UNIVERSIDADE FEDERAL DO CEARÁ

CENTRO DE CIÊNCIAS

DEPARTAMENTO DE COMPUTAÇÃO

MESTRADO E DOUTORADO EM CIÊNCIA DA COMPUTAÇÃO

ÍTALO LINHARES DE ARAÚJO

EVOLUINDO O MÉTODO CHAPTER EM DIREÇÃO À GERAÇÃO DE CASOS DE TESTE

FORTALEZA

2014

ÍTALO LINHARES DE ARAÚJO

EVOLUINDO O MÉTODO CHAPTER EM DIREÇÃO À GERAÇÃO DE CASOS DE TESTE

Dissertação apresentada ao Curso de do Mestrado e Doutorado em Ciência da Computação do Centro de Ciências da Universidade Federal do Ceará, como requisito parcial à obtenção do título de mestre em Ciência da Computação. Área de Concentração: Engenharia de Software

Orientadora: Profa. Dra. Rossana Maria de Castro Andrade

Co-Orientador: Prof. Dr. Pedro de Alcântara dos Santos Neto

FORTALEZA

2014

Dados Internacionais de Catalogação na Publicação
Universidade Federal do Ceará
Biblioteca Universitária
Gerada automaticamente pelo módulo Catalog, mediante os dados fornecidos pelo(a) autor(a)

A692e Araújo, Ítalo Linhares de.
Evoluindo o método ChAPTER em direção à geração de casos de teste / Ítalo Linhares de Araújo. – 2014.
106 f. : il. color.

Dissertação (mestrado) – Universidade Federal do Ceará, Centro de Ciências, Programa de Pós-Graduação em Ciência da Computação, Fortaleza, 2014.
Orientação: Profa. Dra. Rossana Maria de Castro Andrade.
Coorientação: Prof. Dr. Pedro de Alcântara dos Santos Neto.

1. Teste de Software. 2. Linha de produtos de software. 3. Aplicações sensíveis ao contexto. 4. Linha de produtos de software sensíveis ao contexto. 5. Caso de uso textual. I. Título.

CDD 005

ÍTALO LINHARES DE ARAÚJO

EVOLUINDO O MÉTODO CHAPTER EM DIREÇÃO À GERAÇÃO DE CASOS DE TESTE

Dissertação apresentada ao Curso de do Mestrado e Doutorado em Ciência da Computação do Centro de Ciências da Universidade Federal do Ceará, como requisito parcial à obtenção do título de mestre em Ciência da Computação. Área de Concentração: Engenharia de Software

Aprovada em: 07 de março de 2014

BANCA EXAMINADORA

Profa. Dra. Rossana Maria de Castro Andrade (Orientadora)
Universidade Federal do Ceará (UFC)

Prof. Dr. Pedro de Alcântara dos Santos Neto (Co-Orientador)
Universidade Federal do Piauí (UFPI)

Prof. Dr. Pedro Porfírio Muniz Farias
Universidade de Fortaleza (Unifor)

Prof. Dr. Reinaldo Bezerra Braga
Universidade Federal do Ceará (UFC)

Prof. Dr. Windson Viana de Carvalho
Universidade Federal do Ceará (UFC)

À minha amada mãe, Adriana, ao meu pai, Francisco, e ao meu irmão, Igor.

AGRADECIMENTOS

Agradeço primeiramente ao Pai, Inteligência Suprema e Causa primeira de todas as coisas, pela oportunidade de estar terminando mais uma etapa na minha vida. Agradeço, também, por ter colocado pessoas maravilhosas que acompanharam minha caminhada e que contribuíram de alguma forma para que essa etapa fosse concluída. Assim, peço a Ti que converta em bençãos todo auxílio e aprendizado proporcionado por essas pessoas:

Minha amada mãe, pela oportunidade preciosa da vida e por, desde esse momento, batalhou para que eu chegasse até aqui. Ela que é meu exemplo de dedicação, amor, cuidado, zelo pelos filhos. À ela, Senhor, todo o Amor que houver nesse mundo.

Meu adorado pai, que apesar da distância, sempre se fez presente na minha vida. Ele que sempre apoiou minhas decisões e que também contribuiu para que essa etapa fosse concluída.

Meu dedicado irmão, que um dia disse que queria ter uma vida semelhante a minha e hoje possui exatamente o que ele desejou, só tenho a agradecer a palavra amiga, o conforto nas horas difíceis, as mensagens edificantes e o amor dedicado nesses dois anos em que me encontro fisicamente distante dele.

Minha cunhada e irmã, Kalina, que junto com meu irmão dedicou momentos para me auxiliar e me dar conforto durante esses anos em que me encontro aqui em Fortaleza.

Meus tios e padrinhos, Armando e Claudia, que com muito amor me receberam no seu lar e cuidam de mim como um filho. Meus primos, Armando Filho e Amanda, por ter me recebido da maneira como sempre nos tratamos: como irmãos. A Érica que com zelo cuida de mim, e o Armando Neto, pela alegria que ele traz a todos nós.

Aos demais familiares, tios, primos, avós, incluindo os meus avós paternos que se encontram em um local ao teu lado Senhor, que apoiaram minha decisão e que confiaram em mim.

Minha orientadora Rossana, pela experiência que me proporcionou ao confiar em mim e que me ensinará muito mais durante os próximos 4 anos.

Meu co-orientador Pedro, que desde a gradução acompanha meus passos na área acadêmica, e que desde essa época me ensina a ser dedicado ao trabalho e a como ser um bom profissional.

Meu amigo, co-co-orientador, Ismayle, que, sem o auxílio dele durante esses dois anos de mestrado não seria possível terminar essa etapa. Sua esposa Paula, que também me

apoiou nesta etapa.

Meus amigos que caminharam diariamente comigo nesses dois anos, Andressa, Candré, Christiano, Nayane, Paulo Artur, Rafael, Rainara, Thalisson, Zezim, pois a cada momento ao lado deles eu ganhava força para continuar essa etapa. Meus amigos de Teresina, que apesar da distância sempre me apoiaram para continuar essa etapa.

Meus amigos da Federação Espírita Piauiense, Gracelcia, Hinália, Lísnia, Rosário, Pablo, Antônio Carlos, Francisco, Irismar e os demais que me deram apoio, coragem, palavras amigas e fortalecedoras nos momentos que precisei.

Todos do MDCC, do GREat e da CTQS e que de alguma forma contribuíram para que esse momento chegasse.

Todos da CAPES, que me proporcionou a bolsa para que concluísse essa etapa.

"A maior ignorância do homem é não saber amar."

(Autor Desconhecido)

RESUMO

Uma Linha de Produtos de Software (LPS) facilita o desenvolvimento de aplicações de um mesmo domínio, pois permite a reutilização de artefatos. Uma LPS Sensível ao Contexto (LPSSC), por sua vez, tem como objetivo desenvolver aplicações que mudam o seu comportamento dinamicamente com base em informações de contexto. Um dos problemas associado à LPSSC é a garantia da qualidade, pois a complexidade é maior do que em uma aplicação tradicional, uma vez que uma LPSSC agrega os desafios inerentes tanto à linha de produtos quanto às aplicações sensíveis ao contexto. O teste de software é uma das formas de garantir a qualidade e para reduzir os custos envolvidos na etapa dos testes, a geração automática de casos de testes é uma solução utilizada na literatura. Essa geração pode ser feita com o uso de um Caso de Uso Textual (CUT). Entretanto, há desafios, pois um CUT é descrito em linguagem natural (LN), o que pode levar a ambiguidade e imprecisão na sua descrição, dificultando a geração dos casos de testes. Para resolver o problema do uso de uma LN para descrever um CUT, o vocabulário e a gramática de uma linguagem natural podem ser restringidos, levando à utilização de uma Linguagem Natural Controlada (LNC). Na literatura foi encontrada apenas uma proposta para automatizar os testes para LPSSC a partir de casos de uso, o método ChAPTER, o qual utiliza linguagem natural para descrever um CUT. Diante dos problemas citados ao utilizar um CUT descrito em LN, este trabalho primeiro propõe uma LNC, denominada CARNAUbA, para auxiliar na descrição e na identificação de informações de um CUT a serem utilizadas nos testes. Em seguida, este trabalho estende o método ChAPTER e a ferramenta que o implementa de modo a permitir o uso da CARNAUbA para gerar a estrutura necessária dos testes para LPSSC em direção à execução deles. Para verificar a viabilidade da extensão proposta no método e implementada na ferramenta, duas linhas de produtos de software são utilizadas como estudos de caso e é verificado se os testes gerados se encontram condizentes com os casos de uso utilizados.

Palavras-chave: Teste de software. Linha de produtos de software. Aplicações sensíveis ao contexto. Linha de produtos de software sensíveis ao contexto. Caso de uso textual.

ABSTRACT

A Software Product Line (SPL) facilitates the development of applications within the same domain as it enables the reuse of artifacts. A Context-Aware Software Product Line (CASPL), in turn, aims to develop applications that change their behavior dynamically based on context information. One of the problems associated with CASPL is the quality assurance, because its complexity is greater than a traditional application since a CASPL adds the challenges inherent to both the software product line and the context-aware applications. Software testing is one of the ways to ensure quality and to reduce the costs involved in the testing phase, the automatic generation of test cases is a solution found in the literature. This generation can be done using a Textual Use Case (TUC). However, there are challenges involved, because a CUT is described in natural language (NL), which can lead to ambiguity and imprecision in its description, making it difficult to generate test cases. To solve the problem of using an NL to describe a TUC, the vocabulary and the grammar of a natural language can be restricted, leading to the use of a Controlled Natural Language (CNL). In the literature, only one proposal was found to automate the tests for LPSSC from use cases, the ChAPTER method, which uses natural language to describe a TUC. Given the problems mentioned before when using a CUT described in NL, this work first proposes a CNL, called CARNAUbA, to help in the description and identification of the TUC information to be used in the tests. This work then extends the ChAPTER method and the tool that implements it to allow the use of CARNAUbA to generate the necessary structure of the tests for LPSSC to execute them. To verify the feasibility of the proposed extension in the method and implemented in the tool, two software products lines are used as case studies, and it is checked if the tests generated are in accordance with the use cases.

Keywords: Software testing. Software product line. Context-aware applications. Context-aware software product line. Textual use case.

