reestruturado.

3.5 Considerações Finais

Como descrito anteriormente, neste capítulo foi apresentada a primeira iteração do processo de pesquisa deste trabalho. Esta iteração teve como objeto de estudo a aplicação da mão robótica.

A aplicação da mão robótica permitiu avaliar, por meio de um processo de engenharia reversa e reengenharia, a técnica UML Statechart como artefatos que pudessem compor o processo de engenharia avante para o desenvolvimento de SEs. A definição desse processo de desenvolvimento a partir de um processo de engenharia avante é o objetivo de um projeto maior em que está inserido este trabalho de pesquisa.

A técnica UML Statechart, como forma de representação em um nível mais alto de abstração, retratou os requisitos contidos no modelo Simulink, permitindo reestruturá-lo de forma mais legível, estruturado e organizado. Entretanto, o modelo UML Statechart não representou todas as informações contidas no modelo Simulink. Em uma primeira instância, as representações foram identificadas e descritas de forma textual para auxiliar a construção do modelo por outros desenvolvedores. Depois, foram exploradas notações da própria técnica UML Statechart para representar as informações, antes, descritas textualmente.

Após a modelagem utilizando a UML Statechart, o modelo Simulink original foi reestruturado, gerando um modelo mais organizado e legível. Com o objetivo de avaliar o impacto dessa reestruturação, foram gerados os códigos de ambos os modelos, original e reestruturado. Contudo, por meio da análise das métricas dos códigos gerados, verificou-se que a reestruturação do modelo não ocasionou em uma melhora significativa do código.

Assim, nesta primeira iteração, a técnica UML Statechart, por permitir uma organização melhor do modelo Simulink, é a técnica selecionada para conduzir a segunda iteração de ação tendo como aplicação como objeto de estudo um sistema de controle de refrigeração. A segunda interação explora a técnica UML Statechart como parte de um documento de requisito para desenvolver SEs em um processo de engenharia avante, sendo apresentada no Capítulo 4.

Capítulo 4

CONTROLE DE REFRIGERAÇÃO – PLANEJAMENTO DA PESQUISA-AÇÃO – AÇÃO 2

Neste capítulo é descrita a segunda parte etapa de Planejamento, no que se refere à Definição da Ação; as etapas de Ação; e Avaliações e Análises da metodologia Pesquisa-Ação, tendo como objeto de estudo o sistema de controle de refrigeração.

4.1 Considerações Iniciais

Neste capítulo é apresentada a ação que explorou o uso de um modelo elaborado com a UML Statechart como um possível artefato para compor um processo de engenharia avante, no contexto do desenvolvimento de sistemas embarcados. Lembra-se que o projeto mais geral, do grupo de pesquisa, tem por objetivo definir as etapas anteriores (e seus respectivos artefatos) à etapa de construção de um modelo Simulink. Essa ação é uma ação complementar àquela apresentada no Capítulo 3, no qual se avaliou o uso de se adotar o modelo UML Statechart no contexto da engenharia reversa para modelos em Simulink.

Assim, em termos do método de pesquisa Pesquisa-Ação, as ações descritas no Capítulo 3 e neste capítulo contribuem para a mesma finalidade de avaliar a adequação da UML Statechart para representar o modelo Simulink de uma forma mais abstrata. Além disso, elas foram executadas nessa ordem, pois a primeira ação

foi um momento de aprender como abstrair o modelo Simulink. Esse conhecimento foi importante para elaborar o modelo usado na segunda ação.

Este capítulo está organizado da seguinte maneira: na Seção 4.2 descreve-se a Definição da Ação; na Seção 4.3 é descrita a Ação, apresentando a aplicação e ação realizada; e por fim, na Seção 4.4 apresentam-se as Considerações Finais.

4.2 Definição da Ação

A Definição da Ação apresentada neste capítulo segue as mesmas definidas no Capítulo 3. O objetivo é avaliar a aplicação da técnica por meio de um processo de engenharia avante.

4.2.1 Foco da Ação

Objetivo:

O objetivo deste trabalho é explorar a adequação da técnica UML Statechart para representar as funcionalidades de um modelo Simulink em um nível mais alto de abstração.

Questões:

Questão 1: A técnica UML Statechart é uma alternativa viável para representar um modelo Simulink em um nível mais alto de abstração?

Questão 2: A técnica UML Statechart foi suficiente para representar os requisitos descritos no documento de especificação?

Questão 3: A representação do modelo Simulink por meio da UML Statechart contribui para um desenvolvimento de um modelo mais organizado?

Resultados:

Resultados esperados para a questão 1:

R.1.1 – Representar o comportamento da aplicação modelada no Simulink em um nível mais alto de abstração.

R.1.2 – Representar as funcionalidades da aplicação modelada no Simulink em um nível mais alto de abstração.
R.1.3 – Representar características de hierarquia e concorrência da aplicação modelada no Simulink.
R.1.4 – Representar requisitos não funcionais da aplicação modelada no Simulink

Resultados esperados para a questão 2:
R.2.1 – Identificar se existem informações do documento de requisitos que não são representadas na técnica UML Statechart.

Resultados esperados para a questão 3:
R.3.1 – Facilitar atividades de manutenção do modelo Simulink decorrente de um modelo mais alto nível.
R.3.2 – Proporcionar uma forma complementar de documentar o modelo Simulink.

4.2.2 Hipóteses

- A técnica UML Statechart é uma alternativa para representar de forma mais alto nível um modelo Simulink, identificando os requisitos funcionais, não funcionais, hierarquia e concorrência;
- A técnica UML Statechart permite representar as informações contidas no documento de requisitos contribuindo para um processo de engenharia avante;
- A técnica UML Statechart permite organizar o modelo Simulink representando características como hierarquia e concorrência das funcionalidades;

4.2.3 Definições Operacionais

As definições operacionais descrevem as técnicas, instrumentos e ferramentas utilizadas para conduzir esta ação do trabalho.

iv. **Instrumentos –** Nesta ação do trabalho não foi utilizado nenhuma forma de instrumentação.
v. **Técnicas –** Nesta ação do trabalho foram utilizadas as seguintes técnicas:

a. **UML Statechart –** Técnica para modelar sistemas reativos. Permite representar hierarquia, concorrência e comunicação broadcast.

vi. **Ferramentas**

a. **MagicDraw –** ferramenta para modelagem de sistemas, que facilita a análise e desenvolvimento de sistemas orientados a objetos e banco de dados (MagicDraw[4], 2011);
b. **Matlab/Simulink –** ferramenta baseada em modelos para desenvolvimento de SEs (MathWorks[5], 2011);
c. **Understand –** ferramenta de análise estática para manutenção, medição e análise do código fonte (Scitools[6], 2011);

4.3 Descrição da Ação

Nesta seção é apresentada a aplicação do sistema de controle de refrigeração selecionada como objeto de estudo. A condução dessa iteração difere da aplicação anterior por aplicar a técnica UML Statechart em um processo de engenharia avante para desenvolvimento de SE. O objetivo é avaliar os modelos UML Statechart elaborados a partir do documento de requisitos e o *feedback* do engenheiro desenvolvedor em relação às duas formas de especificação, descrição textual e modelo mais alto nível de abstração das funcionalidades.

4.3.1 Descrição da Aplicação Sistema de Controle de Refrigeração

A aplicação utilizada na ação descrita neste capítulo corresponde a uma aplicação real, cujas informações foram cedidas por uma empresa da cidade de São Carlos, São Paulo, cujo produto principal é a linha de compressores. A empresa

[4] MagicDraw – versão demo, licença de 30 dias.

[5] Matlab/Simulink – versão gratuita para estudante.

[6] Scitools/Undestand – versão demo, licença de 30 dias.

possui um setor de desenvolvimento de SEs que são aplicados para automatizar e controlar a operação dos compressores.

Essa aplicação, que se tornará um SE, tem a função de controlar o compressor e dispositivos como, ventoinha, resistor de descongelamento, resistor de anti-condensação e lâmpadas, além de outros dispositivos elétricos. Ela deve controlar a temperatura de uma cabine de refrigeração, do sistema de evaporação e condensação. Possui um sistema detector de falhas para evitar detrimento elétrico ou mecânico da cabine. Contém recursos sofisticados para comunicação que incluem telemetria e comando remoto; este último, com conexão a cabo (*wired*) e sem cabo (*wireless*).

A seguir são apresentadas as três funcionalidades selecionadas a partir do documento de requisitos. O documento foi redigido em inglês e o trechos extraídos, os quais descrevem as funcionalidades selecionadas, foram traduzidos conforme a descrição do texto, sendo realizadas adaptações apenas para ajudar na compreensão de alguns termos.

4.3.2 Descrição da Ação Realizada

A ação realizada pode ser sintetizada da seguinte forma: extração dos requisitos, elaboração do modelo UML Statechart e elaboração do modelo Simulink. A partir do documento de requisitos dessa aplicação foram selecionadas três funcionalidades que o controlador do sistema de refrigeração deve executar, as quais estão descritas em seguida. Após a identificação e entendimento dos requisitos selecionados, foram elaborados os modelos UML Statechart e Simulink os quais são apresentados em seguida às suas descrições.

a) Funcionalidade 1: Iniciar Refrigerador (*Starting Refrigerator*)

A funcionalidade de inicialização do refrigerador envolve as tarefas de inicializar o compressor e medição da tensão. Para iniciar o compressor ou qualquer outro dispositivo, um sensor realiza a medição da tensão da corrente elétrica, verificando se está dentro dos limites operáveis, definidos por UV (*Under Voltage*) e OV (*Over Voltage*). Se a tensão estiver fora da faixa de operação, o controlador não inicializa o compressor, mas a tensão continua sendo medida indefinidamente. Caso a tensão esteja dentro dos limites operáveis, as bobinas de inicialização e execução

são acionadas. Sequencialmente conectado à bobina, outro sensor realiza a medição da corrente e da fase, para evitar que a bobina seja danificada durante a execução, em consequência da oscilação da tensão.

Para inicializar o motor, uma alta corrente é exigida podendo provocar uma queda excessiva na tensão. Em consequência disso, a tensão é verificada. Caso ocorra uma queda excessiva da voltagem, o controlador aborta a inicialização do compressor. Mesmo após a inicialização do motor, a corrente continua sendo medida para assegurar que a execução do compressor ocorra nos limites operacionais. A execução do compressor é mantida dependendo da temperatura, do modo de operação para economia de energia ou da detecção de falhas.

Com base nessa especificação de requisitos, construiu-se o modelo UML Statechart apresentado na Figura 4.1. O modelo é composto por um superestado ortogonal que descreve as funcionalidades concorrentes *Measuring Voltage* e *Starting Compressor*. O estado *Measuring Voltage* é executado indefinidamente com a finalidade de avaliar a tensão da corrente elétrica, evitando que o equipamento seja danificado. Se a tensão estiver dentro dos limites operáveis, o estado *Measurin AC Mains Voltage* dispara um evento por meio de uma ação para conectar o compressor aos interruptores de energia (TRIACS). Caso a tensão não esteja nos limites operáveis, o compressor é desligado. Se a tensão estiver nos limites operáveis o compressor continua operando (*Compressor Running*). Antes de entrar no estado *Compressor Ranning*, a fase da energia é medida no estado *Measuring Crossing Phase* para identificar oscilação de fase durante a operação do compressor. A Figura 4.2 ilustra o diagrama Simulink que foi elaborado a partir do modelo UML Statechart da Figura 4.1.

O modelo UML Statechart contribuiu para uma visualização mais ampla do comportamento das funcionalidades selecionadas, representando atividades concorrentes presentes na Funcionalidade 1. As atividades concorrentes mostram a dependência entre as duas atividades apresentadas na Figura 4.1. Essa concorrência é apresentada na Figura 4.2 da seguinte forma:

O bloco denominado de *Voltage Line* faz a leitura da tensão, verificando se está dentro dos limites operáveis. Se a tensão estiver dentro dos limites operáveis, então a saída será para o bloco denominado *Ove Voltage Fail* que dispara os dados para inicializar o compressor. O compressor continuará operando se todos os blocos

de entrada (1 – *Compressor Command*, 2 – *Voltage Fail*, 3 – *Energy Saver*, 4 – *Thermostat*, 5 – *Temperature Warnings/Fail*) estiverem enviando sinal para continuar a operação. A execução do compressor é dependente do sinal enviado pelos cinco blocos de entrada e principalmente da tensão que é medida indefinidamente, para garantir que o compressor não seja danificado.