LISTA DE FIGURAS

Figura 1 – Arquitetura de referência para uma aplicação sensível ao contexto 23
Figura 2 – Ciclo de desenvolvimento de uma LPS 26
Figura 3 – Exemplo de features obrigatórias e opcionais da linha Mobiline 28
Figura 4 – Exemplos de features alternativas da linha Mobiline 29
Figura 5 – Exemplo de cenário de teste para o GREat Tour 33
Figura 6 – Exemplo de caso de teste para o GREat Tour 33
Figura 7 – Exemplo de procedimento de teste para o GREat Tour 34
Figura 8 – Caso de Uso "Autenticação" baseado no template de (SANTOS, 2013) . . . 40
Figura 9 – Funcionamento da proposta de [Nebut et al. 2006] 43
Figura 10 – Exemplo de elementos do trabalho de (GOIS *et al.*, 2010) 44
Figura 11 – Template de caso de uso definido em (SANTOS, 2013) 48
Figura 12 – Cenário de teste gerado pelo ChAPTER 49
Figura 13 – Processo de análise semântica 63
Figura 14 – Caso de Uso Play Brickles 67
Figura 15 – Visão Geral da extensão do ChAPTER 74
Figura 16 – Tela da ferramenta que exibe informações da CARNAUbA 76
Figura 17 – Inserção de detalhes para as entradas 76
Figura 18 – Diagrama de Pacotes 77
Figura 19 – Lista de Linhas cadastradas 82
Figura 20 – Definição de um passo para o caso de uso Mostra Textos 82
Figura 21 – Extração das informações do passo 1 do caso de uso Mostra Textos 82
Figura 22 – Tela para inserir mais informações para uma entrada 84
Figura 23 – Teste gerado na FitNesse 87
Figura 24 – Fixture gerada para o caso de uso Mostra Textos 89
Figura 25 – Caso de uso "Play Bowling" 90
Figura 26 – Caso de teste na FitNesse para o caso de uso "Play Bowling" 92
Figura 27 – Fixture gerada para o caso de uso "Play Bowling" 94

LISTA DE TABELAS

Tabela 1 – Comparação entre uma Linguagem Natural e uma LNC 42
Tabela 2 – Comparação entre os métodos de teste 49
Tabela 3 – Definição da sintaxe do elemento “Nome do caso de uso” 52
Tabela 4 – Definição da sintaxe do elemento “Ponto de extensão” 53
Tabela 5 – Definição da sintaxe do elemento “Categoria de Reuso” 54
Tabela 6 – Definição da sintaxe do elemento “Restrição de Contexto” 54
Tabela 7 – Definição da sintaxe do elemento “Atores” 55
Tabela 8 – Definição da sintaxe do elemento “Ponto de Variação e Variante” 57
Tabela 9 – Definição da sintaxe do elemento “Número do Passo” 57
Tabela 10 – Definição da sintaxe de uma “Ação” do elemento “Passo” 59
Tabela 11 – Definição da sintaxe de uma “Ponto de Extensão” do elemento “Passo” . . . 60
Tabela 12 – Definição da sintaxe de uma “Condição” do elemento “Passo” 60
Tabela 13 – Definição da sintaxe de uma “Repetição” do elemento “Passo” 61
Tabela 14 – Caso de uso Mostra Documentos construído 66
Tabela 15 – Caso de uso Play Brickles convertido para a CARNAUbA 69
Tabela 16 – Caso de Uso Mostra Texto 81
Tabela 17 – Cenário de Teste 1 para o caso de uso Mostra Textos 85
Tabela 18 – Cenário de Teste 2 para o caso de uso Mostra Textos 86
Tabela 19 – Caso de uso “Play Bowling” convertido para a CARNAUbA 93

LISTA DE ABREVIATURAS E SIGLAS

CAPLUC	*Context Aware software Product Line Use Case template*
ChAPTER	*Context Aware software Product line TEsting geneRation method*
CUT	Caso de Uso Textual
GREat	Grupo de Redes de Computadores, Engenharia de Software e Sistemas
LNC	Linguagem Natural Controlada
LPS	Linha de Produtos de Software
LPSSC	Linha de Produtos de Software Sensível ao Contexto

SUMÁRIO

1 INTRODUÇÃO 16
1.1 Contextualização e Caracterização do Problema 16
1.2 Motivação 18
1.3 Objetivos e Contribuições 19
1.4 Organização da Dissertação 19
2 LINHA DE PRODUTOS DE SOFTWARE SENSÍVEIS AO CONTEXTO 21
2.1 Aplicações Sensíveis ao Contexto 21
2.1.1 Uma arquitetura para Aplicações Sensíveis ao Contexto 22
2.1.2 Aplicações Móveis e Sensíveis ao Contexto 24
2.2 Linhas de Produtos de Software 25
2.2.1 Ciclo de Desenvolvimento de uma LPS 26
2.2.2 Modelo de Características 27
2.2.3 Linha de Produtos de Software Sensível ao Contexto 29
2.3 Conclusão 30
3 GERAÇÃO DE CASOS DE TESTE A PARTIR DE CASOS DE USO . . 31
3.1 Teste de Software 31
3.1.1 Cenário de Teste 32
3.1.2 Caso de Teste 33
3.1.3 Procedimento de Teste 33
3.1.4 Teste em Aplicações Sensíveis ao Contexto 34
3.1.5 Teste em LPS 35
3.1.6 Teste baseado em Requisitos 37
3.2 Caso de Uso 37
3.3 Especificação de Caso de Uso utilizando Linguagem Natural Controlada 39
3.4 Métodos para Geração de Testes 42
3.4.1 Trabalho de Nebut et al. (2006) 42
3.4.2 Trabalho de Gois (2010) 43
3.4.3 Trabalho de Chen e Li (2010) 44
3.4.4 Trabalho de Siqueira (2010) 45
3.4.5 Trabalho de Bertolino e Gnesi (2003) 45

3.4.6 ***Trabalho de Neto (2011)*** . . . 46
3.4.7 ***Trabalho de Santos (2013)*** . . . 46
3.4.7.1 *CAPLUC* . . . *47*
3.4.7.2 *ChAPTER* . . . *47*
3.4.8 ***Comparação entre os Métodos para Geração de Teste a partir de Caso de Uso*** . . . 49
3.5 **Conclusão** . . . 50
4 **CARNAUBA** . . . 51
4.1 **Visão Geral** . . . 51
4.2 **Análise Sintática** . . . 52
4.2.1 ***Nome do Caso de Uso*** . . . 52
4.2.2 ***Caso de Uso estendido*** . . . 53
4.2.3 ***Ponto de Extensão*** . . . 53
4.2.4 ***Categoria de Reuso*** . . . 53
4.2.5 ***Restrição de Contexto*** . . . 54
4.2.6 ***Resumo*** . . . 55
4.2.7 ***Atores*** . . . 55
4.2.8 ***Pré-condição*** . . . 56
4.2.9 ***Pós-condição*** . . . 56
4.2.10 ***Passo*** . . . 56
4.2.10.1 *Ponto de Variação e Variante* . . . *56*
4.2.10.2 *Número do Passo* . . . *57*
4.2.10.3 *Ações do Ator e do Sistema* . . . *58*
4.2.10.3.1 *Ação* . . . 58
4.2.10.3.2 *Ponto de Extensão* . . . 59
4.2.10.3.3 *Condição* . . . 60
4.2.10.3.4 *Repetição* . . . 61
4.3 **Análise Semântica** . . . 61
4.4 **Exemplo de Uso** . . . 63
4.4.1 ***Caso de Uso: Mostra Documentos*** . . . 64
4.4.2 ***Caso de Uso: Play Brickles*** . . . 66
4.5 **Limitações** . . . 70

4.6 **Conclusão** 70
5 **O MÉTODO E A FERRAMENTA PARA ESTENDER O CHAPTER** . 71
5.1 **Limitações do ChAPTER** 71
5.2 **Funcionamento da extensão proposta** 71
5.3 **Ferramenta de Apoio ao Método** 75
5.4 **Arquitetura da Ferramenta** 75
5.4.1 ***Diagrama de Pacotes*** 75
5.5 **Limitações** 77
5.6 **Conclusão** 78
6 **APLICAÇÃO DA EXTENSÃO PROPOSTA** 79
6.1 **Mobiline e Arcade Game Maker** 79
6.2 **Aplicação da Extensão na LPSSC Mobiline** 80
6.3 **Aplicação da Extensão na LPS Arcade Game Maker** 89
6.4 **Conclusão** 94
7 **CONCLUSÃO** 96
7.1 **Resultados Alcançados** 96
7.2 **Trabalhos Futuros** 98
REFERÊNCIAS 99
APÊNDICE A – Lista de Verbos Aceitos na CARNAUbA 105

1 INTRODUÇÃO

Esta dissertação apresenta uma extensão do método *Context Aware software Product line TEsting geneRation method* (ChAPTER) (*Context Aware software Product line TEsting geneRation method*) em direção a geração automática da estrutura para execução de testes em Linhas de Produtos de Software Sensíveis ao Contexto. Para isso, é definida uma linguagem natural controlada que permite essa geração.

Este capítulo está estruturado da seguinte maneira: a Seção 1.1 trata da contextualização e da caracterização do problema; a Seção 1.2 apresenta a motivação para este trabalho; a Seção 1.3 aborda os objetivos e as principais contribuições deste trabalho; e a Seção 1.4 apresenta a organização desta dissertação.

1.1 Contextualização e Caracterização do Problema

Uma Linha de Produtos de Software (Linha de Produtos de Software (LPS)) pode ser definida como "um conjunto de sistemas de software que compartilham um conjunto de recursos comuns que satisfazem as necessidades específicas de um segmento particular do mercado e que é desenvolvido a partir de artefatos comuns de forma sistemática" (NORTHROP, 2002). O conceito de LPS é utilizado para explorar similaridades entre os produtos desenvolvidos (NORTHROP, 2002), e assim, uma LPS apresenta várias possíveis configurações de um dado domínio (ENSAN *et al.*, 2011).

LPS é um paradigma de desenvolvimento de software bastante utilizado quando se deseja obter redução nos custos e no tempo necessário para desenvolver aplicações que pertençam a um mesmo domínio (NORTHROP, 2002). Essa redução no tempo e nos custos acontece devido à reutilização de artefatos. Por isso, esse paradigma é uma das principais abordagens do desenvolvimento baseado em reuso (SELBY, 2005).

Esse tipo de abordagem também pode ser utilizada para desenvolver aplicações sensíveis ao contexto. Uma aplicação sensível ao contexto utiliza informações capturadas do ambiente em que se encontram para prover serviços aos usuários ou mudar o seu comportamento. Quando o objetivo de uma LPS é desenvolver esse tipo de aplicação, que se adapta com base em situações contextuais (DU; WANG, 2008) ou provê um serviço com maior qualidade ou de forma mais adequada, ela é dita Linha de Produto de Software Sensível ao Contexto (FERNANDES *et al.*, 2011).

Tanto em uma Linha de Produtos de Software Sensível ao Contexto (LPSSC) quanto em uma aplicação tradicional é preciso garantir que as aplicações tenham mais qualidade, evitando que a aplicação chegue até o usuário final com problemas. E uma das formas de se assegurar a qualidade de um software é através do teste de software (LIU *et al.*, 2005) que se torna mais complexo à medida que a complexidade e a criticidade das aplicações aumenta (LOKE, 2006). Sendo assim, pode-se afirmar que os testes executados em linhas de produtos e aplicações sensíveis ao contexto são mais complexos do que os testes executados em aplicações tradicionais. No caso de uma LPS, por exemplo, uma das dificuldades de se testá-la vem da definição, por exemplo, do momento correto dos testes serem gerados. Isso porque existem dois momentos para se executar tal atividade (POHL *et al.*, 2010). Em um desses momentos, os artefatos reutilizáveis de uma LPS podem ser verificados e, no outro, os produtos desenvolvidos a partir de uma LPS podem ser testados (POHL *et al.*, 2010). Em se tratando de aplicações sensíveis ao contexto, por sua vez, devem ser consideradas questões como a volatilidade do contexto (LU, 2009), que trata da rapidez da mudança do contexto e como a aplicação se adapta a essas mudanças.

No caso dos testes realizados em uma LPSSC, que une os conceitos de LPS e de aplicações sensíveis ao contexto, a complexidade é maior neles, visto que as dificuldades das duas áreas que a compõem são agregadas.

Para reduzir a complexidade da criação dos testes, eles podem ser automatizados a partir de informações contidas nos artefatos da especificação. Essa técnica é conhecida como testes baseados em especificação e permite uma abordagem efetiva para testar a corretude de software (MILUZZO *et al.*, 2008). Como exemplo, os testes podem ser gerados a partir de casos de uso, os quais descrevem o comportamento do sistema sem revelar a estrutura do comportamento interno (SOMÉ, 2006).