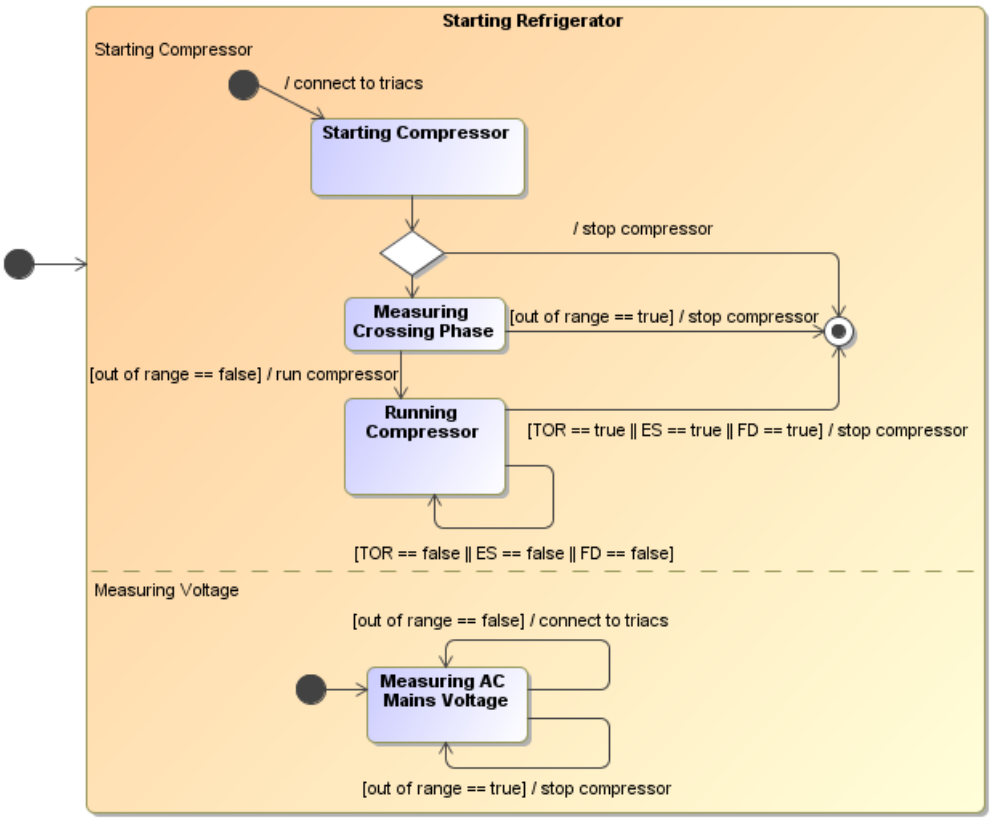


Figura 4.1 – Modelo UML Statechart do Sistema de Inicialização do Refrigerador.

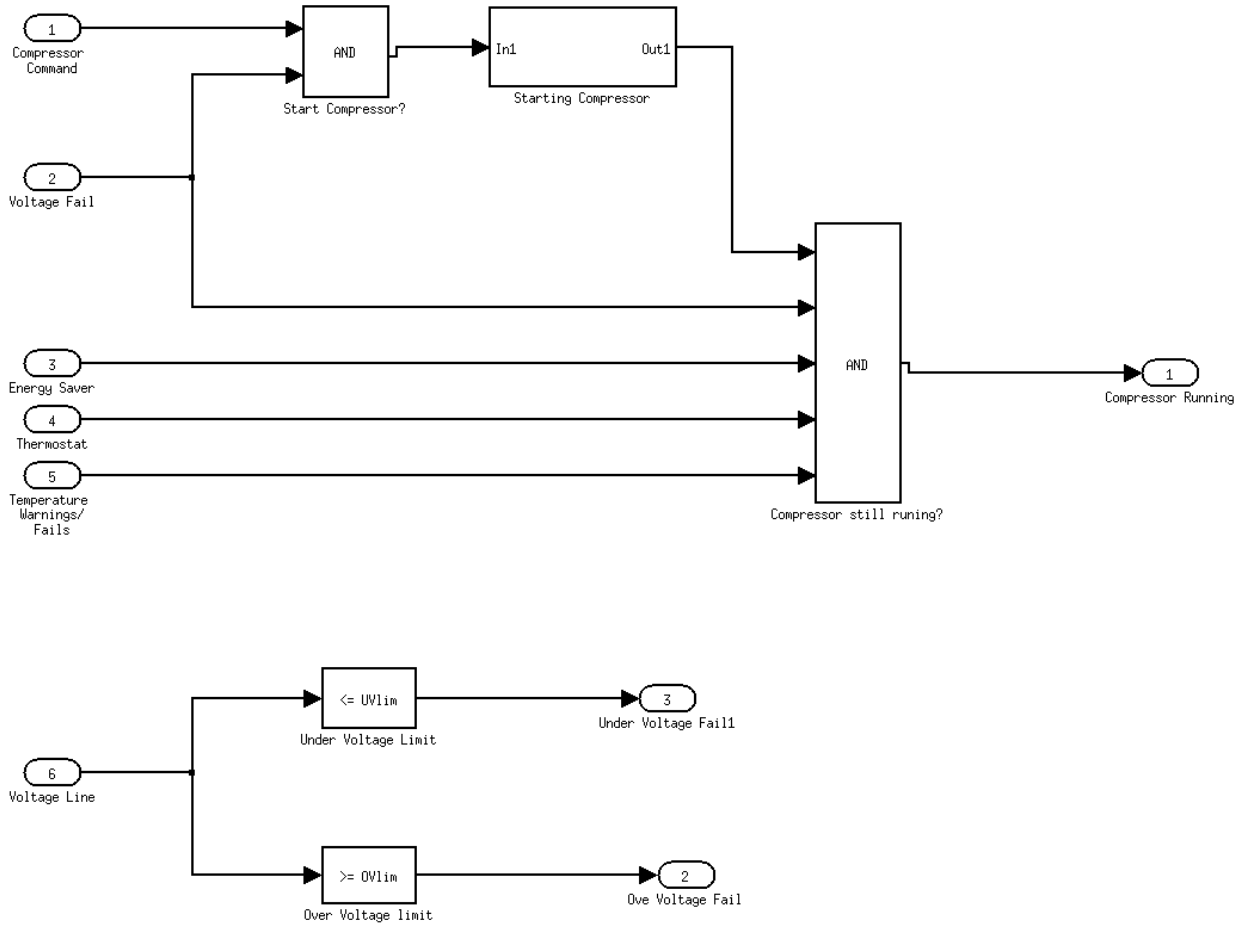


Figura 4.2 – Modelo Simulink de Inicialização do Refrigerador.

b) Funcionalidade 2: Iniciar Compressor (*Starting Compressor*)

A principal funcionalidade executada pelo controlador é a funcionalidade de acionamento do compressor. Essa funcionalidade identifica o fim da inicialização do motor medindo a mudança de fase entre a tensão e a corrente. Por meio dessa medição, é detectada a desaceleração do motor e desconectada a bobina de inicialização.

Para inicializar o motor, a corrente elétrica e a tensão podem sofrer mudanças de fase até que o motor alcance a velocidade operacional. Após ter atingido a velocidade operacional, a fase se estabiliza, sendo o momento propício para desligamento da bobina de inicialização. Um programa de software avalia a mudança de fase entre a corrente e a tensão para verificar o fim da aceleração do motor e desconectar a bobina de inicialização.

Contudo, se a fase indicar que o rotor não atingiu a velocidade rotacional desejada, o controlador desenergiza as bobinas principais e auxiliares. Após um determinado tempo, é iniciada uma nova sequência de acionamento. Existe um número fixo para tentativas de acionamento do motor. Caso o número de tentativas seja ultrapassado e o motor não conseguir ser acionado, é estabelecida uma falha rotacional e o sistema de inicialização é travado.

Conforme ilustra a Figura 4.3, para iniciar o compressor, duas bobinas trabalham em paralelo, a bobina de inicialização e a bobina de execução. A bobina de execução irá continuar operando até que o termostato atinja a temperatura desejada.

A bobina de inicialização é acionada e em seguida, sem a necessidade de um evento, realiza a medição da fase. Se a fase não estiver estável as bobinas são desenergizadas e ocorre outra tentativa após um determinado intervalo de tempo. Se exceder o número determinado de tentativas, o rotor é travado indicando falha da inicialização. Caso a fase esteja estável, então a bobina de inicialização é interrompida, operando apenas a bobina de execução. A Figura 4.3 ilustra a modelagem em UML Statechart e a Figura 4.4 ilustra o modelo Simulink.

A UML Statechart permite complementar a especificação de uma funcionalidade caracterizando o seu comportamento interno por meio de representações hierárquicas. Por exemplo, na Figura 4.1 o estado *Starting Compressor* é representado em um nível hierárquico mais alto de forma que a representação do modelo seja mais enxuta. A representação mais detalhada do estado *Starting Compressor* é apresentada na Figura 4.3.

Assim, o desenvolvimento do modelo Simulink pode ser dividido entre os desenvolvedores de SE, ou seja, um desenvolvedor se encarrega de implementar a funcionalidade de inicialização do refrigerador e um outro desenvolvedor pode implementar a funcionalidade de inicialização do compressor. Essa forma de desenvolvimento segue a ideia de divisão de trabalho bastante conhecida entre os desenvolvedores de sistemas de software tradicionais.

c) Funcionalidade 3: Economizador de Energia (*Energy Saver*)

A funcionalidade do *energy saver* é uma das mais importantes do controlador, sendo responsável pela economia de energia do refrigerador. O objetivo é

racionalizar o consumo de energia desligando dispositivos que não necessitam estar operando.

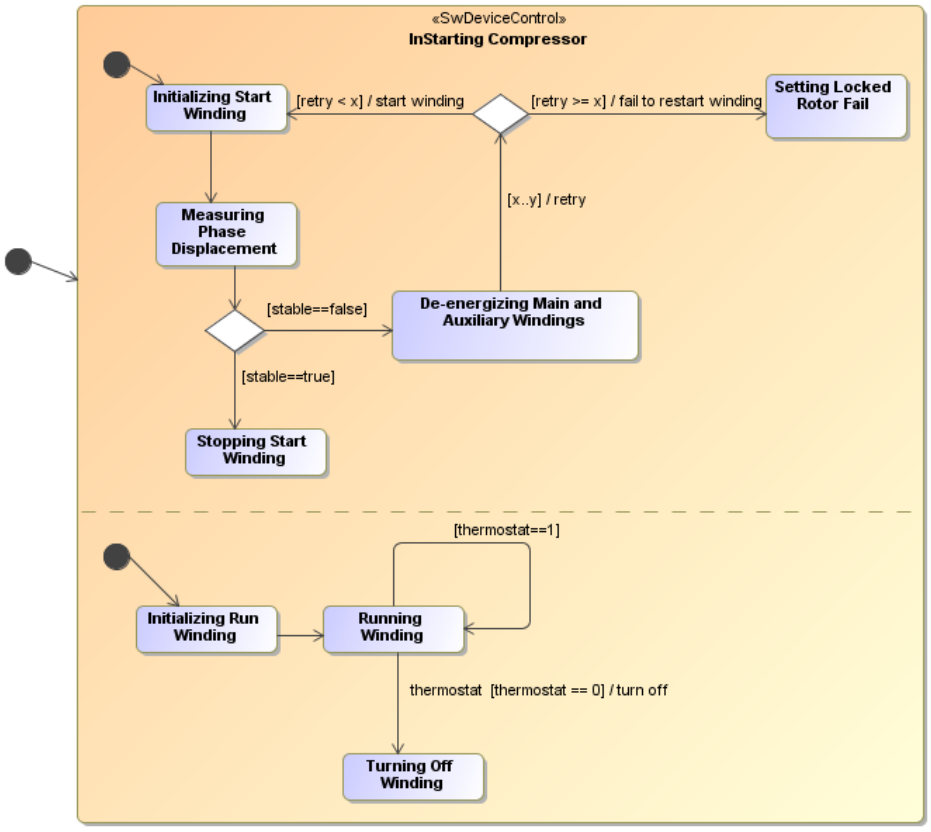


Figura 4.3 – Modelo UML Statechart da Inicialização do Compressor.