O uso dessa abordagem é útil, pois os testes podem ser gerados antes do código estar completo e até mesmo antes deles serem iniciados, por exemplo, com o uso da técnica Desenvolvimento Dirigido por Testes (TDD – Test Driven Development) (BECK, 2003) ou da técnica Desenvolvimento Dirigido por Testes de Aceitação (ATDD – Acceptance Test Driven Development) (KOSKELA, 2008).

Apesar dos benefícios do teste baseado em requisitos, existem alguns problemas quando se trata da geração de teste a partir de artefatos como casos de uso. Esses artefatos geralmente são descritos utilizando linguagem natural, que pode ser ambígua e imprecisa

(SCHNELTE, 2009), o que dificulta a geração automática de testes. Uma solução para esse problema é o uso de uma Linguagem Natural Controlada (LNC), a qual permite restringir definições de gramática e vocabulário de uma linguagem natural (SCHNELTE, 2009).

1.2 Motivação

Na literatura são encontrados trabalhos que tratam da geração de testes para aplicações tradicionais e para LPS (BERTOLINO; GNESI, ; CHEN; LI, 2010; NEBUT *et al.*, 2006; GOIS *et al.*, 2010; NETO, 2011; SIQUEIRA, 2010), para LPSSC (SANTOS, 2013) e trabalhos que definem LNC (SCHNELTE, 2009; BARROS *et al.*, 2011) para a geração de testes.

Entretanto, apenas o trabalho de (SANTOS, 2013) tem como objetivo a geração de testes para LPSSC. Para realizar a automação da geração dos testes, o autor define um template de caso de uso, denominado *Context Aware software Product Line Use Case template* (CAPLUC) (*Context Aware software Product Line Use Case template*), que permite a inserção de informações associadas a LPS e a sensibilidade ao contexto, como é detalhado no Capítulo 2. O autor também define um método, chamado de ChAPTER (*Context Aware software Product line TEsting geneRation method*), para gerar automaticamente cenários de teste, os quais são uma representação abstrata do teste (SOMÉ; CHENG, 2008), para uma LPSSC, a partir de casos de uso baseados no CAPLUC. Assim, em (SANTOS, 2013), há a lacuna em relação a estudos para automatizar a criação de testes que estejam mais próximos da execução, para reduzir os custos e o tempo envolvidos na geração dos testes.

Conforme mencionado anteriormente, uma forma de permitir a automação da geração dos testes pode ser por meio de uma LNC para descrever casos de uso. O trabalho de (SCHNELTE, 2009) define uma LNC para gerar testes para LPS de uma indústria automotiva, entretanto, a LNC não foi definida para descrever casos de uso. (BARROS *et al.*, 2011), por sua vez, definem uma LNC para descrever casos de uso para aplicações convencionais com o objetivo de gerar testes a partir dos casos de uso. Porém, os autores ainda não contemplam as restrições em uma linguagem natural controlada para descrever a variabilidade de uma LPS ou informações contextuais.

Sendo assim, existe uma ausência de métodos na literatura que gerem automaticamente testes para uma LPSSC. Isso pode contribuir para uma maior qualidade dos testes gerados, bem como pode contribuir na redução do tempo necessário para a geração dos testes para uma LPSSC.

1.3 Objetivos e Contribuições

Este trabalho possui como objetivo propor uma evolução do trabalho desenvolvido por (SANTOS, 2013) em direção a geração automática de testes para Linhas de Produtos de Software Sensíveis ao Contexto. Essa evolução consiste da geração de uma estrutura para a execução dos testes a partir de casos de uso descritos textualmente usando uma Linguagem Natural Controlada.

Para alcançar esse objetivo, é necessário atingir as seguintes metas:

- Definir uma Linguagem Natural Controlada que permita descrever casos de uso que possuam informações de uma LPS e também informações contextuais;
- Propor uma evolução para o método ChAPTER de maneira que seja possível gerar automaticamente testes para LPSSC a partir de casos de uso descritos com a LNC definida neste trabalho;
- Implementar a evolução do método ChAPTER na ferramenta desenvolvida por Santos (2013); e
- Avaliação do método proposto com uma prova de conceito.

Como principais contribuições desta dissertação, espera-se: a) uma Linguagem Natural Controlada para descrever o CAPLUC; b) uma evolução do método ChAPTER para permitir a geração de testes a partir de casos de uso; e c) uma ferramenta de apoio ao uso da LNC e do método.

1.4 Organização da Dissertação

Este capítulo fez uma breve introdução sobre os temas abordados neste trabalho, bem como a motivação, os objetivos a serem alcançados e das principais contribuições. O restante da dissertação está dividida nos seguintes capítulos.

Capítulo 2 – Linhas de Produtos de Software Sensíveis ao Contexto: Este capítulo trata dos conceitos básicos de aplicações sensíveis ao contexto e de Linhas de Produtos de Software, bem como de LPSSC.

Capítulo 3 – Geração de Casos de Teste a partir de Casos de Uso: Este capítulo tem como objetivo descrever os conceitos básicos de teste de software, bem como os trabalhos da literatura que descrevem métodos para gerar testes a partir de casos de uso, além da especificação de casos de uso utilizando uma linguagem natural controlada.

Capítulo 4 – CARNAUbA: Este capítulo trata da descrição da linguagem natural controlada definida nesta pesquisa.

Capítulo 5 – O Método e a Ferramenta para Estender o ChAPTER: Este capítulo aborda o método definido durante esta pesquisa e que estende o método ChAPTER.

Capítulo 6 – Aplicação da Extensão Proposta: Este capítulo apresenta uma aplicação do método proposto que estende o ChAPTER em dois estudos de caso.

Capítulo 7 – Conclusão: Este capítulo apresenta as conclusões desta pesquisa.

Apêndice A – Lista de Verbos Aceitos na CARNAUbA: Este apêndice tem como objetivo descrever os verbos aceitos na linguagem natural controlada definida nesta dissertação.

2 LINHA DE PRODUTOS DE SOFTWARE SENSÍVEIS AO CONTEXTO

Este capítulo aborda os conceitos associados à Linha de Produtos de Software Sensíveis ao Contexto, que são a base para a compreensão desta pesquisa. Ele está dividido da seguinte maneira: na Seção 2.1 são apresentados os conceitos associados à aplicações sensíveis ao contexto e sensibilidade ao contexto; e a Seção 2.2 aborda o conceito de linha de produtos de software, bem como dos conceitos necessários para a compreensão de LPSSC.

2.1 Aplicações Sensíveis ao Contexto

Conforme apresentado na Introdução, uma aplicação sensível ao contexto é aquela que se adapta com base nas informações contextuais atuais do usuário (DU; WANG, 2008) ou que fornece um serviço com maior qualidade ou de maneira mais adequada. Porém, para uma melhor compreensão desse conceito é necessário compreender o que é contexto. (DEY, 2001) definiu contexto como sendo qualquer informação que pode ser usada para caracterizar a situação de qualquer entidade, que pode ser uma pessoa, lugar ou objeto relevante para a interação entre o usuário e a aplicação, incluindo o próprio usuário e a aplicação.

Segundo (WANG *et al.*, 2007), uma aplicação sensível ao contexto é uma aplicação que adapta seu comportamento baseado em dados situacionais para prover serviços ricos e gerenciar recursos escassos. Esses dados situacionais, ou contexto, são informações relevantes para aplicação e podem estar associados a localização do usuário, nível da bateria, hora do dia, dados ambientais (temperatura, umidade) ou preferências do usuário, entre outros. Exemplos de uso de algumas dessas características são descritos a seguir:

- **Localização do Usuário:** a aplicação pode apresentar informações como dados climáticos ou pessoas presentes no ambiente em que o usuário se encontra;
- **Nível da bateria:** o sistema deixa de apresentar determinadas funcionalidades, como a exibição de um vídeo, quando a bateria de um dispositivo móvel estiver em um nível crítico; e
- **Preferências do usuário:** a aplicação pode mudar o idioma de exibição de acordo com o idioma definido no sistema.

Aplicações sensíveis ao contexto podem ser categorizadas de acordo com o momento em que reagem a mudanças no contexto. Este tipo de aplicação pode ser considerada proativa ou reativa. Uma aplicação sensível ao contexto é dita reativa quando ela percebe que alguma

mudança no contexto ocorreu e é dita proativa quando tenta prever os momentos em que alterações no contexto podem ocorrer (SALEHIE; TAHVILDARI, 2009).

Um exemplo de aplicação sensível ao contexto é o GREat Tour (LIMA *et al.*, 2013) desenvolvida pelo Grupo de Redes de Computadores, Engenharia de Software e Sistemas – Grupo de Redes de Computadores, Engenharia de Software e Sistemas (GREat)[1]. O objetivo dessa aplicação é guiar os visitantes dentro do laboratório, exibindo, quando solicitado pelos mesmos, informações associadas a cada ambiente. Essas informações podem ser textos, imagens, vídeos e pessoas que trabalham nesses ambientes. Ela foi implementada a partir de uma Linha de Produtos de Software Sensíveis ao Contexto, a Mobiline (MOBILINE, 2013), que será descrita na seção 2.2.3.

2.1.1 Uma arquitetura para Aplicações Sensíveis ao Contexto

A complexidade das informações de contexto torna o processo de desenvolvimento de software sensível ao contexto mais complexo devido a necessidade de uso de técnicas específicas para desenvolvê-las. Muitas vezes é necessário utilizar middlewares, por exemplo (WANG *et al.*, 2007). Assim, para desenvolver uma aplicação sensível ao contexto, é importante que a mesma possua componentes que permitam desde a captura de informações contextuais até a adaptação do comportamento da aplicação. Dessa forma, é importante que durante o desenvolvimento de aplicativos sensíveis ao contexto sejam seguidas recomendações para a arquitetura.

(MARINHO *et al.*, 2010), por exemplo, propuseram uma arquitetura de referência com o intuito de tentar facilitar o desenvolvimento de aplicações sensíveis ao contexto. Essa arquitetura, que é apresentada na Figura 1, é composta por 4 camadas: (i) camada de aplicação, (ii) camada de serviço, (iii) camada de gestão de contexto e (iv) camada de sensores de contexto.

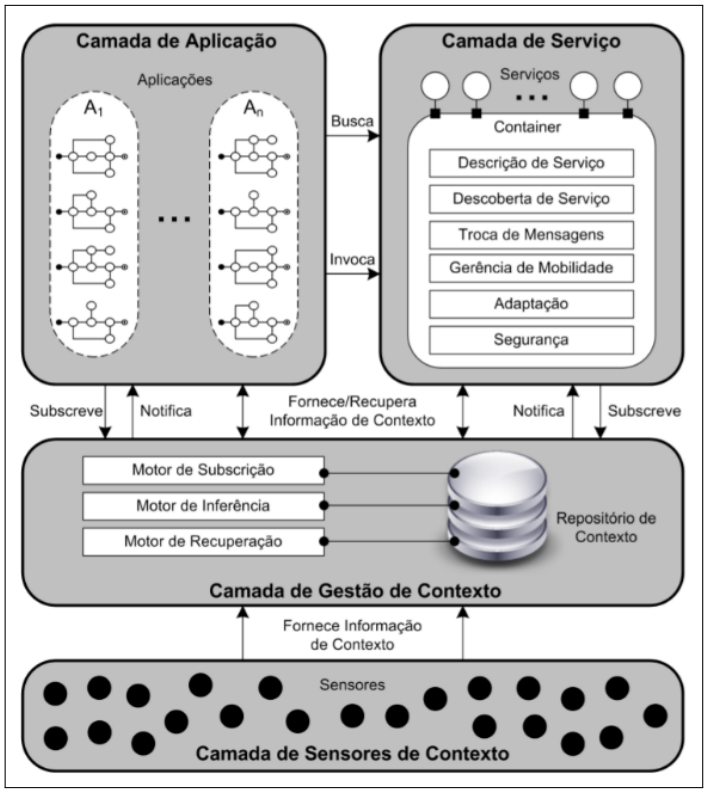
A camada de sensores de contexto é a camada mais inferior dessa arquitetura e tem como papel a captura das informações contextuais do meio em que a aplicação está executando. Tais informações devem ser enviadas para as camadas superiores para serem tratadas e as adaptações de conteúdo ou de serviços sejam realizadas de maneira mais adequada.