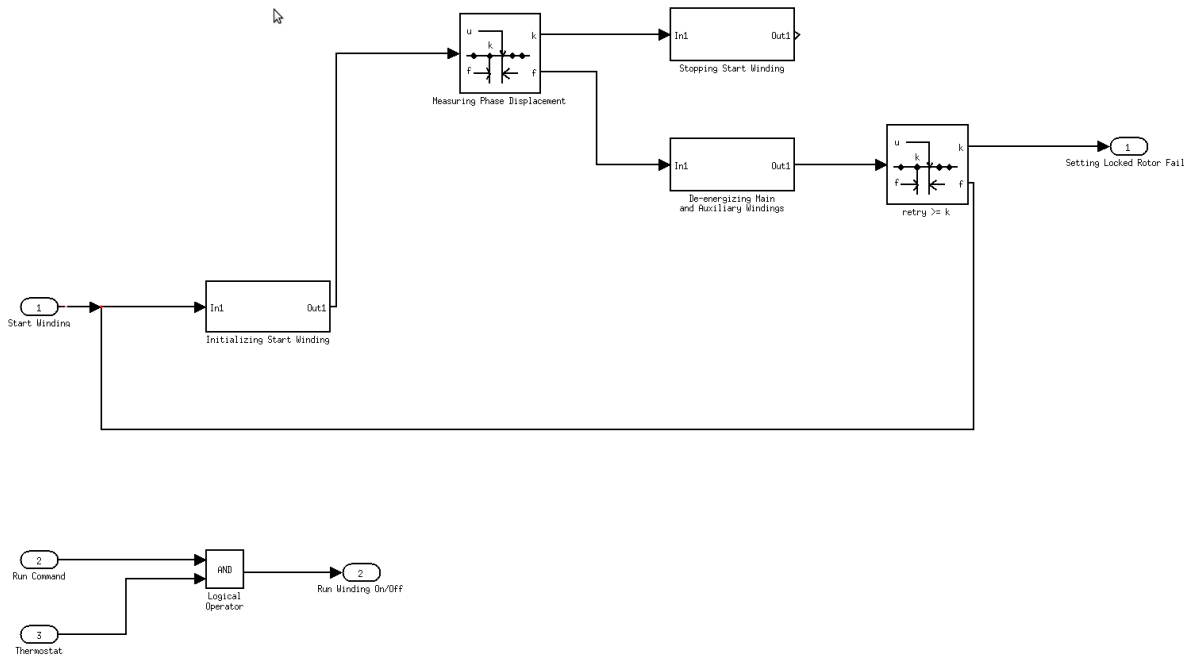


Figura 4.4 – Modelo Simulink de Inicialização do Compressor.

Dias Comerciais

***Stand-by* –** Nesta configuração o compressor e as ventoinhas do condensador e evaporador estão operando. O ponto de temperatura é estabelecido como alta no termóstato. Os pontos alto e baixo de temperatura correspondem às temperaturas máxima e mínima que se deseja que a cabine opere.

Inicialização (*pull-down*) – Nesta configuração é estabelecido um tempo para iniciar o resfriamento da cabine. Esse tempo é ilustrado na Figura 4.5 pela sigla PDD (*Pull-Down Duration*). A inicialização irá resfriar a cabine até o horário comercial, estabelecendo a temperatura até o ponto baixo (temperatura estabelecida para a cabine operar durante o horário comercial). Caso o tempo de duração do PDD não esfrie a cabine até o ponto baixo estabelecido, no dia seguinte uma funcionalidade do controlador irá calcular um novo tempo para o PDD atingir a temperatura desejada.

Horário Comercial – Nesta configuração a cabine opera com lâmpadas e resistência da porta ligada, e o sistema de descongelamento executando a cada oito horas. Se a porta não for aberta durante as quatro horas iniciais, o sistema de controle estabelece a configuração da cabine para o modo *stand-by*. Caso a porta seja aberta uma determinada quantidade de vezes, o sistema de controle estabelece o modo de horário comercial.

A Figura 4.5 descreve os horários de funcionamento para os dispositivos listados ao lado esquerdo da figura. A primeira linha vertical tracejada mais a esquerda define o horário em que os dispositivos estão operando no modo *Stand-by* no horário 0:00. As seguintes linhas tracejadas definem os horários principais de funcionamento do *Energy Saver*, que são, 8:00, 18:00 e 24:00.

Na parte superior da figura, os modos de operação *Stand-by* e Duração Horário Comercial, são definidos por setas de duplo sentido que ilustram o intervalo de tempo do funcionamento do modo de operação selecionado.

As linhas horizontais contínuas, respectivas para cada dispositivo listado à esquerda, exceto para o dispositivo da lâmpada, estão em nível baixo para o modo de operação em *Stand-by* e em nível alto quando em modo de operação de Horário Comercial. A lâmpada, em *Stand-by*, tem seu nível estabelecido como alto, em consequência da temperatura estar estabelecida em ponto alto e o nível

estabelecido como baixo, quando entra em modo de operação de Horário Comercial. A Tabela 4.1 apresenta as legendas para as siglas contidas na Figura 4.5.

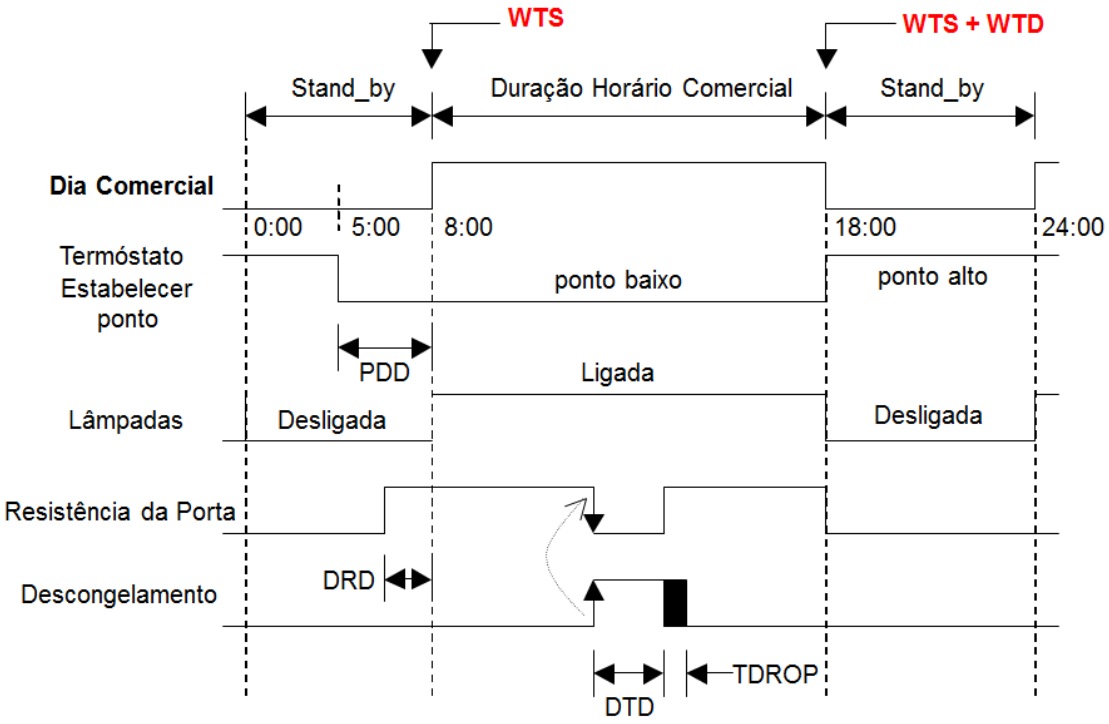


Figura 4.5 – Funcionamento em Dias Comerciais.

Tabela 4.1 – Tabela de Legendas.

WTS (*Working Time Start*)	Início do Horário Comercial
WTD (*Working Time Duration*)	Duração do Horário Comercial
PDD (*Pull-Down Duration*)	Intervalo de Inicialização
DRD (*Door Resistance Duration*)	Duração da Resistência da Porta
DTD (*Defrost Time Duration*)	Duração do Tempo de Descongelamento

Dias Não Comerciais

***Stand-by* –** Nesta configuração, se a porta for aberta uma vez o sistema de controle irá reduzir a temperatura até zero grau Celsius. Caso a porta seja aberta duas vezes na mesma hora, o controlador irá estabelecer a configuração para horário comercial.

A Figura 4.6 ilustra o funcionamento dos dispositivos em um dia não comercial. Todos os dispositivos estão desligados com exceção do termóstato que é

estabelecido no ponto alto (temperatura estabelecida para funcionamento em horários não comerciais).

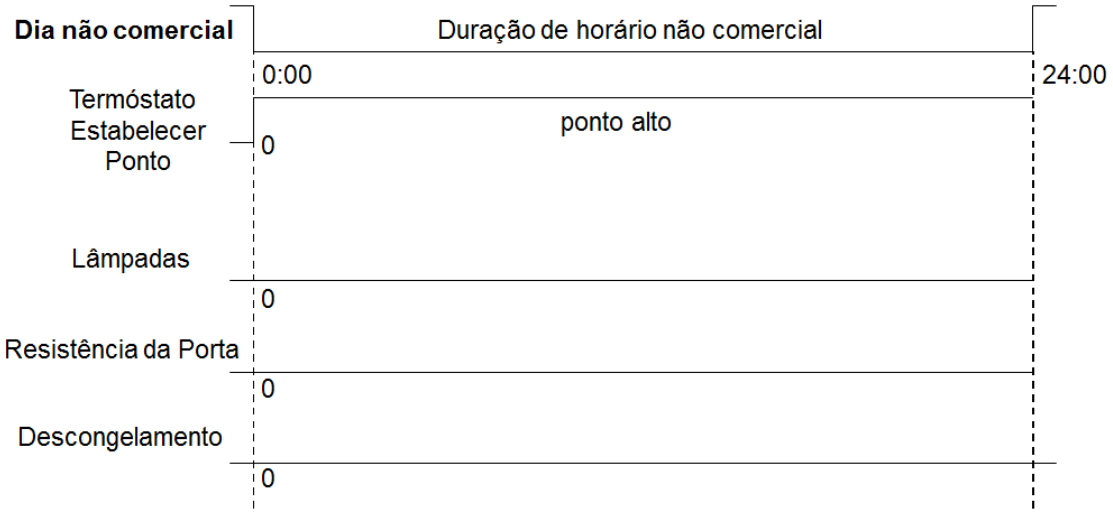


Figura 4.6 – Funcionamento em Dias Não Comerciais.

De acordo com as descrições apresentadas de como o *Energy Saver* opera, foram elaborados os modelos UML Statechart (Figura 4.7) e Simulink do *Energy Saver* (Figura 4.8).

O modelo UML Statechart, validado pelo engenheiro da empresa, permitiu descrever as funcionalidades concorrentes. Para cada estado concorrente foi descrito todo o ciclo de funcionamento, descrevendo as restrições de tempo em que cada funcionalidade deve ser executada. Alguns dispositivos têm suas funcionalidades disparadas por meio da comunicação em *broadcast*. Quando uma determinada hora é atingida, apenas as funcionalidades em que as transições possuem a ação irão ser executadas em paralelo. Assim, apenas algumas funcionalidades irão transitar de um estado para outro, a depender do horário.

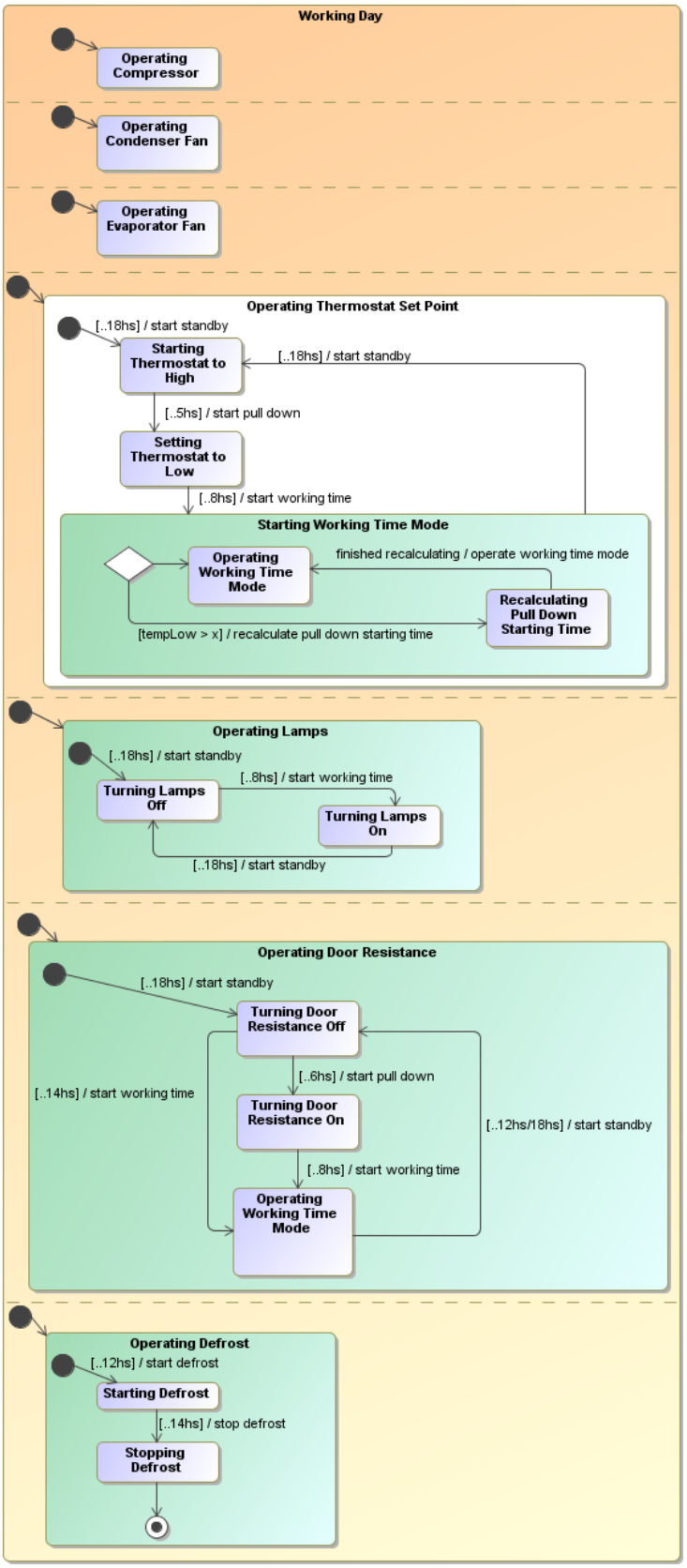


Figura 4.7 – Modelo UML Statechart do *Energy Saver*.

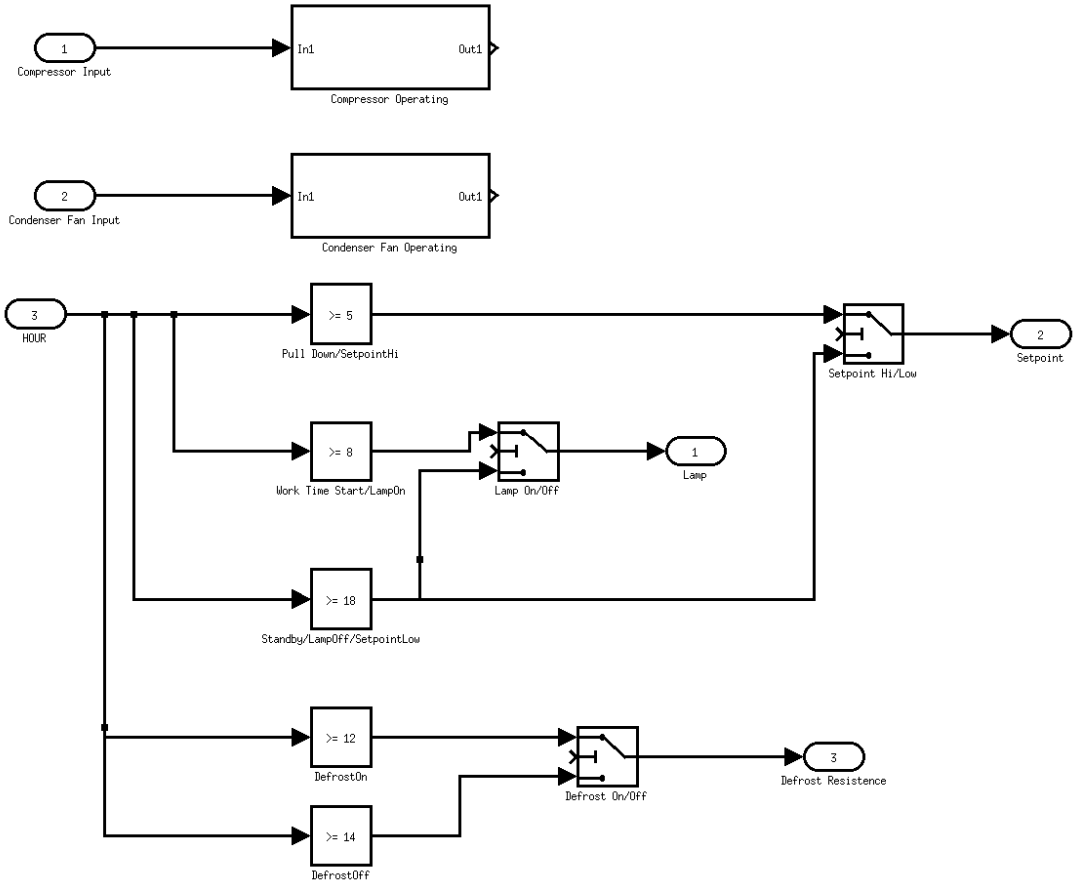


Figura 4.8 – Modelo Simulink do *Energy Saver*.

4.4 Avaliações e Análises

A aplicação do controlador de sistema de refrigeração serviu como uma segunda iteração para avaliar a aplicação da UML Statechart como artefato para o desenvolvimento de modelos Simulink. Nessa iteração, o desenvolvimento seguiu um processo de engenharia avante, partindo de um documento de requisitos, que descreve textualmente as funcionalidades do sistema de controle de refrigeração.

Com base nos modelos UML Statechart, foram construídos os modelos Simulink por um engenheiro desenvolvedor da empresa que cedeu as informações sobre essa aplicação. Foi pedido para o engenheiro que, durante a construção dos modelos Simulink, identificasse informações que não estivessem representadas no modelo UML Statechart. Entretanto, o engenheiro informou que os modelos UML

Statechart estavam de acordo com as descrições do documento de requisitos e que não seria necessário acrescentar nenhuma informação auxiliar.

A seguir são apresentadas as avaliações com base no planejamento apresentado na Seção 4.2, o qual é similar ao planejamento da primeira iteração da Pesquisa-Ação, correspondente à ação descrita no Capítulo 3:

Resultados referentes à questão 1:

Como apresentado na etapa de planejamento, a questão 1 tem o objetivo de verificar se a UML Statechart é uma alternativa para representar a aplicação de tal forma que dê subsídios para construir o modelo Simulink e que seja, portanto, uma representação do Simulink em um nível mais alto de abstração. Para avaliar essa questão, observou-se que a UML foi capaz de representar os seguintes itens:

a) R 1.1 – comportamento:

O modelo UML Statechart permitiu representar o comportamento da aplicação em um nível mais alto de abstração elaborado a partir da descrição do documento de requisitos. Utilizando os modelos UML Statechart, mesmo sendo necessário utilizar algum artefato para acrescentar informações adicionais, a compreensão se torna mais objetiva e direta no que se refere a uma visão geral do sistema. O fato de ter construído o modelo UML Statechart antes do modelo Simulink permitiu entender a aplicação e construir o Simulink já de maneira organizada. O modelo UML Statechart descreveu as funcionalidades dependentes, como por exemplo, na Figura 4.1 o estado *Starting Compressor* irá estabelecer uma conexão aos interruptores de energia se o estado *Measuring AC Mains Voltage* disparar uma ação chamada *connect to triacs*. Esse é um exemplo de comunicação em *broadcast* do modelo UML Statechart e exemplo de que uma funcionalidade é dependente de outra para ser executada.

b) R.1.2 – funcionalidades:

O modelo UML Statechart permitiu representar as funcionalidades em um nível mais alto de abstração. As funcionalidades descritas no documento de requisitos não oferecem uma visão geral de todas as funcionalidades e suas dependências. No caso específico do documento de requisitos utilizado nesta

pesquisa, as descrições referentes a uma funcionalidade são fragmentadas em várias partes do texto.

Por meio do modelo UML Statechart o comportamento é visualizado de uma maneira melhor. Por exemplo, a Figura 4.1 ilustra que para inicializar o refrigerador é preciso inicializar o compressor que, por sua vez, depende da medição da tensão elétrica. A Figura 4.3 ilustra que o estado *Starting Compressor* depende da operação de dois tipos de bobina: inicialização (*Initializing Start Windings*) e execução (*Initializing Run Windings*).

Uma das dificuldades encontradas ao elaborar o modelo UML Statechart foi compreender a relação entre as funcionalidades, inicializar refrigerador, inicializar compressor e inicializar a bobina. Diversas vezes o documento foi relido para entender como se comportava a estrutura geral das funcionalidades. Com o modelo UML Statechart, a visualização da estrutura do sistema se tornou mais objetiva. Descobriu-se que para inicializar o refrigerador é necessário inicializar o compressor. O compressor, por sua vez, depende das bobinas de inicialização e execução para ser iniciado.

Assim, a UML Statechart permite facilitar a compreensão da estrutura do sistema. As informações contidas no documento de requisitos não expressavam a visualização geral dessa estrutura, sendo necessárias diversas releituras para compreender como essas estruturas se organizavam.

c) R.1.3 – hierarquia e concorrência:

O modelo UML Statechart permitiu representar características de hierarquia e concorrência presentes na especificação dos requisitos. Por exemplo, As Figura 4.1 e 4.3 estão relacionadas por representarem hierarquicamente o estado *Starting Compressor*. Na Figura 4.1 o estado *Starting Compressor* é apresentado em nível mais alto de hierarquia, sem descrever o seu comportamento interno. Na Figura 4.3, o estado *Starting Compressor* é representado como superestado e detalhado em suas atividades que compõem essa funcionalidade.

Ainda na Figura 4.3, para inicializar o compressor (*Starting Compressor*), as bobinas de inicialização e execução são acionadas concorrentemente. Essa concorrência é identificada pela linha vertical tracejada que separa os dois estados

concorrentes. No estado superior da Figura 4.3, acima da linha tracejada, estão representadas as atividades para acionar a bobina de inicialização. No estado inferior, abaixo da linha tracejada, estão representadas as atividades da bobina de execução.

d) R.1.4 – requisitos não funcionais:

O modelo UML Statechart permitiu representar o requisito não funcional de tempo, que era o único requisito desse tipo presente na especificação. Na Figura 4.5 foram descritos os horários em que a funcionalidade *Energy Saver* deve iniciar os modos de operação correspondentes aos dias comerciais.

Resultados referentes à questão 2:

O sistema de controle de refrigeração é um estudo que seguiu um processo de engenharia avante. Assim, foi verificado se a UML Statechart permitiu representar todas as informações contidas no documento de requisitos das funcionalidades selecionadas. O sistema de controle de refrigeração, por ser um exemplo mais simples, não explorou as notações da técnica para caracterizar como a funcionalidade foi implementada, sendo suficientes as informações contidas no modelo UML Statechart, de acordo com o engenheiro desenvolvedor.

Contudo, as funcionalidades exigiram das três características principais da técnica UML Statechart, hierarquia, concorrência e comunicação em *broadcast*. A Figura 4.7 explorou essas três características. Foram modelados sete dispositivos que executam concorrentemente durante o horário comercial. Alguns desses dispositivos apresentam hierarquia de estados para descrever a sua funcionalidade durante o dia, como por exemplo, o estado *Operating Lamps*. A comunicação em *broadcast* está contida nos estados que dependem de uma ação em comum a outros dispositivos para mudança de estados, como por exemplo, os estados *Operating Thermostat Point*, *Operating Lamps* e *Operating Door Resistance* realizam uma transição de estados quando a ação */starting working time*, com condição de oito (8) horas, for disparada.

Assim, a técnica UML Statechart permitiu que as informações contidas no documento de requisitos, referentes às três funcionalidades selecionadas, fossem representadas, apresentando restrições de tempo, condições para execução, como

temperatura atingida, fase estabilizada, e a dinâmica das funcionalidades por meio de uma visualização mais direta e objetiva.

Resultados referentes à questão 3:

a) R.3.1 – facilitar atividade de manutenção:

Não foram realizadas atividades de manutenção para avaliar o grau de manutenibilidade de um modelo Simulink desenvolvido com apoio da técnica UML Statechart. Entretanto, o documento de requisitos, com descrições detalhadas das funcionalidades, em conjunto com a técnica UML Statechart permitem melhor compreensão e organização do sistema, facilitando a manutenção.

b) R.3.2 – documentar o modelo Simulink

Com o apoio do engenheiro desenvolvedor, a UML Statechart, a partir de uma representação em um nível mais alto de abstração, permitiu uma visão geral do comportamento do sistema. Sendo assim, a técnica pode ser utilizada como artefato para apoiar a documentação do sistema.

4.5 Considerações Finais

Neste capítulo foi apresentada a segunda iteração do processo de pesquisa deste trabalho, tendo como objeto de estudo o dispositivo de controle de refrigeração. A ação do sistema de controle de refrigeração seguiu um processo de engenharia avante, contribuindo para avaliar a técnica UML Statechart como artefato para representar modelos Simulink de forma mais abstrata.

A partir de um documento de requisitos as funcionalidades do dispositivo de controle de refrigeração foram modeladas na UML Statechart. Os modelos elaborados permitiram avaliar a técnica UML Statechart como artefato para especificação em processo de engenharia avante.