A camada de gestão de contexto interage com as outras três camadas e o seu objetivo é armazenar as informações de contexto recebidas das outras três camadas no Repositório de Contexto. Dentro dessa camada existem motores responsáveis para prover informações de contexto para as outras camadas. O primeiro desses motores é o Motor de Recuperação cuja

[1] http://www.great.ufc.br

atividade é permitir que as aplicações realizem consultas no Repositório de Contexto a fim de obter as informações de contexto desejadas. O Motor de Inferência, por sua vez, deriva novas informações com base nas informações já existentes e o Motor de Subscrição comunica, de forma assíncrona, às camadas de Aplicação e de Serviço, sobre as informações contextuais no momento em que as mesmas forem necessárias.

Figura 1 – Arquitetura de referência para uma aplicação sensível ao contexto



Fonte: Marinho *et al.* (2010).

A camada de serviço contém os serviços providos pelo software. (MARINHO *et al.*, 2010) definiram que um serviço é "uma unidade modular de software passível de composição e que provê uma funcionalidade específica podendo ser implantado de maneira independente". Assim, um serviço possui um Container, cuja responsabilidade é gerenciar o ciclo de vida do serviço fornecido e que possibilita a comunicação com outros serviços e aplicações. Cada Container é composto por 6 módulos:

- **Descrição de Serviços:** módulo responsável pela descrição do serviço, contendo dados que o representem, podendo ser essa descrição feita por descrição sintática (palavras-chave) ou por descrição semântica (lógicas descritivas);
- **Descoberta de Serviços:** módulo responsável pela descoberta do serviço mais adequado

para uma requisição da camada de aplicação;

- **Adaptação:** módulo responsável pela coordenação da adaptação dos serviços com base nas informações de contexto coletadas;
- **Segurança:** módulo responsável pela mudança do ponto de conexão que o usuário está conectado e pela garantia da manutenção da sessão do usuário;
- **Gerência de Mobilidade:** módulo responsável pela gerência do código e do estado de execução de um serviço, além de gerenciar os dados acessados pelo serviço; e
- **Troca de Mensagens:** módulo responsável pela gerência da interoperabilidade entre as diferentes tecnologias de redes de computadores.

A última camada é formada pelas próprias aplicações que, por sua vez, são compostas de serviços providos aos usuários das mesmas. E, conforme apresentado anteriormente, tais serviços são selecionados e adaptados baseados nas informações contextuais coletadas e tratadas pelas demais camadas.

2.1.2 Aplicações Móveis e Sensíveis ao Contexto

Como apresentado anteriormente, uma aplicação sensível ao contexto reage de acordo com informações relevantes para o comportamento da aplicação. Este tipo de aplicação pode ser mais facilmente encontrado em dispositivos móveis devido aos avanços desses, além da popularização de tais dispositivos. Tais avanços estão associados à conectividade da rede, processamento e armazenamento de recursos (MAIA *et al.*, 2009), sensores presentes nos dispositivos, entre outros fatores. Sendo uma aplicação móvel uma aplicação que executa em um dispositivo móvel (DANTAS, 2009), uma aplicação móvel e sensível ao contexto une os conceitos dos termos que o compõem. Assim, uma aplicação móvel e sensível ao contexto é uma aplicação que executa sobre um dispositivo móvel e que se adapta ou provê um serviço com base em informações contextuais.

Apesar dos avanços nos dispositivos móveis permitindo que sejam desenvolvidas aplicações mais robustas, ainda existem limitações no hardware (e.g., memória e bateria) que agregam desafios ao desenvolvimento de aplicações móveis e sensíveis ao contexto. (BUTHPITIYA *et al.*, 2012) enumeraram alguns desses desafios presentes no desenvolvimento de aplicações sensíveis ao contexto em um ambiente móvel:

- **Limitação de Recursos:** os dispositivos móveis possuem limitações quanto a processamento, capacidade de armazenamento, memória, largura de banda, entre outros;

- **Número de fontes de informações de contexto:** as fontes podem ser de sensores de software ou de hardware. Devido a isso, os dispositivos têm que tratar com diversas fontes de informação de contexto, o que acarreta em um maior processamento e, como mostrado, o mesmo ainda é limitado;
- **Informação de contexto "em mãos erradas":** a aplicação tem que tratar a segurança da informação de maneira que não permita que ela "caia em mãos erradas". Isso se deve ao fato de não se poder garantir a segurança física de sensores ubíquos ou de plataformas de computação móvel; e
- **Aplicações estão se tornando bastante complexas:** as aplicações estão mais complexas, principalmente em ambientes móveis distribuídos. Outro aspecto que aumenta a complexidade é a lógica do comportamento proativo da aplicação que podem ser inteligíveis para o usuário.

Em suma, ao desenvolver aplicações sensíveis ao contexto é de extrema importância tratar os aspectos da limitação de recursos dos dispositivos, bem como da quantidade de fontes de informação de contexto que podem ser utilizadas para prover um serviço com maior qualidade ao usuário. O aumento do número de fontes acarreta em um maior processamento e também numa maior complexidade das aplicações, além de necessidade, por exemplo, de uma memória com maior capacidade para armazenar, mesmo que temporariamente, tais informações, exigindo, assim, um dispositivo com maiores recursos. Além disso, mas não menos importante, tem-se o aspecto da segurança das informações utilizadas pela aplicação de tal modo que não permitam que aplicativos ou usuários sem autorização tenham acesso aos dados dos usuários com fins prejudiciais para os mesmos.

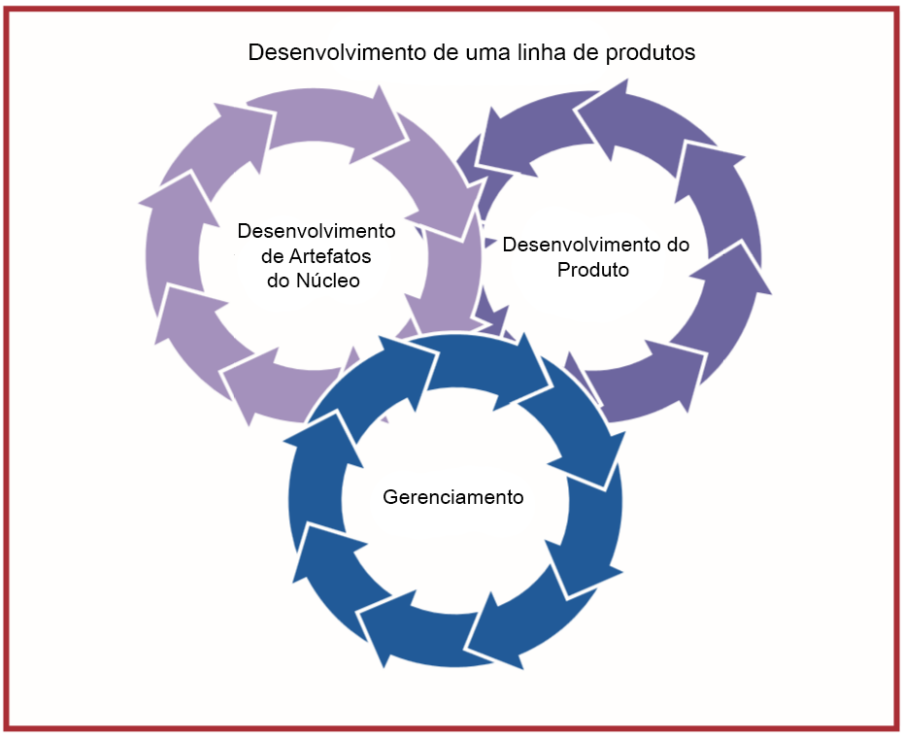
2.2 Linhas de Produtos de Software

Uma Linha de Produtos de Software (LPS) representa um conjunto de produtos de software que compartilham características comuns segundo (ENSAN *et al.*, 2011). (POHL *et al.*, 2010), por sua vez, definiram como sendo um paradigma de desenvolvimento de software que facilita o reuso de artefatos reduzindo, assim, os custos envolvidos no desenvolvimento de aplicações (POHL *et al.*, 2010) que pertençam a um mesmo domínio. Por isso, esse paradigma é uma das principais abordagens do desenvolvimento baseado em reuso (SELBY, 2005).

2.2.1 Ciclo de Desenvolvimento de uma LPS

A Figura 2.2 apresenta o ciclo de desenvolvimento de uma LPS definido por (NORTHROP, 2002). Esse ciclo possui dois processos principais, Desenvolvimento dos Artefatos do Núcleo e Desenvolvimento do Produto, e outro processo secundário que representa o Gerenciamento das atividades desenvolvidas nos processos principais. Essa figura mostra que o ciclo de desenvolvimento de uma LPS é contínuo.

Figura 2 – Ciclo de desenvolvimento de uma LPS



Fonte: Northrop (2002).

O primeiro processo, denominado de Desenvolvimento dos Artefatos do Núcleo, é responsável por definir o domínio da linha e suas principais características. Tais características devem estar presentes em todos os produtos desenvolvidos a partir da mesma. Segundo (POHL *et al.*, 2010), nessa etapa são definidas as similaridades e variabilidades, que indicam o que todos os produtos devem ter em comum e os que diferenciam para se tornarem novos produtos, além do escopo. Nesse momento, também ocorre a construção de artefatos reutilizáveis.

No Desenvolvimento do Produto, são definidos os requisitos específicos de cada produto a ser produzido a partir da linha. É importante ressaltar que os requisitos não podem estar em desacordo com os requisitos definidos para a linha. Portanto, as variabilidades devem estar previstas desde o processo anterior. É nessa etapa que os artefatos produzidos no Desenvol-

vimento dos Artefatos do Núcleo são reutilizados e também onde é explorada a variabilidade da linha (POHL *et al.*, 2010). Segundo (POHL *et al.*, 2010) os principais objetivos desse processo são:

- Alcançar um alto grau de reuso ao reutilizar os artefatos do domínio quando definir e desenvolver uma aplicação da LPS;
- Explorar as similaridades e variabilidades da LPS durante o desenvolvimento de uma aplicação;
- Documentar os artefatos da aplicação e relacionar com os artefatos do domínio; e
- Vincular as variabilidades de acordo com a necessidade da aplicação.

O último processo, denominado Gerenciamento, tem entre suas responsabilidades a supervisão dos outros dois processos, garantindo que nos mesmos, os artefatos necessários sejam produzidos e que os processos definidos para a linha possam ser seguidos corretamente. É atribuição ainda dessa etapa a gestão de pessoas e alocação de recursos para que os processos possam ser executados com sucesso e dentro dos prazos estimados.

2.2.2 Modelo de Características

Para uma melhor compreensão da representação das similaridades e variabildiades presentes nos produtos desenvolvidos a partir de uma LPS, é importante conhecermos alguns termos bastante utilizados no domínio de LPS. O primeiro desses conceitos é o de *features* (características). Segundo (KANG *et al.*, 1990), *feature* é uma característica do sistema visível para o usuário final e pode ser obrigatória, opcional ou alternativa.