A aplicação do dispositivo de controle, por ser mais simples, não exigiu especificação detalhada das informações. As informações contidas nos modelos UML Statechart foram suficientes para o engenheiro desenvolvedor construir o modelo Simulink. As funcionalidades implementadas realizavam tarefas simples de

controle (medindo temperatura e corrente elétrica, horário para iniciar e encerrar uma tarefa), ao contrário da aplicação da mão robótica, que contém funcionalidades específicas da área de robótica.

Dessa forma, nesta segunda iteração, a aplicação da técnica UML Statechart avaliou a técnica em um processo de engenharia avante. Essa iteração contribuiu com a avaliação iniciada no Capítulo 3, em que a ação foi conduzida por um processo de engenharia reversa e reengenharia. Em paralelo às duas iterações, foi aplicada uma terceira iteração para investigar a cultura de desenvolvimento entre os desenvolvedores de SE. A terceira ação tem como objetivo investigar se os desenvolvedores inseridos no contexto do INCT-SEC utilizam alguma forma de representação gráfica para apoiar o desenvolvimento de SEs.

Capítulo 5

APLICAÇÃO DO *SURVEY* – PLANEJAMENTO DA PESQUISA-AÇÃO – AÇÃO 3

Neste capítulo é apresentada a ação correspondente ao survey realizado entre os desenvolvedores de sistemas embarcados, no contexto do INCT, cujo objetivo foi identificar a cultura de desenvolvimento entre esses desenvolvedores.

5.1 Considerações Iniciais

Neste capítulo é descrita a ação que foi executada paralelamente às outras duas, e que se refere a um levantamento de como os desenvolvedores de SEs constroem suas aplicações. O objetivo dessa ação foi agregar informações para enriquecer as análises e possíveis conclusões que poderiam ser traçadas com as informações obtidas nas outras duas ações.

Durante as outras ações realizadas neste trabalho, foram levantadas algumas dúvidas referentes à cultura de desenvolvimento dos desenvolvedores de SE, inseridos no contexto do INCT. Viu-se então a necessidade de investigar se esses desenvolvedores utilizavam alguma técnica de modelagem, qual abordagem de desenvolvimento era praticada por eles, e se eles preparavam alguma documentação do sistema para futuras atividades de manutenção. Assim, o *survey* teve por objetivo caracterizar o procedimento adotado por desenvolvedores de SE

que fazem uso das ferramentas Simulink e/ou LabVIEW, uma vez que, no contexto desta pesquisa, já se tinha decidido estabelecer como alvo a ferramenta Simulink.

Com base na metodologia Pesquisa-Ação, o *survey* aqui apresentado foi definido de maneira similar às outras duas ações. O capítulo está organizado da seguinte forma: na Seção 5.2 descreve-se a Definição da Ação; na Seção 5.3 é descrita a Ação, na Seção 5.4 são apresentadas Avaliações e Análises e, na Seção 5.5, as Considerações Finais.

5.2 Definição da Ação

Nesta seção é definida a ação para a condução da pesquisa, descrevendo o objetivo, as questões e resultados esperados.

5.2.1 Foco da Ação

Como o foco desta ação é verificar se os desenvolvedores de SE fazem uso da técnica Statechart, independentemente de ser a técnica Statechart tradicional, desenvolvida por Harel (1987), ou qualquer de suas variações, utilizou-se simplesmente o nome Statechart, ao invés de UML Statechart, como foi usado nos demais capítulos.

Objetivo:

O objetivo da ação é identificar se os desenvolvedores de SE, inseridos no contexto do INCT, utilizam alguma forma de documentação e especificação durante o processo de desenvolvimento. Assim, este objetivo foi implementado na forma de um *survey*, no contexto de participantes do projeto INCT-SEC.

Questões:

Questão 1: Os desenvolvedores de SE que fazem uso da técnica Statechart para representar modelos Simulink em um nível mais alto de abstração exploram os recursos do Statechart de forma a facilitar o desenvolvimento do modelo Simulink?

Resultados:

Resultados esperado para questão 1:

R.1.1: Verificar se os desenvolvedores de SE utilizam alguma técnica para representar a aplicação em um nível mais alto de abstração do que o modelo Simulink.

R.1.2: Verificar se os desenvolvedores de SE fazem uso da técnica Statechart para apoiar o desenvolvimento de modelos Simulink.

R.1.3: Verificar se os recursos da técnica Statechart estão sendo explorados de forma apropriada para levar à construção de um modelo Simulink que retrate os requisitos da aplicação.

5.2.2 Hipóteses

Com base no objetivo da ação, a hipótese é a seguinte:

- A técnica Statechart é utilizada pelos desenvolvedores participantes do *survey* para apoiar o desenvolvimento de modelos Simulink, explorando as características da técnica de forma a representar o sistema em um nível alto de abstração e de compor um artefato de documentação que retrata as características da aplicação.

5.2.3 Definições Operacionais

As técnicas, instrumentos e ferramentas utilizadas para conduzir esta ação são:

vii. **Instrumentos –** Formulário online, disponibilizado para os participantes durante um período previamente definido.

viii. **Técnicas –** por se tratar de um *survey*, as técnicas foram apenas referenciadas na pesquisa e não utilizadas:

a. **Statechart –** Técnica para modelar sistemas reativos. Permite representar hierarquia, concorrência e comunicação *broadcast*.

ix. **Ferramentas –** por se tratar de um *survey*, as ferramentas foram apenas referenciadas na pesquisa e não utilizadas:
   a. Matlab/Simulink
   b. LabVIEW

5.3 Descrição da Ação

O *survey*, composto de dez questões, foi aplicado em um grupo de dezenove desenvolvedores de SE que utilizam as ferramentas Simulink ou LabVIEW, e que estão inseridos no contexto do projeto INCT-SEC. Essas questões foram organizadas em um formulário online, que ficou acessível por um período de trinta dias. Os participantes foram sugeridos pelo engenheiro desenvolvedor envolvido na ação descrita no Capítulo 3, uma vez que esses indivíduos eram, com certeza, desenvolvedores de SEs. Os participantes foram contatados por email e, depois de aceitarem participar do *survey*, receberam o convite para poder acessar o formulário.

A seguir, apresentam-se as questões do formulário e seus respectivos resultados:

1 – Qual seu nível de experiência em Simulink ou LabVIEW (Figura 5.1)?

a) INICIANTE - LABVIEW: Operadores Lógicos, Estruturas de Repetição, Estruturas Condicionais e Painéis de Visualização - SIMULINK: Criar, modificar e simular sistemas dinâmicos, modelagem no tempo contínuo e discreto, construir hierarquias no Simulink, criar componentes reutilizáveis em Simulink;

b) INTERMEDIÁRIO - LABVIEW: Sub VIs, Controle de IO via GPIB, USB, RS232 e porta Paralela - SIMULINK: Subsistemas, Controle de IO via GPIB, USB, RS232 e porta Paralela, Integração com C;

c) AVANÇADO - LABVIEW: Padrões de Projeto, Tratamento de Erros, Semáforos e Integração com outras linguagens - SIMULINK: Gerar código para sistemas embarcados, Integrar código Simulink com bibliotecas externas, Customizar código gerado, Deployment de código embarcado.

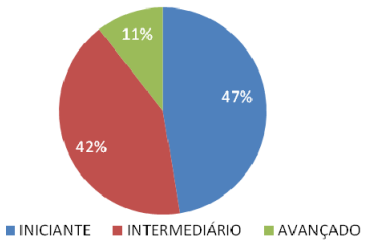


Figura 5.1 - Nível de Experiência dos Participantes da Pesquisa.

2 – Qual a sua área de atuação (Figura 5.2)?

a) Industrial

b) Acadêmico

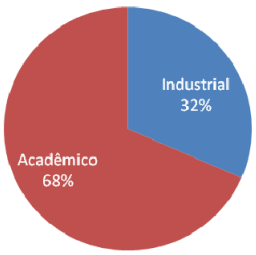


Figura 5.2 - Área de Atuação dos Participantes da Pesquisa.

3 – Os sistemas que você desenvolve estão no domínio de Sistemas Embarcados (Figura 5.3)?

a) Sim

b) Não

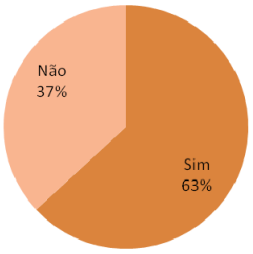


Figura 5.3 – Participantes que Desenvolvem Sistemas Embarcados.

4 – Marque o(s) item(s) que, antes de você desenvolver um sistema usando Simulink ou LabVIEW, você utiliza para fazer algum tipo de anotação sobre as características desse sistema (Figura 5.4).

a) Descrição informal

b) Diagrama de blocos

c) Relatório de requisitos

d) Outro tipo de registro

e) Não registro as informações

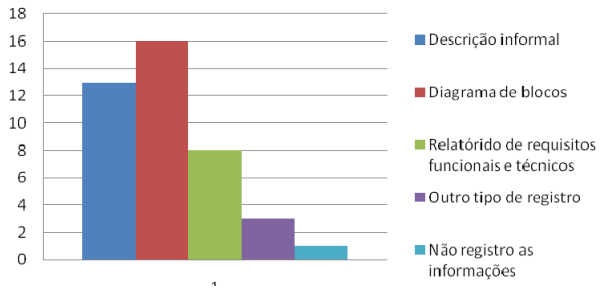


Figura 5.4 - Artefatos Utilizados pelo Participantes para Auxiliar o Desenvolvimento.

5 – Qual das abordagens você usa para desenvolver em Simulink ou LabVIEW (Figura 5.5)?

a) Você começa pelos subsistemas (Matlab) ou subVI (LabVIEW) e depois integra esses subsistemas ou subVI.

b) Você começa por partes maiores e depois detalha em subsistemas ou subVI (ao contrário da alternativa anterior).

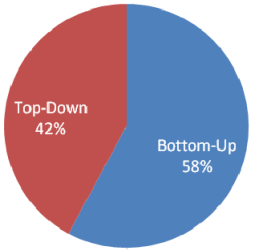


Figura 5.5 - Abordagens de Desenvolvimento pelos Participantes da Pesquisa.

6 – Marque o(s) item(s) desenvolvidos por outros autores que você já reutilizou no seu desenvolvimento em Simulink ou LabVIEW (Figura 5.6)?

a) Componentes

b) Bibliotecas

c) Diagramas

d) Não reutilizo algum desses itens

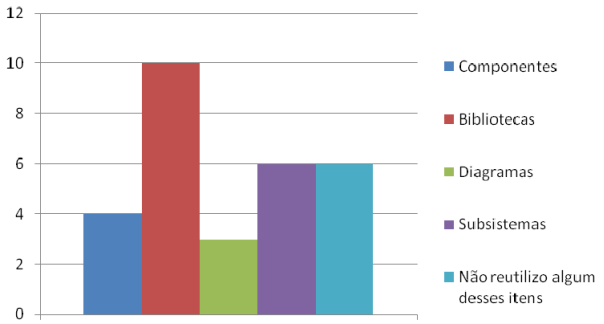


Figura 5.6 - Itens Reutilizados pelos Participantes da Pesquisa.

7 – Você procura organizar o modelo de forma que você e outros desenvolvedores consigam compreendê-lo para possíveis manutenções (Figura 5.7)?

a) Sim

b) Não

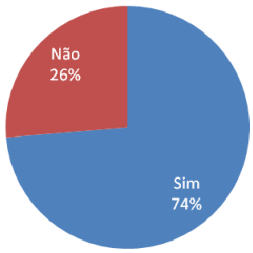


Figura 5.7 - Organiza O Modelo para Outros Desenvolvedores.

8 – Marque o(s) item(s) de representação gráfica que, ao usar o Simulink ou LabVIEW, você procura usar para auxiliar no desenvolvimento do seu sistema (Figura 5.8)?

a) Statecharts

b) Fluxogramas

c) Outros

d) Não utilizo representações gráficas

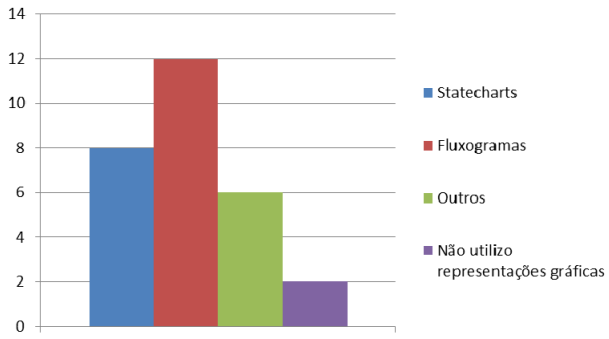


Figura 5.8 - Representações Gráficas Utilizadas pelos Participantes da Pesquisa.

9 – Você se preocupa com questões de precedência e paralelismo ao construir o modelo Simulink ou LabVIEW (Figura 5.9)?

a) Sim

b) Não

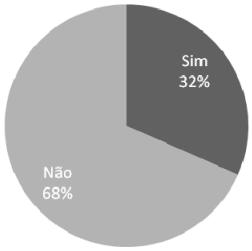


Figura 5.9 - Preocupação com Paralelismo.

10 – No modelo Simulink ou LabVIEW, você faz algum tipo de anotação extra para facilitar ou enriquecer tanto o seu quanto o entendimento de outros desenvolvedores (Figura 5.10)?

a) Sim

b) Não

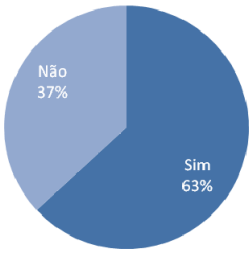


Figura 5.10 - Realiza Anotações Extras.

5.4 Avaliações e Análises

Com base na questão e nos resultados esperados, nesta seção os resultados provenientes do *survey* são avaliados.

Considerando que o objetivo é identificar se a técnica Statechart é utilizada pelos participantes do *survey* para apoiar o desenvolvimento de modelos Simulink, mas explorando as características dessa técnica de forma a representar o sistema em um nível alto de abstração e de representar uma documentação para esse sistema, a discussão dos resultados esperados retrata a seguinte cultura de desenvolvimento:

a) R.1.1 – verificar se os participantes utilizam alguma técnica para representar a aplicação em um nível mais alto de abstração do que o modelo Simulink:

Os resultados mostram que vários participantes utilizam algum tipo de técnica com essa finalidade. Mais especificamente, esses resultados mostram que as técnicas mais utilizadas são Fluxograma e Statechart, como apresentado na Figura 5.8.

b) R.1.2 – verificar se os participantes utilizam Statechart para auxiliar o desenvolvimento:

Os resultados do *survey* indicam que 42% dos entrevistados utilizam Statechart como representação gráfica para auxiliar o desenvolvimento de modelos Simulink ou LabVIEW (Figura 5.11).

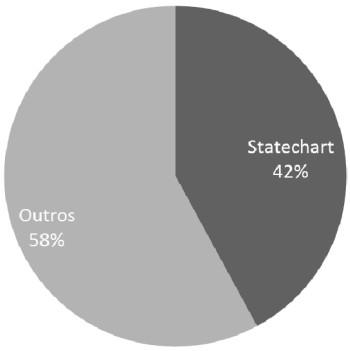


Figura 5.11 - Utilizam Statecharts.

c) R.1.3 – utilizam os recursos do Statechart de forma apropriada para levar à construção de modelos Simulink:

Os desenvolvedores de SE da pesquisa dizem utilizar os recursos da técnica Statechart para levar à construção de modelos Simulink. Porém, os resultados da pesquisa mostram que os recursos que a técnica oferece não são bem empregados para representar as funcionalidades antes do modelo Simulink ser construído.

Uma análise foi feita para verificar se os entrevistados utilizam a técnica Statechart de forma apropriada para levar a construção de modelos Simulink.

Na questão 9 da pesquisa foi elaborada a pergunta para verificar quantos dos desenvolvedores tem alguma preocupação com precedência e paralelismo ao desenvolver o modelo Simulink ou LabVIEW. A análise feita identificou que 63% dos

entrevistados fazem uso da técnica Statechart e não têm preocupação com precedência e paralelismo, como apresenta a Figura 5.13. Isso demonstra que a técnica Statechart não é empregada como um artefato que pode organizar as funcionalidades do modelo Simulink ou LabVIEW.

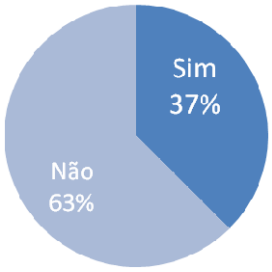


Figura 5.12 - Preocupação com Paralelismo.

5.5 Considerações Finais

Neste capítulo foi apresentada a terceira ação deste trabalho de pesquisa, sendo conduzida em paralelo com as outras duas iterações. Esta ação teve como objetivo realizar uma pesquisa entre os desenvolvedores de SE inseridos no contexto o INCT, para investigar a cultura de desenvolvimento desses desenvolvedores. Para conduzir a pesquisa foram definidas as ações descrevendo o objetivo, as questões e os resultados esperados seguindo a proposta da metodologia Pesquisa-Ação.

A pesquisa aplicada analisou as características dos desenvolvedores, tentando identificar se estes fazem uso de alguma forma de representação gráfica para auxiliar o desenvolvimento de SE. A partir dos resultados da pesquisa, identificou-se que parte dos participantes utiliza a técnica Statechart e as análises tiveram foco nesses entrevistados para verificar se a técnica é bem empregada para representar e organizar as funcionalidades que vão levar a construção dos modelos Simulink.

A partir dos resultados obtidos, verificou-se que os desenvolvedores de SE que fazem uso da técnica Statechart, como representação gráfica para auxiliar a construção de modelos Simulink, não é empregada de forma apropriada. Esta

conclusão foi obtida por meio de uma análise em que foram avaliadas a relação de entrevistados que utilizam a técnica Statechart e têm preocupação com precedência e paralelismo ao desenvolver modelos Simulink.

A análise realizada envolveu uma maioria de desenvolvedores acadêmicos, em que os SEs desenvolvidos variam bastante, desde membros do corpo humano robotizados, à veículos autônomos não tripulados. O que se verificou foi que apenas o modelo Simunlink implementado pelos desenvolvedores seria artefato suficiente para servir como documentação do sistema.

Assim, conclui-se, por meio dos resultados obtidos da pesquisa, que a técnica Statechart, quando utilizada como representação gráfica para auxiliar o desenvolvimento de modelos Simulink entre os desenvolvedores inseridos no contexto do INCT, não é empregada de forma apropriada da técnica. A técnica, quando bem empregada, permite representar as funcionalidades de forma organizada e descrevendo o comportamento do sistema.

Capítulo 6

CONCLUSÃO - REFLEXÕES E APRENDIZADOS DA PESQUISA-AÇÃO

Neste capítulo é descrita a etapa de Aprendizado e Reflexões da Pesquisa-Ação, que tem por objetivo apresentar as conclusões provenientes deste trabalho.

6.1 Considerações Iniciais

A partir das ações, avaliações e análises apresentadas nos Capítulos 3, 4 e 5, correspondentes às iterações da Pesquisa-Ação, este capítulo tem como objetivo apresentar as reflexões e aprendizados deste trabalho de pesquisa, referentes à aplicação da técnica UML Statechart para representar modelos Simulink em um nível mais alto de abstração.

O estudo aqui apresentado explorou a tendência em aplicar os recursos da Engenharia de Software para auxiliar no desenvolvimento de sistemas embarcados. O desenvolvimento de SE é uma atividade que tem se tornado cada vez mais complexa em decorrência da abrangência, cada vez maior, da área de atuação. Como a engenharia de software detém conhecimento sólido para o desenvolvimento de sistemas tradicionais, nota-se que a proposta que se percebe na literatura é adaptar as técnicas e métodos da engenharia de software para o desenvolvimento de SE.

As reflexões são apresentadas, de uma forma geral, relacionando as duas iterações de ações, que utilizaram como objeto de estudo duas aplicações distintas:

a mão robótica e o sistema de controle de refrigeração. A terceira ação, que foi uma pesquisa com desenvolvedores de SE, é relacionada com as reflexões das outras duas ações, apresentando uma reflexão relacionando as técnicas UML Statechart e Rede de Petri. Assim, são descritas as reflexões, limitações, trabalhos futuros e aprendizados adquiridos no com a condução dessa pesquisa.

O capítulo está organizado da seguinte forma: Na Seção 6.2 apresentam-se as reflexões do trabalho e na Seção 6.3 apresentam-se os aprendizados do trabalho.

6.2 Reflexões

Nesta seção são apresentadas as limitações e os trabalhos futuros referentes a este trabalho de pesquisa. As aplicações utilizadas como objeto de estudos enriqueceram a condução do trabalho e ofereceram subsídios para concluir sobre a viabilidade de aplicar a técnica UML Statechart como artefato para representar modelos Simulink em um nível mais alto de abstração, descrevendo o comportamento e as funcionalidades de forma organizada.

Assim, considerando que este trabalho está inserido em um contexto maior, que envolve outros trabalhos de mestrado e de doutorado, ele cumpriu seu objetivo de contribuir para a definição do processo a ser definido para dar suporte ao desenvolvimento de sistemas embarcados. Em uma primeira instância, pode-se dizer que modelos em UML Statechart são fortes candidatos a permanecer como artefatos que devem compor esse processo a ser definido.

Salienta-se que as ações só puderam ser realizadas como foram, pois se contou com o auxílio de desenvolvedores de SEs em todas elas. Sob o ponto de vista desses especialistas que contribuíram no trabalho, o modelo em UML Statechart ajuda significativamente tanto no caso de representar uma aplicação que já esteja modelada em Simulink, como também para desenvolver um modelo Simulink de forma mais organizada de uma aplicação cujo ponto de partida tenha sido um documento de requisitos.

Entretanto, a condução da pesquisa apresenta algumas limitações em relação à aplicação da técnica UML Statechart nas ações descritas neste trabalho. A seguir são apresentadas as limitações:

- A técnica UML Statechart oferece notações para serem exploradas como forma de representar informações adicionais que possam enriquecer o modelo. No entanto, informações muito específicas, mas ao mesmo tempo essenciais para se desenvolver o sistema de forma correta, não foram representadas no modelo UML Statechart. Esta conclusão foi definida entre os membros do grupo participantes da pesquisa para que o modelo não fosse sobrecarregado com muitas informações.
- Com base na restrição descrita anteriormente, mesmo com as informações especificadas, o modelo UML Statechart ainda requer um artefato adicional para descrever informações mais específicas, como equações, dados de entrada e saída, tabelas de dados. Essas informações são úteis para quem não é da área de desenvolvimento da aplicação.
- O grau de informação que um modelo UML Statechart vai representar depende do contexto aplicado. Por exemplo, no contexto da mão robótica, foi necessário representar informações de implementação, como informações do motor e cálculo de planejamento de trajetória. No contexto do sistema de controle de refrigeração, as funcionalidades foram mais simples não exigindo informações relativas à forma de implementação.
- Durante a condução das ações, como foram descritas nos Capítulos 3 e 4, os engenheiros desenvolvedores tiveram a tarefa de implementar o modelo Simulink tendo em mãos o modelo UML Statechart. Os resultados obtidos foram satisfatórios no que tange à organização, representação em nível mais alto de abstração e documentação. Entretanto, os engenheiros desenvolvedores já tinham um conhecimento prévio das aplicações implementadas que, consequentemente, facilitou o desenvolvimento dos modelos Simulink a partir do modelo UML Statechart.
- Neste trabalho não foram conduzidos experimentos com diferentes desenvolvedores de Simulink para avaliar se as informações contidas no modelo UML Statechart são suficientes.

- Não foram identificadas neste trabalho outras formas de representação gráfica para complementar o modelo UML Statechart. Este estudo é parte do trabalho de mestrado de outro integrante deste grupo de pesquisa.
- Não foram realizadas atividades de manutenção para avaliar o grau de manutenibilidade dos modelos Simulink construídos a partir do modelo UML Statechart.
- Não foram aplicados meios para avaliar o modelo Simulink original e reestruturado. Este estudo está sendo conduzido no trabalho de doutorado de outro aluno do grupo de pesquisa.
- Não foram realizados estudos mais aprofundados para realizar uma comparação mais significativa entre as técnicas UML Statechart e Rede de Petri. Os levantamentos apresentados relacionando as duas técnicas são provenientes de uma revisão da literatura.

Com base nas limitações mencionadas, podem-se relacionar alguns trabalhos futuros:

- Identificar artefatos que possam complementar o modelo UML Statechart para o desenvolvimento de sistemas embarcados.
- Identificar meios para avaliar quantitativamente o impacto que a reestruturação do modelo Simulink ocasionou em relação ao modelo original.
- Aplicar atividades de manutenção para avaliar o grau de manutenibilidade do modelo Simulink construído a partir do modelo UML Statechart.
- Avaliar a suficiência das informações do modelo UML Statechart por meio de desenvolvedores de Simulink de diferentes áreas de atuação.
- Realizar estudos mais aprofundados para comparar o uso das técnicas UML Statechart e Rede de Petri para o propósito investigado neste trabalho.