Uma *feature* obrigatória representa uma característica que deve estar presente em todas as aplicações da linha. Este tipo de *feature* pode ser visualizado na Figura 3, que possui um excerto do modelo de características da LPSSC Mobiline (MOBILINE, 2014), no retângulo com linha cheia com a inscrição “Text” que indica que todos os produtos originados a partir dessa linha devem conter textos como uma das fontes de informação para os usuários.

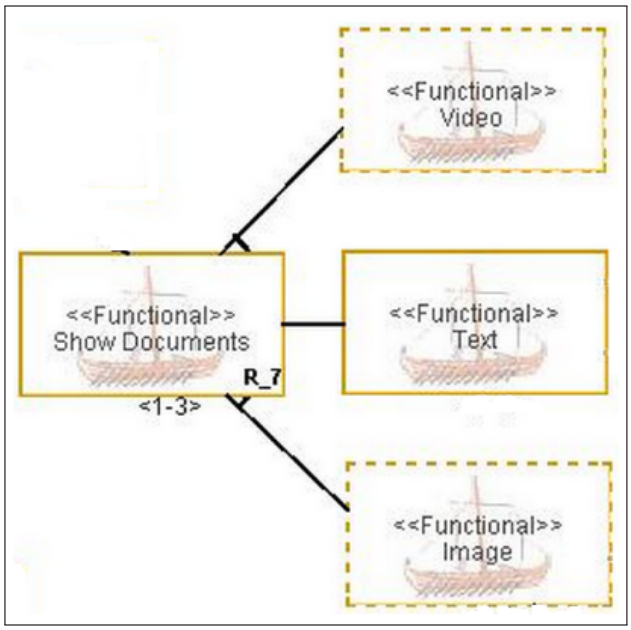
Uma *feature* dita opcional é aquela que pode ou não estar presente em um produto, cabendo a fase de “Desenvolvimento do Produto” definir o que deverá estar presente após o desenvolvimento. Ainda na Figura 3, é possível observar *features* opcionais. Elas são apresentadas em um retângulo com linha tracejada como é o caso das *features* “Video” e “Image”.

Uma *feature* é considerada alternativa quando ela restringe a seleção de uma ou mais *features* entre várias, como as *features* “Via External Service” (via Serviço Externo), “From

Sensor" (Sensor) e "From Memory" (Memória) referentes a propriedade "Capture" mostradas na Figura 4. Ao desenvolver um produto, uma ou mais das *features* alternativas podem ser selecionadas. Elas são representadas por *features* que possuem uma ligação em comum que seria a mesma *feature* pai. No caso da notação presente na Figura 4, é possível escolher entre 1 ou 3 *features* alternativas devido à existência da notação "<1-3>" logo abaixo da *feature* "Capture".

Ainda no modelo de características, podemos definir relacionamentos entre as *features*. Esses relacionamentos podem ser "requer" e "mutuamente exclusivo" (KANG *et al.*, 1990). O relacionamento "requer" dene que uma *feature* necessita que outra esteja ativa para que possa também estar presente representado, assim, uma dependência entre as mesmas. O outro relacionamento diz que duas *features* não podem estar presentes no mesmo produto (KANG *et al.*, 1990). Assim, se uma *feature* estiver ativa a outra *feature*, que possui esse relacionamento com a primeira, deve estar obrigatoriamente inativa.

Figura 3 – Exemplo de features obrigatórias e opcionais da linha Mobiline

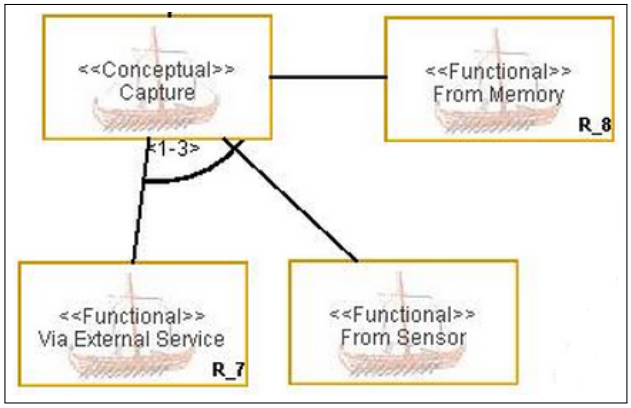


Fonte: MOBILINE (2014).

As notações "R_7" e "R_8", como podem ser vistas nas Figuras 3 e 4, indicam que as características marcadas fazem parte de uma regra de composição inclusiva. Existem também formas de representar regras de composição exclusivas ou regras de contexto. Mais detalhes sobre essas representações podem ser encontrados em (FERNANDES, 2009).

Outras definições importantes quando se trata de modelo de características são as de ponto de variação e variante. Um ponto de variação é denido como um local do programa que

Figura 4 – Exemplos de features alternativas da linha Mobiline



Fonte: MOBILINE (2014).

pode variar entre os diversos produtos de uma linha, que pode ser representada como uma *feature* pai ou uma *feature* alternativa. Um exemplo de ponto de variação é a *feature* "Capture" mostrada na Figura 4. Uma variante é denida como sendo uma das possíveis variações de uma LPS associadas a um ponto de variação, por exemplo, as *features* "Via External Service" (via Serviço Externo), "From Sensor" (Sensor) e "From Memory" (Memória) filhas da *feature* "Capture" apresentada na Figura 4.

2.2.3 Linha de Produtos de Software Sensível ao Contexto

Uma Linha de Produtos de Software Sensíveis ao Contexto (LPSSC) é uma linha cujo objetivo é desenvolver aplicações sensíveis ao contexto (FERNANDES *et al.*, 2011). As questões associadas ao desenvolvimento de uma aplicação sensível ao contexto são agregadas às questões de uma LPS, aumentando, assim, a dificuldade de implementação dos produtos da linha, além de aumentar a dificuldade relacionada a atividade de testes da linha.

Um aspecto relevante do teste em uma LPS ou LPSSC é o fato que eles podem ser executados durante a definição das características da linha, no processo "Desenvolvimento dos Artefatos do Núcleo", ou no desenvolvimento do produto. Quando se trata do teste na linha, ele pode ser executado para verificar algum componente que já esteja desenvolvido e que será reutilizado por outros produtos originários a partir da LPS ou LPSSC.

Outro aspecto importante para este trabalho é referente às mudanças de contexto que podem afetar as características dos sistemas (MARINHO *et al.*, 2011), e, assim, mudando os serviços que são fornecidos para o usuário ou até a corretude da execução destes. Devido a isso, garantir qualidade a essas aplicações requer um esforço maior. Tais problemas são melhor

abordados no Capítulo 3.

Um exemplo de LPSSC é a linha Mobiline, como citado anteriormente, que tem como objetivo desenvolver aplicações de guias de visitas móveis e sensíveis ao contexto. Essas aplicações devem ser desenvolvidas para dispositivos móveis e devem guiar os usuários dentro do ambiente fornecendo ao usuário informações específicas dos ambientes em que ele se encontra, além de informações sobre as pessoas que trabalham no ambiente. É possível também melhorar as informações fornecidas baseada no perfil do visitante e com base na carga da bateria.

Algumas aplicações podem ser desenvolvidas a partir da LPSSC Mobiline, entre elas pode-se citar, por exemplo, guias de visitas para museus, laboratórios ou pontos turísticos. Um produto gerado pela Linha é o GREat Tour que, conforme mencionado na Seção 2.1, tem como intuito guiar os visitantes dentro do laboratório GREat, indicando textos, imagens, vídeos e pessoas alocadas em cada ambiente visitado.

2.3 Conclusão

Neste capítulo foram apresentados os conceitos de aplicações sensíveis ao contexto e Linhas de Produtos de Software Sensível ao Contexto.

No tocante à aplicações sensíveis ao contexto, além dos conceitos, também foi apresentada uma arquitetura de referência encontrada na literatura, a qual descreve os componentes necessários, como um componente para descoberta de serviços existentes no ambiente. Também foram abordados os conceitos de aplicações móveis e sensíveis ao contexto.

Quanto às LPSSCs, depois que os conceitos básicos de LPS foram introduzidos, o ciclo de desenvolvimento de uma LPS foi descrito. O conceito de *features* também foi trabalhado, sendo uma *feature* uma característica que os produtos de uma linha podem possuir. Em seguida, foi apresentado o conceito de LPSSC, que é uma linha de produto de software para desenvolver aplicações sensíveis ao contexto.

3 GERAÇÃO DE CASOS DE TESTE A PARTIR DE CASOS DE USO

Neste capítulo são apresentados primeiro os conceitos associados a teste de software, casos de uso, além de linguagem natural controlada para especificação de casos de uso.

Em seguida, são apresentados os métodos existentes na literatura que possuem como objetivo a geração de casos de teste ou cenários de teste para aplicações convencionais e para linhas de produtos de software, bem como para LPSSC. Trabalhos relacionados que, para alcançar esse objetivo de geração de casos de testes, definem uma LNC para gerar testes também são descritos. Todos esses trabalhos foram encontrados após serem realizadas pesquisas na literatura que definissem métodos para geração de testes a partir de casos de uso ou que definissem uma LNC. Assim, neste capítulo são descritos os trabalhos de (NEBUT *et al.*, 2006), (GOIS *et al.*, 2010), (CHEN; LI, 2010), (SIQUEIRA, 2010), (BERTOLINO; GNESI,), (NETO, 2011) e (SANTOS, 2013) definem métodos para geração de testes, além do trabalho de (BARROS *et al.*, 2011) que define uma LNC para gerar testes.

Para isso, este capítulo se encontra assim dividido: na Seção 3.1 são apresentados os conceitos de teste de software; na Seção 3.2 casos de uso são conceituados; na Seção 3.3 é definido o conceito de Linguagem Natural Controlada, bem como uma LNC pode ser utilizada em um caso de uso; a Seção 3.4 contém os trabalhos que propõem um método para geração de testes ou definem uma LNC; na Seção 3.4.8 é apresentada uma análise comparativa desses trabalhos; e a Seção 3.5 apresenta as conclusões deste capítulo.

3.1 Teste de Software

O teste de software é uma atividade fundamental para garantir a qualidade dos sistemas [Liu et al. 2005] e está focado em revelar erros em um sistema para assegurar confiabilidade ao mesmo (CHEN; LI, 2010). É importante ressaltar que o teste é uma das atividades mais caras e que mais consome tempo durante o desenvolvimento de software (SHAMSODDIN-MOTLAGH, 2012; SANTOS *et al.*, 2011), chegando a 50% do tempo gasto no desenvolvimento (MYERS *et al.*, 2011) e a 50% dos custos da produção de um software (MYERS *et al.*, 2011; LIU *et al.*, 2005; HIERONS *et al.*, 2009).

(MYERS *et al.*, 2011) definiram que o teste de software é um processo ou conjunto de processos para garantir que o código faça aquilo que deve fazer e não faça aquilo para o qual não foi planejado. Logo, o teste deve tentar garantir que o software execute corretamente de

acordo com os requisitos.

A criação dos testes em etapas iniciais do desenvolvimento de software traz alguns benefícios como a descoberta de defeitos ainda na fase de especificação que é importante tanto para o desenvolvimento da aplicação em si como para a atividade de testes servindo de referência para os mesmos (POHL, 2010). Facilitando, assim, a identificação da conformidade do software com os requisitos. Além disso, os custos associados com a definição, implementação, quando possível, e execução dos testes podem ser reduzidos.