6.3 Aprendizados

Os principais aprendizados obtidos a partir desta pesquisa são:

- A metodologia Pesquisa-Ação foi de grande valia para a condução da pesquisa, oferecendo meios para que o objetivo do trabalho pudesse ser aplicado em iterações de ações diferentes.
- A técnica UML Statechart mostrou-se ser uma forte candidata para compor um dos artefatos a serem construídos no processo de engenharia avante, investigado pelo projeto em que este trabalho está inserido. A técnica permitiu apoiar, documentar e organizar as funcionalidades que irão compor um modelo Simulink.
- Apesar da técnica Statechart ser bastante difundida para representar o comportamento de sistemas, verificou-se, por meio do *survey*, que poucos fazem uso apropriado da técnica. As análises do *survey* mostraram que os desenvolvedores não exploram suas características, de forma a apoiar e organizar o desenvolvimento de SEs.
- Os engenheiros desenvolvedores de SE mostraram ter uma dificuldade em utilizar documentos textuais muito extensos para especificar os requisitos. Por isso, o grupo propõe elaborar um processo que contenha um conjunto simples de artefatos pode caracterizar um suporte relevante para ser utilizado pelos desenvolvedores de SE.

REFERÊNCIAS BIBLIOGRÁFICAS

BARR, M. **Programming Embedded Systems in C and C++**. 1ª. ed. Sebastopol: O'Reilly Media, Inc., 1999. 301 p.

BICCHI, A.; KUMAR, V. Robotic Grasping and Manipulation. In: **Lecture Notes in Control and Information Sciences**. Berlin, Heidelberg: Springer Berlin Heidelberg, v. 270, 2001. Cap. 3, p. 55-74.

BOWEN, J.; STAVRIDOU, V. Safety-critical systems, formal methods and standards. **Software Engineering Journal**, 8, n. 4, Jul 1993. 189-209.

BRISOLARA, L. B. et al. **Using UML as a front-end for an efficient Simulink-based multithread code generation targeting MPSoCs**. International UML DAC Workshop: UML for SOC Design. San Diego, USA: [s.n.]. 2007. p. 1-6.

CAURIN, G. A. P.; ALBUQUERQUE, A. R. L.; MIRANDOLA, A. L. A. **Manipulation strategy for an anthropomorphic robotic hand**. Proceedings of IEEE/RSJ International Intelligent Robots and Systems. Sendai, Japan: IEEE Press. 2004. p. 1656-1661.

DOUGLASS, B. P. **Doing hard time:** Developing Real-time Systems with UML, Objects, Frameworks, and Patterns. 1ª. ed. Upper Saddle River: Addison-Wesley Professional, v. 1, 1999. 766 p.

EBERT, C.; JONES, C. Embedded Software: Facts, Figures, and Future. **Computer**, 42, n. 4, April 2009. 42-52.

EBERT, C.; SALECKER, J. Guest Editors' Introduction: Embedded Software - Technologies and Trends. **IEEE Software**, 26, n. 3, Maio 2009. 14-18.

ESPINOZA, H. et al. **Challenges in Combining SysML and MARTE for Model-Based Design of Embedded Systems**. Proceedings of the 5th European Conference on Model Driven Architecture - Foundations and Applications. Berlin: Springer-Verlag. 2009. p. 98-113.

FARKAS, T.; NEUMANN, C.; HINNERICHS, A. **An Integrative Approach for Embedded Software Design with UML and Simulink**. Proceedings of the 2009 33rd Annual IEEE International Computer Software and Applications Conference. Washingto, DC: IEEE Computer Society. 2009. p. 516-521.

FREIRE, G. M.; PEDRO, L. P.; ANTONIO, E.; NEPOMUCENO, J.; CAURIN, G.; FABBRI, S. **Applying Reengineering on Simulink Model Assisted by UML Statechart**. Conferência Brasileira de Sistemas Embarcados Críticos. São Carlos: [s.n.]. 2011. p. 6.

GANSSLE, J. A. **The Art of Designing Embedded Systems**. 2ª. ed. Burlington: Newnes, 2008. 298 p.

HAREL, D. Statecharts: A visual formalism for complex systems. **Science of Computer Programming**, 8, n. 3, Jun 1987. 231-274.

HAREL, D.; PNUELI, A. On the development of reactive systems. In: **Logics and models of concurrent systems**. [S.l.]: Springer-Verlag New York, Inc., 1985. p. 477-498.

HAREL, D.; POLITI, M. **Modeling Reactive Systems with Statecharts:** The Statemate Approach. 1ª. ed. [S.l.]: McGraw-Hill, 1998. 258 p.

INSTITUTO Nacional de Ciência e Tecnologia - Sistemas Embarcados Críticos. **inct-sec**, 2011. Disponivel em: <http://www.inct-sec.org/>. Acesso em: 20 nov. 2011.

ITO, S. A.; CARRO, L.; JACOBI, R. P. Making Java Work for Microcontroller Applications. **IEEE Design & Test of Computers**, Los Alamitos, CA, 18, n. 5, Setembro 2001. 100-110.

KONRAD, S.; CHENG, B. H. C.; CAMPBELL, L. A. Object Analysis Patterns for Embedded Systems. **IEEE Transactions on Software Engineering**, Piscataway, NJ, 30, n. 12, Dezembro 2004. 970-992.

LEE, E. A. **Model-Driven Development, From Object-Oriented Design to Actor-Oriented Design**. Workshop on Software Engineering for Embedded Systems: From Requirements to Implementation. Chicago, EUA: [s.n.]. 2003.

LEE, E. A.; LIU, X.; NEUENDORFFER, S. Classes and Inheritance in Actor-Oriented Design. **ACM Transactions on Embedded Computing Systems**, New York, EUA, 8, n. 4, Jul 2009. 1-26.

LEE, E. A.; NEUENDORFFER, S. **Actor-oriented models for codesign:** balancing re-use and performance. Formal methods and models for system design. Norwell, EUA: Kluwer Academic Publishers. 2004. p. 33-56.

LEE, E. A.; NEUNDORFFER, S.; WIRTHLIN, M. J. Actor-oriented design of embedded hardware and software systems. **Journal of Circuits, Systems, and Computers**, 12, 2003. 231-260.

LI, Q.; YAO, C. **Real-Time Concepts for Embedded Systems**. Lawrence: CMP Books, 2003. 294 p.

LIGGESMEYER, P.; TRAPP, M. Trends in Embedded Software Engineering. **IEEE Software**, 26, n. 3, Maio 2009. 19-25.

MAGICDRAW. **MagicDraw - Architecture Made Simple**, 2011. Disponivel em: <http://www.magicdraw.com>. Acesso em: 06 Agosto 2011.

MARWEDEL, P. **Embedded System Design**. 2ª. ed. Dordrecht: Springer, 2006. 258 p.

MATHWORKS. **MathWorks**, 2011. Disponivel em: <http://www.mathworks.com/products/simulink/description1.html>. Acesso em: 07 Junho 2011.

MODEL Driven Architecture. **MDA**, 2011. Disponivel em: <http://www.omg.org/mda/>. Acesso em: 17 jun. 2011.

MURATA, T. **Petri nets:** Properties, analysis and applications. Proceedings of the IEEE. [S.l.]: [s.n.]. 1989. p. 541-580.

MURRAY, A. T.; SHAHABUDDIN, M. **OO techniques applied to a real-ti embedded, spaceborne application**. ACM SIGPLAN International Conference on Object-Oriented Programming Systems, Languages, and Applications. Portland, USA: ACM. 2006. p. 830-838.

NATIONAL Instruments. **NI**, 2011. Disponivel em: <http://www.ni.com/labview/>. Acesso em: 17 jun. 2011.

NOERGAARD, T. **Embedded Systems Architecture:** A Comprehensive Guide for Engineers and Programmers. Burlington: Newnes, 2005. 640 p.
OBJECT Management Group. **OMG**, 2011. Acesso em: 17 jun. 2011.

OMGMARTE. **UML MARTE**, 2011. Disponivel em: <http://www.omgmarte.org/>. Acesso em: 17 agosto 2011.

PRESSMAN, R. S. **Engenharia de Software**. 6ª. ed. São Paulo: McGraw-Hill, 2006. 720 p.

PTOLOMY Project. **Ptolomy Project**, 2011. Disponivel em: <http://ptolemy.eecs.berkeley.edu/>. Acesso em: 17 jun. 2011.

RAJAGOPAL, R. et al. A Rapid Prototyping Tool for Embedded, Real-Time Hierarchical Control Systems. **Eurasip Journal on Embedded Systems**, 2008, 2008. 1-15.

SANTOS, P. S. M. D.; TRAVASSOS, G. H. Action Research Can Swing the Balance in Experimental Software Engineering. In: ZELKOWITZ, M. V. **Advances in Computers**. Burlington: Academic Press, v. 83, 2011. p. 205-276.

SCITOOLS. **Understand Your Code - Source Code Analysis & Metrics**, 2011. Disponivel em: <http://www.scitools.com/>. Acesso em: 06 Agosto 2011.

SIMULINK® Getting Started Guide. [S.l.]: The MathWorks, Inc., 2010. p. 81.

SOMMERVILLE, I. **Engenharia de Software**. 6ª. ed. São Paulo: Addison Wesley, 2003. 592 p.

STANKOVIC, J. A. Misconceptions about real-time computing: a serious problem for next-generation systems. **Computer**, October 1988. 10-19.

STANKOVIC, J. A. Real-time and embedded systems. **ACM Computing Surveys**, 28, n. 1, 1996. 205-208.

STRINGER, E. T. **Action Research**. 3ª. ed. [S.l.]: SAGE, 2007. 279 p.

SYSTEMC. **Open SsystemC Initiative:** Defining & Advancing SystemC Standards, 2011. Disponivel em: <http://www.systemc.org/about/>. Acesso em: 17 jun. 2011.

TROWITZSCH, J.; ZIMMERMANN, A. **Real-time UML state machines:** An analysis approach. In Workshop Object Oriented Software Design for Real Time and Embedded Computer Systems. [S.l.]: [s.n.]. 2005.

TROWITZSCH, J.; ZIMMERMANN, A. **Using UML state machines and petri nets for the quantitative investigation of ETCS**. Proceedings of the 1st International Conference on Performance Evaluation Methodolgies and Tools. Pisa, Itália: ACM. 2006.

TROWITZSCH, J.; ZIMMERMANN, A.; HOMMEL, G. Towards quantitative analysis of real-time UML using stochastic petri nets. **International Parallel and Distributed Processing Symposium**, Los Alamitos, EUA, 3, 2005. 2-9.

TU-ILMENAU. **Faculty of Computer Science and Automation System and Software Engineering**, 2011. Disponivel em: <http://www.tu-ilmenau.de/fakia/Home.8078.0.html?&L=1>. Acesso em: 06 Agosto 2011.

UNIFIED Modeling Language. **UML**, 2011. Disponivel em: <http://www.uml.org/>. Acesso em: 16 jun. 2011.

VERILOG. **Verilog DOT COM**, 2011. Disponivel em: <http://www.verilog.com/>. Acesso em: 17 jun. 2011.

WANG, Y. et al. **A MDA-based approach for general embedded software simulation platform**. Proceedings of the 2009 International Conference on Scalable

Computing and Communications; Eighth International Conference on Embedded Computing. Dalian, China: [s.n.]. 2009. p. 20-25.

WEHRMEISTER, M. A. et al. **An object-oriented platform-based design process for embedded real-time systems**. An Object-Oriented Platform-based Design Process for Embedded Real-Time Systems Distributed Computing. Washington, DC: IEEE Computer Society. 2005. p. 125-128.

Apêndice A

PESQUISA-AÇÃO

Neste apêndice contém um resumo proveniente do trabalho apresentado por Santos e Travassos (2011), que descreve a estratégia de desenvolvimento de trabalho científico abordada nesta pesquisa, denominado Pesquisa-Ação.

A.1 Estratégia de Desenvolvimento de Trabalho Científico

A Engenharia de Software vem introduzindo estratégias de pesquisa que possam dar fundamentação para entender os limites e aplicabilidade das tecnologias de software. A experimentação é uma abordagem que visa, por meio da investigação, entender a prática da engenharia de software. Atualmente, essa abordagem tem procurado ampliar e intensificar os resultados da pesquisa por meio da aplicação de estudos de caso, etnografias, *surveys* e técnicas de entrevistas. Os resultados científicos obtidos a partir da aplicação da experimentação servem como base no desenvolvimento de conceitos para serem aplicadas no contexto industrial, bem como estabelecer fundamentação teórica para melhor compreensão do fenômeno estudado no contexto acadêmico.