É importante ressaltar que os testes podem ser categorizados em duas categorias principais: quanto ao objetivo e quanto ao nível. Neste trabalho focamos na classificação quanto ao nível que podem ser de três tipos: (i) teste de unidade; (ii) teste de integração; e (iii) teste de sistema. O teste de unidade representa o processo de teste para partes menores de um programa como sub-rotinas, subprogramas ou procedimentos em um programa (MYERS *et al.*, 2011). O teste de integração exercita o mesmo código que o teste de unidade, porém verificando a comunicação e as ações entre as unidades (RUBINOV, 2010). O teste de sistema verifica se o sistema atende aos objetivos (MYERS *et al.*, 2011).

Existem ainda outros termos utilizados na parte de teste de software que é em relação à classificação no que concerne à especificação do teste. Assim, podemos ter uma especificação menos concreta sendo em um nível mais alto ou uma especificação mais detalhada ou ainda as etapas a serem executadas durante o teste. Esse último tipo de especificação é melhor detalhado nas subseções seguintes.

3.1.1 Cenário de Teste

Um cenário de teste, segundo (SOMÉ; CHENG, 2008), é uma representação abstrata do teste. Logo, o cenário não deve conter informações detalhadas dos testes a serem executados. Por exemplo, ele não contém os valores a serem utilizados e nem os passos a serem executados detalhadamente. A Figura 5 apresenta um exemplo de cenário de teste.

Nesse cenário de teste descrito na Figura 5, é possível perceber questões associadas a identificação do cenário de teste através do elemento ID, além de associar a um caso de uso e a um cenário do caso de uso definido. Essa identificação do cenário de teste permite identificar se é um cenário principal, alternativo, opcional ou outras possibilidades que podem ser definidas pelo analista de testes. As definições das entradas e saídas é descrita no caso de teste que é explicado na Seção 3.1.2 e os passos a serem executados são informados no procedimento de teste que será

apresentado na Seção 3.1.3.

Figura 5 – Exemplo de cenário de teste para o GREat Tour

ID	Cenário de Teste 01
Caso de Uso	Autenticação
Categoria	Cenários
Cenário do Caso de Uso	Principal

Fonte: O autor.

3.1.2 Caso de Teste

Um caso de teste contém as entradas a serem utilizadas no teste, as saídas esperadas e o procedimento de teste a que se refere. A necessidade de se identificar os valores de entrada e saída é relevante em caso de erros, pois permite reproduzir o teste novamente de maneira igual a fim de verificar os erros encontrados e corrigi-los. Após a correção ou em caso de sucesso, garante que em determinadas condições o sistema executa conforme o esperado.

A Figura 6 apresenta um exemplo de caso de teste para a aplicação GREat Tour. Nesse caso de teste é possível identificar o objetivo do mesmo, que é a realização da autenticação através de login na aplicação. Como entradas, tem-se que para o campo “username” é usado o valor “guest” e no campo “password” é utilizado “1234”. Como saída esperada, tem-se definida a tela inicial da aplicação com o mapa da Recepção do laboratório. Esse caso de teste está associado ao procedimento de teste “PT001”.

Figura 6 – Exemplo de caso de teste para o GREat Tour

ID	CT001	
Descrição	Fazer login no GREat Tour	
Itens a testar	Verificar se a tentativa de fazer login com um usuário e senha corretos é realizada com sucesso.	
Entradas	**Campo**	**Valor**
	Username	guest
	Password	1234
Saída	Tela inicial da aplicação com o mapa da Recepção do laboratório.	
Procedimento	PT001	

Fonte: O autor.

3.1.3 Procedimento de Teste

Um procedimento contém os passos a serem executados no software sob teste. Eles podem estar associados a mais de um caso de teste, pois os passos podem ser utilizados para se

testar várias situações. Um exemplo de procedimento de teste para o GREat Tour é apresentado na Figura 7.

Na Figura 7, é possível ver que os passos para se realizar o login GREat Tour são definidos. O primeiro passo é o preenchimento do campo “username”. Em seguida, ocorre o preenchimento do campo “password”, e, por fim, há o clique em “Login”. Assim, esse procedimento pode ser utilizado em várias situações de teste, que podem ser entradas consideradas válidas (e.g., senha tem que ter entre 6 e 13 caracteres contendo caracteres especiais) ou valores que representam entradas inválidas.

Figura 7 – Exemplo de procedimento de teste para o GREat Tour

ID	PT001
Descrição	Login no GREat Tour
Procedimento	1. Preencher o campo “username” 2. Preencher o campo “password” 3. Clicar em “Login”

Fonte: O autor.

3.1.4 Teste em Aplicações Sensíveis ao Contexto

Segundo (LOKE, 2006), o teste se torna mais difícil e têm o custo aumentado devido ao aumento da complexidade e da criticidade das aplicações ou quando elas se tornam pervasivas. Ainda segundo (LOKE, 2006), essas são aplicações sensíveis ao ambiente que processam a informação recebida do meio e agem de acordo com o resultado do processamento. Assim, esse tipo de aplicação é o que foi chamado neste trabalho de aplicações sensíveis ao contexto.

É importante lembrar que mudanças no contexto podem ocorrer e afetar o comportamento da aplicação em qualquer momento durante a execução (WANG *et al.*, 2007). Isso ocorre devido a alta dinamicidade do contexto (e.g., força do sinal), dados aproximados (e.g., localização) ou ainda dados contraditórios (e.g., sensores percebem eventos diferentes em um mesmo momento) (WANG *et al.*, 2007). (LU, 2007) identificou alguns desafios associados ao teste em aplicações sensíveis ao contexto, como a incerteza da computação que diz que as aplicações interagem com o ambiente em mudança e isso é difícil de reproduzir em um teste.

Com base nisso, existe uma dificuldade para garantir o comportamento desejado e dos níveis de qualidade desejados (BERTOLINO, 2007). Para tentar reduzir essa dificuldade (WANG *et al.*, 2007) enumeraram alguns passos a serem executados para se obter uma melhor qualidade nos testes em aplicações sensíveis ao contexto:

1. Identificar os pontos chaves onde uma informação de contexto pode afetar o comportamento da aplicação;
2. Gerar potenciais pontos de variação para cada caso de teste que devem explorar a execução de diferentes sequências de contexto; e
3. Direcionar a aplicação para uma sequência de contexto gerada.

Quando se trata de aplicações sensíveis ao contexto, executando em dispositivos móveis, existem outros desafios a serem solucionados. (MYERS *et al.*, 2011) identificaram que existem mais desafios para o teste em aplicações móveis do que em qualquer outro tipo de aplicação ou plataforma exigindo, assim, um esforço adicional em relação ao processo de teste tradicional (DANTAS, 2009).

(MYERS *et al.*, 2011) também identificaram que mais do que as próprias aplicações, os desafios são inseridos pelo ambiente e pelos dispositivos. Assim, é preciso considerar alguns desafios ao se testar aplicações móveis. São exemplos de alguns desses desafios:

- **Variedade de dispositivos:** devido à grande variedade de dispositivos, é difícil garantir que uma mesma aplicação execute corretamente em todos eles;
- **Restrições de hardware:** apesar dos avanços no hardware dos dispositivos móveis, ainda existem limitações na memória ou processador como também há o tamanho reduzido da tela; e
- **Variedade de dispositivos de entrada:** é preciso analisar a variedade de dispositivos de entrada, pois as informações podem ser fornecidas por teclados, botões, telas touch-screen, entre outros.

Além desses tópicos identificados por (MYERS *et al.*, 2011), é possível identificar também a grande variedade de meios que o dispositivo pode interagir com o usuário para indicar uma resposta a uma solicitação feita por ele. Por exemplo, podemos ter o retorno háptico (vibração), sonoro, visual, entre outros. Tais modos de saída podem ser influenciados também diante das mudanças de contexto identificadas pela aplicação. Assim, tudo interfere na definição e execução dos testes.

3.1.5 Teste em LPS

Quando se trata do teste em LPS, desafios também são agregados ao processo de teste tradicional. Isso ocorre devido ao fato de existirem dois momentos para a execução dos testes em uma LPS (POHL *et al.*, 2010). Essas etapas são referentes aos testes de domínio, que

devem ser executados ainda na fase de Desenvolvimento de Artefatos do Núcleo, e testes de aplicação, que devem ser executados na fase de Desenvolvimento do Produto.

Quando se trata do teste de domínio, é possível identificar defeitos nos artefatos gerados ainda na primeira etapa e também nos artefatos que podem ser reutilizados no teste da aplicação (POHL *et al.*, 2010). Além disso, componentes reutilizáveis podem ser desenvolvidos e testados ainda nesse primeiro momento, não excluindo a execução posterior de testes na etapa referente ao desenvolvimento de produtos de modo que assegure o correto funcionamento do componente quando associado a outros componentes.

O teste de domínio possui dois objetivos principais. O primeiro deles refere-se à validação dos artefatos gerados na primeira fase onde os objetivos e escopo da linha são definidos. Assim, seria assegurada a concordância dos produtos com a linha permitindo uma rastreabilidade entre as etapas do processo de desenvolvimento. O segundo objetivo diz respeito à definição de um processo de teste geral e eficiente (POHL *et al.*, 2010) para que os componentes e os produtos, compostos por aqueles, possam ser testados de maneira mais adequada, evitando, assim, retestes desnecessários, o que encareceria o projeto.

Na etapa de teste de domínio, os componentes ou funções desenvolvidas para serem utilizados por todos os produtos da linha podem ser testadas a partir de artefatos produzidos na especificação de requisitos ou artefatos de arquitetura ou design. Esses componentes também podem ser testados apenas nas partes comuns a todos os produtos deixando o teste para as partes variáveis para o teste de aplicação (POHL *et al.*, 2010).

O teste da aplicação, por sua vez, compreende as atividades que verificam e validam uma aplicação contra sua especificação. Isso ocorre porque a aplicação é validada conforme as definições na primeira fase do desenvolvimento em uma LPS e também conforme os requisitos específicos do produto. Há também a reutilização de artefatos produzidos na etapa anterior para encontrar defeitos nas aplicações geradas a partir de uma LPS. Assim, essa etapa tem como objetivo garantir uma qualidade mínima para a aplicação sob teste (POHL *et al.*, 2010). Para alcançar esses objetivos, algumas verificações devem ser realizadas:

- Verificar variantes que não devem estar presentes;
- Assegurar a presença das variantes que deveriam existir; e
- Garantir que a aplicação foi configurada corretamente sem violar nenhuma restrição ou dependência de variabilidade.

3.1.6 Teste baseado em Requisitos

A partir de requisitos, é possível gerar testes de sistema, de aceitação (POHL, 2010), de unidade, funcionais, entre outros. Assim, é possível garantir uma maior qualidade aos sistemas desenvolvidos, pois serão verificados e validados de maneira que seja assegurada a conformidade dos mesmos em relação às necessidades dos clientes.

A geração de testes a partir de requisitos possui vantagens, sendo a primeira delas o fato de que os artefatos de requisitos são uma excelente base para a derivação dos testes. Isso ocorre devido a esses artefatos possuírem propriedades do sistema que são relevantes para o usuário (POHL, 2010).

Em segundo lugar, pode ser citada como outra vantagem o fato de que erros nos requisitos podem ser identificados durante a criação e execução dos testes. Essa situação pode ser identificada, porque, como apresentado anteriormente, é feita uma verificação do comportamento do sistema com o intuito de assegurar a sua conformidade em relação aos requisitos do usuário (POHL, 2010).

A terceira vantagem refere-se à identificação de falhas nos artefatos dos requisitos. Caso uma falha na especificação de requisitos não seja encontrada durante a verificação e validação dos artefatos, elas podem ser encontradas durante a derivação de testes a partir dos requisitos (POHL, 2010). Assim, enquanto os testes estão sendo criados, erros nos requisitos podem ser encontrados e os mesmos são corrigidos, contribuindo, assim, para a redução dos custos que estão associados com uma possível correção em um momento posterior.

Na geração dos testes podem ser utilizadas duas abordagens. A primeira delas é a derivação direta de casos de teste a partir dos artefatos de requisitos. Assim, os testes podem ser derivados de casos de uso, entre outros artefatos. A segunda maneira é a geração de testes a partir de modelos, abordagem conhecida como derivação de casos de teste baseada em modelos (POHL, 2010), sendo o foco deste trabalho a geração de testes a partir de casos de uso.

3.2 Caso de Uso

Um caso de uso descreve uma parte do comportamento do sistema sem revelar a estrutura do comportamento interno (SOMÉ, 2006), sendo essa descrição feita de maneira genérica. Ele também pode ser utilizado para capturar, modelar e formalizar os requisitos do usuário (KONG; YUAN, 2009; FANTECHI *et al.*, 2004) o que facilita a compreensão dos

objetivos e das funcionalidades dos sistemas, permitindo uma geração de testes com uma maior qualidade.

(ANTHONYSAMY; SOMÉ, 2008) definiram que um caso de uso captura os interesses dos stakeholders (pessoas que podem influenciar no desenvolvimento do software ou da linha a ser desenvolvida) bem como a interação entre o sistema e os atores (quem executa uma ação no sistema). Um ator representa uma pessoa ou sistema que interage com o sistema descrito nos casos de uso (BERTOLINO *et al.*, 2002). Um ator pode ser classificado como primário ou secundário. Um ator primário é aquele que interage com o caso de uso, de forma que o dispara para ser executado. Um ator secundário interage com o caso de uso, mas não é responsável por iniciar a sua execução (BERTOLINO *et al.*, 2002) aparecendo sua interação apenas no decorrer da execução do caso de uso.

Os casos de uso também podem ser representados graficamente. Isso permite que sejam feitas verificações nas interações entre o sistema e os atores identificando quem pode executar uma determinada ação. Além disso, é possível identificar as relações entre os diferentes casos de uso de um sistema, bem como o relacionamento dos casos de uso e dos atores (POHL, 2010).

Um caso de uso pode se relacionar de três formas com outro caso de uso. A primeira maneira é a "generalização" que diz que um caso de uso especializado "A" herda os passos de interação do caso de uso generalizado "B". O segundo relacionamento diz que um caso de uso "A" estende sequências de interações de outro caso de uso "B", modificando alguns passos dessas interações. O último relacionamento é o que inclui em um caso de uso "A" uma sequência de interações documentadas no caso de uso "B" que é dito incluso no caso de uso "A" (POHL, 2010).

Outro ponto importante quando se trata de casos de uso são os tipos de cenários que podem existir, sendo eles o cenário principal, o alternativo e o excepcional. O principal deve descrever uma sequência de interações que normalmente é executada no sistema. Um cenário alternativo documenta uma sequência de interações que podem ser executadas no lugar do cenário principal, mas que levam ao mesmo fim do cenário principal. Um cenário excepcional representa a execução do sistema quando um evento excepcional (e.g. lançamento de exceções) ocorre durante a execução de outro cenário incluindo a execução de um cenário excepcional.

Um caso de uso é amplamente utilizado para modelar requisitos funcionais em aplicações tradicionais, assim como em linha de produtos de software (ANTHONYSAMY; SOMÉ,

2008). Assim, é importante lembrar que nos casos de uso devem ser inseridas informações referentes as variabilidades existentes na linha. Para facilitar a definição dessas informações, ao se modelar casos de uso podem ser utilizados templates.

Com base nisso, é possível encontrar na literatura alguns templates para modelar casos de uso tanto para aplicações tradicionais, como para linhas de produtos de software. (ANTHONYSAMY; SOMÉ, 2008) identificaram alguns trabalhos ([(JACOBSON *et al.*, 1997; GOMMA, 2004; JOHN; MUTHIG, 2002)) que definiram templates para trabalhar com as variabilidades existentes nas linhas de produtos.

Outro trabalho encontrado na literatura, o de (SANTOS, 2013), além de definir um template próprio, identificou outros templates de caso de uso que representavam variabilidade de uma linha. Ele conseguiu identificar nove templates ([(ERIKSSON *et al.*, 2004; BERTOLINO *et al.*, 2002; GALLINA; GUELFI, 2007; CHOI *et al.*, 2008; BONIFáCIO; BORBA, 2009; ARAúJO, 2010; GOMMA, 2004; JOHN; MUTHIG, 2002; ANTHONYSAMY; SOMÉ, 2008)). Como o foco deste trabalho não é propor um novo template de casos de uso será apresentado apenas o template que é utilizado como base para este trabalho. Na Tabela 3.1 é apresentado um caso de uso para a funcionalidade de autenticação de um produto da LPSSC Mobiline seguindo o template definidod por (SANTOS, 2013). É importante ressaltar que o template será melhor explicado na Seção 3.4.7.1.

3.3 Especificação de Caso de Uso utilizando Linguagem Natural Controlada

Uma linguagem natural controlada (LNC) pode ser denida como sendo um subconjunto de uma linguagem natural completa (ou pura) com restrições sobre a gramática, vocabulário e estilo (SCHWITTER; TILBROOK, 2006). Ela ajuda a eliminar a ambiguidade e complexidade de uma linguagem natural pura (LU, 2010). Além disso, uma Linguagem Natural Controlada (LNC) pode resolver o problema de legibilidade, entendimento e tradução (CARDEY *et al.*, 2008) de uma informação que deve ser compreensível, ao mesmo tempo, para humanos e máquinas. É importante ressaltar que uma sentença válida em uma LNC é uma sentença válida em uma linguagem natural (SCHNELTE, 2009). Porém, o inverso não é sempre verdadeiro (SCHNELTE, 2009).

Ao se comparar uma LNC com outras abordagens, ela apresenta muitas vantagens. Uma linguagem natural pura é fácil de usar, porém pode apresentar ambiguidade e imprecisão (SCHNELTE, 2009), e, por isso, é difícil de ser processada por máquinas. Métodos formais

Figura 8 – Caso de Uso "Autenticação" baseado no template de (SANTOS, 2013)

Nome do Caso de Uso		Autenticação	
Caso de Uso Estendido			
Ponto de Extensão			
Categoria de Reuso		Obrigatório	
Restrição de Contexto			
Informação de Contexto			
Ator Principal		Visitante	
Atores Secundários		Servidor	
Resumo		O caso de uso contém os passos para autenticação	
Pré-condição		O Visitante instalou a aplicação no dispositivo móvel.	
Pós-Condição		O Visitante é autenticado e pode acessar as funções do sistema.	
Passos		**Ações do Ator**	**Ações do Sistema**
	1	O Visitante digita o login e a senha.	
	2		O Sistema envia os dados do Visitante ao Servidor.
	3	O Servidor valida os dados do Visitante no banco de dados.	
	4		O Sistema exibe a tela inicial.
Fluxos Alternativos			
Passo 03 (Os dados do Visitante não existem no ba.)		O Sistema não valida os dados do Visitante. O Sistema exibe uma mensagem de erro.	

Fonte: O autor.

também podem ser utilizados, pois são fáceis de serem compreendidos por um computador, mas são difíceis de serem usados por pessoas sem treinamento (SCHNELTE, 2009). Portanto, uma linguagem natural controlada seria a solução intermediária para os problemas apresentados pelas duas outras abordagens, e, por isso, é utilizada neste trabalho. Alguns exemplos são:

- **PENG (Processable English):** LNC definida para solucionar problemas que surgiam ao usar linguagens de Web Semântica;
- **LiSe (Linguistic et Sécurité):** LNC baseada em francês cujo objetivo é facilitar a tradução para LNCs definidas em outros idiomas;
- **Linguagem de (SCHNELTE, 2009):** essa linguagem foi criada para facilitar a especicação de requisitos para uma indústria automotiva; e
- **ucsCNL:** essa LNC, baseada no inglês, foi criada para descrever casos de uso para aplicações convencionais.

(SCHNELTE, 2009) deniu que uma linguagem natural controlada é composta de duas partes: um vocabulário e uma gramática. O vocabulário deve conter todas as palavras que podem ser usadas dentro da linguagem (SCHNELTE, 2009), pois ele deve ser específico do

domínio para evitar que duas palavras possam representar a mesma entidade, além de evitar ambiguidade léxica (mesmo termo representa duas ou mais entidades) (BARROS *et al.*, 2011). A gramática, por sua vez, é uma versão restrita de uma gramática de uma linguagem natural pura (SCHNELTE, 2009) e pode ser geral ("escrever frases curtas e simples") ou mais formal que restringe a estrutura sintática aceita (BARROS *et al.*, 2011). Em (PENG, 2014), é possível encontrar um exemplo de LNC que restringe a sintaxe:

Sentença -> Sujeito + Predicado

Sujeito -> Determinante {+ Modificador pré-nominal} + Núcleo do Nome {+ Modificador pós-nominal}

Sujeito -> Núcleo do Nome

Predicado -> {Negação} + Núcleo Verbal + Complemento {+ Adjunto}

Nessa linguagem PENG (Processable ENGlish), uma sentença é composta, assim como em linguagem natural, de sujeito e predicado. O sujeito deve possuir um determinante (i.e. artigo), além de poder ter alguns modicadores. É obrigatória a presença do núcleo, que representa o próprio sujeito. "O mordomo" ou "A mãe do mordomo" ou ainda "Agatha" são exemplos de sujeitos aceitos pela linguagem. O predicado dessa linguagem pode conter uma negação além de possuir um núcleo verbal e um complemento para esse verbo. Esse complemento ainda pode possuir um adjunto. Por exemplo, "trabalha na mansão" ou "não trabalha na mansão" ou ainda "trabalha com todos os mordomos".

Em (PENG, 2014), também é possível encontrar um exemplo que compara uma sequência de frases em linguagem natural e a representação da mesma frase utilizando a LNC denida por (SCHWITTER; TILBROOK, 2006). Um trecho dessa comparação é a frase descrita em linguagem natural: "Agatha, o mordomo e Charles vivem na Mansão Dreadsbury.". Ao transformá-la para a linguagem de (SCHWITTER; TILBROOK, 2006) a frase é dividida em três outras frases. Por exemplo, "Agatha vive na Mansão Dreadsbury.", "O mordomo vive na Mansão Dreadsbury." e "Charles vive na Mansão Dreadsbury.". Essa separação ocorre devido a restrições sintáticas da LNC como apresentado anteriormente, porém as frases se equivalem semanticamente. Essa comparação é apresentada na Tabela 1.

É necessário avaliar uma LNC para assegurar que a mesma é de fácil compreensão em relação a uma linguagem natural. Para isso, estudos com usuários são o único modo de vericar esse aspecto de uma LNC (KUHN, 2009). Ao se realizar os estudos com usuários, geralmente são usadas ferramentas de apoio a LNC (KUHN, 2009) o que diculta a análise dos resultados, pois é

difícil determinar o quanto a ferramenta inuencia no entendimento da linguagem (KUHN, 2009). Para facilitar a validação de uma LNC, (KUHN, 2010) deniu um framework, que independe de ferramenta, chamado ontographs e representa gracamente expressões de uma LNC. Frases seguindo a estrutura da LNC devem ser criadas de maneira que possam ser comparadas com a representação na ontographs e se for possível declarar se uma frase é falsa ou não, assim a linguagem pode ser validada.