Diversas empresas ainda têm como base a opinião de especialistas; porém, essas opiniões nem sempre são confiáveis, pois nem sempre envolvem resultados científicos. Isso faz com que tecnologias de software sejam adaptadas a determinadas áreas sem base científica ou critérios bem fundamentados.

Uma das formas de se aplicar a base científica é por meio da Pesquisa-Ação, que é uma metodologia que também pode ser aplicada à prática da engenharia de

software, a qual possibilita intensificar a execução de estudos relevantes e a obtenção de resultados significativos. As características dessa metodologia permite que seja executada, simultaneamente, a pesquisa e a ação, com maior aproximação do cenário real, pois envolve situações que caracterizam situações reais da indústria, como meios para investigar os resultados da ação.

Na seção seguinte é descrito um processo de pesquisa comum e, por meio desse exemplo, é explicado como funciona um processo de pesquisa utilizando a Pesquisa-Ação.

A.2 Processo de Pesquisa-Ação

Para melhor entender como funciona a metodologia de Pesquisa-Ação, é descrito como uma metodologia de pesquisa comum ocorre, para servir de comparativo.

Na metodologia comum, inicia-se organizando um conjunto coeso de idéias sobre um fenômeno a ser estudado que irá compor um framework conceitual **F**. **F** é aplicado seguindo uma metodologia **M** para investigar uma área de interesse **A**. Considerando como exemplo o trabalho de pesquisa em questão, a analogia seria a seguinte: deseja-se explorar modelos mais alto nível para guiar o desenvolvimento de modelos Simulink (**A**) aplicando a engenharia reversa e a reengenharia (**M**) através da utilização da técnica UML Statechart (**F**).

A Figura A.1 descreve a representação de **M** na metodologia do processo de pesquisa comum em que **F** é aplicado em **A**. Em um processo de pesquisa comum o estudo é validado a partir da ocorrência repetitiva dos resultados desejados para que se possa provar o sucesso da hipótese. Entretanto, a desvantagem é que o experimento é aplicado em um cenário limitado que simula o mundo real.

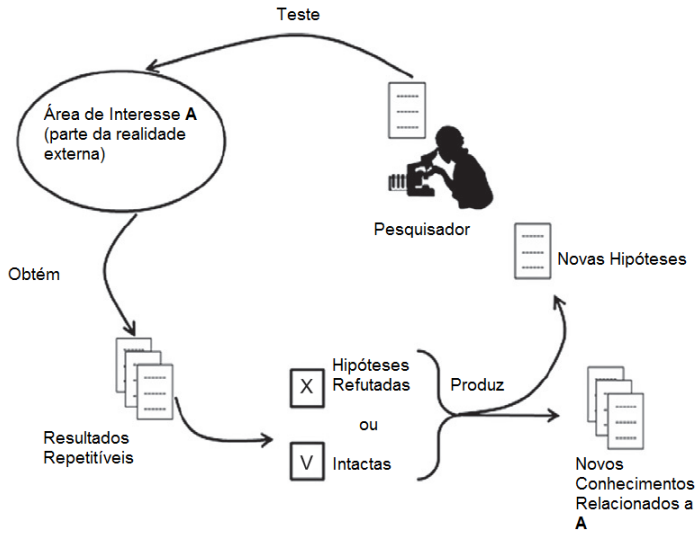


Figura A.1 - Processo de Pesquisa Comum. Representação de M (SANTOS e TRAVASSOS, 2011)

O processo de pesquisa envolvendo a metodologia Pesquisa-Ação permite que os papéis desempenhados por **F**, **M** e, inclusive, **A**, possam ser alterados durante a execução da pesquisa. Isso ocorre, pois durante a execução da pesquisa, as implicações de **F** e **M** são avaliadas. Tanto o framework **F** como a metodologia **M** são declarados e aplicados em estudos de caso reais relacionados à **A**. Como resultado, a Pesquisa-Ação não só apresenta **M** em um ambiente natural, mas também requer um protocolo de pesquisa relacionado à **M** com fundamentos conceituais práticos e científicos a partir de **F**. Isso permite que o processo de pesquisa seja refeito e submetido a uma análise mais crítica.

Em um processo de Pesquisa-Ação, **F** e **M** são declarados como entrada em uma primeira iteração os quais são aplicados em uma situação real e então as ações são avaliadas podendo levar a descobertas e a novos temas de pesquisa. A Figura A.2 descreve como um processo de pesquisa-ação ocorre. O processo que descreve a pesquisa-ação permite que etapas anteriores possam ser reavaliadas compondo um ciclo iterativo. As etapas são:

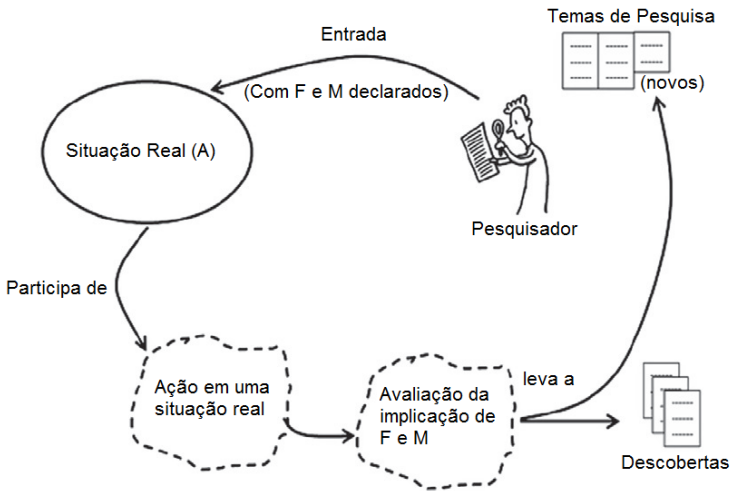


Figura A.2 – Aplicação da Pesquisa-Ação (SANTOS e TRAVASSOS, 2011).

- ***Diagnóstico*** – explora a área de pesquisa, as partes interessadas e suas expectativas holisticamente;
- ***Planejamento*** – define as ações de acordo com as circunstâncias, ou seja, retrata as suposições formuladas pelo pesquisador sobre possíveis soluções e resultados, sendo que estas devem estar de acordo com as formulações teóricas científicas;
- ***Intervenção*** – implementa as ações planejadas, aplicando técnicas que possam ser utilizadas para examinar, discutir e tomar decisões sobre o processo investigativo;
- ***Avaliação*** – analisa os efeitos gerados pela intervenção, considerando a fundamentação teórica como base das ações definidas;
- ***Reflexão*** – dissemina o conhecimento adquirido entre os participantes da pesquisa.

As etapas definidas são executadas ciclicamente permitindo iterações e adaptações entre elas, conforme pode ser observado na Figura A.3.

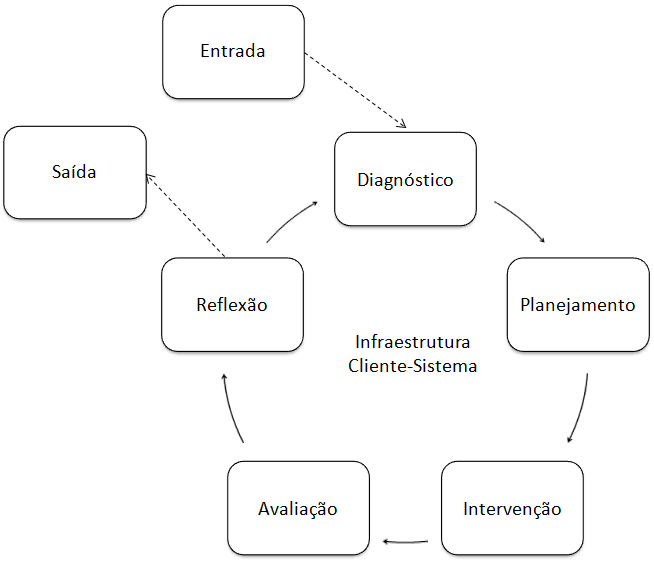


Figura A.3 - Processo Canônico de Pesquisa-Ação (SANTOS e TRAVASSOS et al., 2011)

Na seção seguinte são descritos os paradigmas de pesquisa científica e como a pesquisa-ação está inserida nestes paradigmas.

A.3 Relação da Pesquisa-Ação com Outros Paradigmas de Pesquisa Científica

Para melhor entender a metodologia Pesquisa-Ação, são descritos os quatro paradigmas predominantes em uma pesquisa científica:

- ***Positivismo*** – o positivismo considera que todo conhecimento tem fundamentação em inferências lógicas a partir de fatos observados. Ele é considerado minimalista, pois os fenômenos estudados são divididos em partes mais simples. O conhecimento científico pode ser elaborado de forma incremental, a partir das observações verificadas e as inferências baseadas nessas observações. Esse paradigma está mais associado à estratégia de estudo controlado, podendo ser conduzido por *surveys* e estudos de caso;

- ***Construtivismo*** – esse paradigma rejeita a idéia de separar o conhecimento científico do contexto humano. Tem como objetivo reduzir verificações teóricas e aumentar o entendimento sobre a compreensão individual em relação a um fenômeno estudado e atribuição racional dos seus atos. Diversas teorias podem surgir durante o paradigma do construtivismo, mas sempre associadas ao contexto de estudo. Esse paradigma está mais associado à etnografia, podendo ser explorado em estudos de caso, Pesquisa-Ação e *surveys*;
- ***Teoria Crítica*** – Considera que o conhecimento científico tem a capacidade de romper o raciocínio imposto pela doutrina acadêmica. Defende que o pesquisador e o objeto pesquisado possuem uma conexão em que, o conhecimento do pesquisador, inevitavelmente, influencia a pesquisa. A Pesquisa-Ação é a estratégia que melhor define este paradigma;
- ***Pragmatismo*** – Reconhece que todo conhecimento é incompleto, podendo ser estimado até certo ponto e os resultados obtidos dependem dos métodos utilizados. A escolha deste paradigma é feita julgando o quanto este paradigma é útil para a resolução prática dos problemas. Enfatiza a importância de um consenso, como um avaliador externo da objetividade. Nesse paradigma a melhor estratégia aplicada, quando não há nenhuma solução previamente conhecida, é a metodologia Pesquisa-ação.

Portanto, a Pesquisa-Ação é considerada uma opção em três dos quatro paradigmas: no Construtivismo o pesquisador apresenta uma função de observador; na Teoria e Crítica o pesquisador estabelece uma opinião sobre o fenômeno estudado; e no Pragmatismo o pesquisador soluciona o problema a partir de uma instância mais ativa.

Os paradigmas de pesquisa científica são classificados nas categorias quantitativa e qualitativa dependendo da coleta e organização do conhecimento adquirido. O paradigma do Positivismo está mais relacionado à categoria quantitativa, enquanto os outros três estão relacionados à categoria qualitativa.

A categoria quantitativa tem por objetivo a medição e análise a partir das relações entre variáveis, representando as características do objeto observado. A característica objetiva e minimalista dessa categoria é considerada como a principal desvantagem, já que ela reduz os detalhes descritivos das características e propriedades do mundo real.

A categoria qualitativa visa a caracterizar o objeto de estudo que está sendo investigado por meio de entrevistas e observações; enfatizar as descrições detalhadas; e apresentar os dados quantitativos que foram obtidos através de gráficos e imagens. A Figura A.4 apresenta a relação entre os quatro paradigmas e as categorias qualitativa e quantitativa.

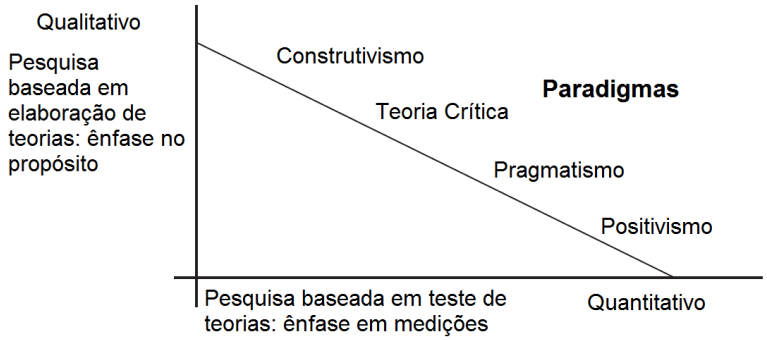


Figura A.4 - Relação entre Os Paradigmas de Pesquisa e As Abordagens Quantitativas e Qualitativas (SANTOS e TRAVASSOS, 2011).

Assim, uma vez que a Pesquisa-Ação é uma metodologia que pode ser aplicada nos paradigmas Pragmatismo, Teoria e Crítica e Case ao propósito que levou à realização da pesquisa, ao invés de medidas quantitativas que sejam capazes de refutar ou corroborar uma hipótese inicial.