Tabela 1 – Comparação entre uma Linguagem Natural e uma LNC

Linguagem Natural	LNC
Agatha, o mordomo e Charles vivem na Mansão Dreadsbury.	Agatha vive na Mansão Dreadsbury.
	O mordomo vive na Mansão Dreadsbury.
	Charles vive na Mansão Dreadsbury.

Fonte: Adaptado de Peng (2014).

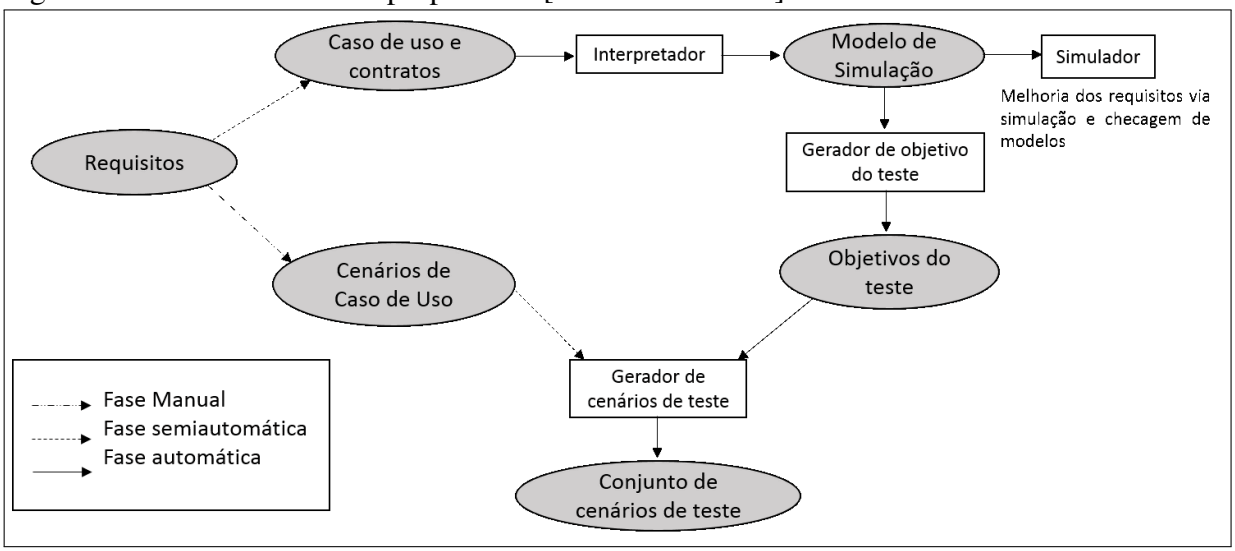
3.4 Métodos para Geração de Testes

3.4.1 Trabalho de Nebut et al. (2006)

(NEBUT *et al.*, 2006) propuseram uma abordagem para automação da geração de cenários de teste a partir de casos de uso no contexto de software embarcado orientado a objeto. O método desenvolvido por (NEBUT *et al.*, 2006) é baseado em um modelo de casos de uso que permite que sejam descobertas ambiguidades nos requisitos descritos em uma linguagem natural. Esse método foi construído com base em caso de uso UML (UML, 2014) aprimorado com contratos, que ajudam a inferir a ordenação parcial correta das funcionalidades que o sistema deve oferecer (NEBUT *et al.*, 2006). A Figura 9 apresenta o funcionamento da proposta de (NEBUT *et al.*, 2006).

Foi também denido pelos autores um simulador de casos de uso. O simulador permite detectar a corretude e consistência dos casos de uso. Essa detecção é possível devido a um modelo de simulação que permite validar as possíveis sequências de casos de uso e extrair caminhos relevantes utilizando critérios de cobertura. Esses caminhos foram chamados de objetivos do teste. A geração dos objetivos de teste a partir dos casos de uso é a primeira etapa da abordagem de (NEBUT *et al.*, 2006). A segunda etapa visa gerar os cenários de teste a partir dos objetivos de teste gerados na etapa anterior.

Figura 9 – Funcionamento da proposta de [Nebut et al. 2006]



Fonte: Adaptado de Nebut *et al.* (2006).

3.4.2 Trabalho de Gois (2010)

(GOIS *et al.*, 2010) propôs a criação de um diagrama para geração de scripts de teste, o Test Script Diagram (TSD). Este diagrama possui uma representação gráca dos uxos de casos de uso e permite associar dados de teste com as respectivas etapas em que são utilizados (GOIS *et al.*, 2010). Além disso, o autor deniu um método para geração de casos de teste a partir do TSD.

O TSD possui 6 elementos, os quais foram descritos utilizando a Forma Normal de Backus Naur (BNF). Esses elementos foram assim definidos:

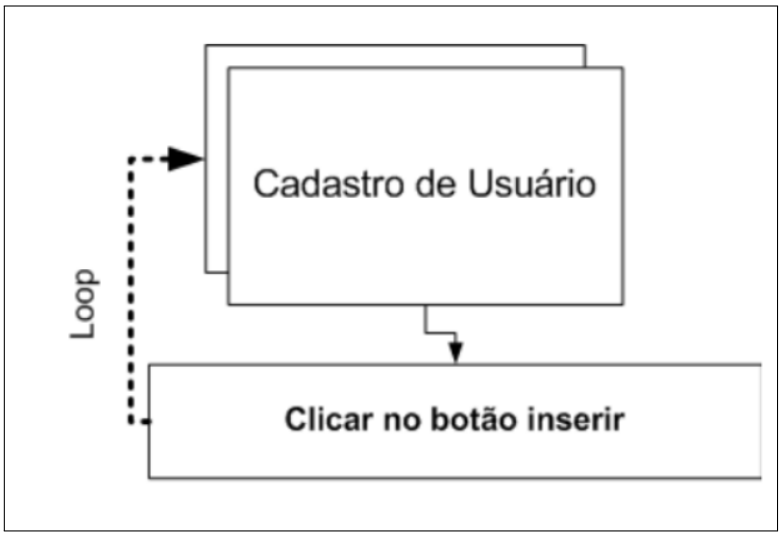
- **Passo:** descreve uma ação ou uma verificação de um caso de teste, possuindo uma ação descritiva que pode conter variáveis utilizadas para associar um passo a um dado de teste. Pode conter as seguintes definições para diferenciar as ações e verificações de casos de teste: «Ação» e «Vericação». Ele é representado por um retângulo e contém a expressão que descreve o passo, e, por isso, não pode ser vazio.
- **Fluxo:** é formado por um conjunto de "Passo", "Sub-diagrama", "Filtro"e "Loop", e é representado por um retângulo não-contínuo. O objetivo é representar os uxos de casos de uso.
- **Seta do Fluxo:** representada por uma seta contínua que indica a sequência de ações e vericações. Ele pode relacionar um "Passo"a outro, ou um "Passo"a um "Sub-diagrama", ou um "Sub-diagrama"a um "Passo", entre outras possibilidades.
- **Filtro:** realiza uma seleção entre os dados pertencentes a uma classe de equivalência

associada a ele. Ele é representado por um triângulo, que possui um dos vértices voltado para o uxo, e uma expressão que descreve as classes de equivalência;

- **Sub-diagrama:** representado por dois retângulos sobrepostos que indicam que outro diagrama foi encapsulado. O objetivo é reutilizar diagramas previamente criados e reduzir a complexidade do diagrama; e
- **Loop:** interliga um "Passo"do uxo à um "Passo"anterior a ele. Possui um nome que o identica e é representado por uma seta não-contínua.

A Figura 10 apresenta um exemplo de uso para os elementos que podem ser utilizados na descrição do Test Script Diagram. Nessa figura, é possível ver um subdiagrama (Cadastro de Usuário), um passo (Clicar no botão inserir), uma seta de fluxo que liga o subdiagrama ao passo, além do elemento loop que indica que após a execução do passo, a execução do teste deve retornar para a etapa de cadastro de usuário.

Figura 10 – Exemplo de elementos do trabalho de (GOIS *et al.*, 2010)



Fonte: Gois *et al.* (2010).

3.4.3 Trabalho de Chen e Li (2010)

(CHEN; LI, 2010) definiram um modelo baseado em estados, chamado de Autômato Finito de Interação (IFA - sigla em inglês), para formalizar caso de uso. Baseado no IFA, um grupo de critérios de teste é denido e é representado para geração de casos de teste. O principal objetivo desses critérios é identicar se os uxos especicados nos casos de uso são suportados no produto nal. O IFA possui apenas estados e transições entre estados e os critérios de teste podem

representar:

- **Cobertura de estados:** Cada estado é acessado pelo menos uma vez;
- **Cobertura de transições:** Cada transição ocorre pelo menos uma vez;
- **Cobertura dos caminhos:** Cada caminho é percorrido, também, pelo menos uma vez; e
- **Cobertura dos caminhos restritos:** Cada caminho no IFA é percorrido pelo menos uma vez com restrições onde cada transição é acessada n vezes.

De acordo com o critério escolhido para ser testado, um conjunto de caminhos no modelo será gerado. É importante ressaltar que, seguindo a ideia de autômatos, os caminhos têm um estado inicial e um estado nal. Com a cobertura de transição, cada uxo do caso de uso pode ser testado.

3.4.4 Trabalho de Siqueira (2010)

TaRGeT (Test and Requirements Generation Tool) é uma ferramenta criada por (SIQUEIRA, 2010) que tem como objetivo gerar cenários de teste a partir de casos de uso. Os cenários de teste são obtidos a partir de uma denição formal do sistema a ser testado. Um modelo para descrição de casos de uso foi denido, e, nele é possível descrever ações do usuário, do sistema e pré-condições para cada passo do caso de uso. É possível, ainda, descrever passos alternativos a passos já mapeados.

A ferramenta possui dois módulos principais. O primeiro, TaRGet Test Case Generation, realiza a geração de suítes de teste que podem ser exportadas para diversos formatos, além de permitir a extensão dessa funcionalidade para novos formatos. O segundo módulo, TaRGeT On The Fly Generation, é um plugin que tem como objetivo gerenciar os casos de teste gerados pelo primeiro módulo. Esse gerenciamento corresponde a inserção de ltros e a exportação da suíte de testes para diversos formatos.

Com o intuito de evoluir esse trabalho e permitir que fossem gerados casos de teste para LPS, (BARROS *et al.*, 2011) definiu uma LNC para descrever os casos de uso que servem de insumo para a TaRGeT. Essa linguagem é um subconjunto da língua inglesa, portanto, as restrições definidas são baseadas na estrutura sintática e no vocabulário dessa língua.

3.4.5 Trabalho de Bertolino e Gnesi (2003)

(BERTOLINO; GNESI,) deniram um template de caso de uso, o PLUC (Product Line Use Case) que é a extensão de outro já existente, o de (COCKBURN, 2001). Essa extensão

foi definida para permitir que sejam inseridas informações de variabilidade das linhas de produtos de software. Essa representação é possível devido ao uso de tags que definem três tipos de variações: alternativas, opcionais e paramétricas. Uma variação é dita alternativa quando se deve escolher uma opção entre várias. Ela é dita opcional quando pode estar ou não presente e é dita paramétrica quando está associada ao valor atual do parâmetro dos requisitos para o produto específico.

Os autores também definiram um método para geração de cenários de teste, o PLUTO (Product Line Use Case Test Optimization) (BERTOLINO; GNESI,). Esse método utiliza o PLUC e a descrição em linguagem natural como insumos para a automação da geração dos cenários de teste. O PLUTO expandiu o método de partição de categorias (MYERS *et al.*, 2011) para permitir a inserção de variabilidade e a instanciação de casos de teste para um produto específico da linha. Assim, ele gera testes com base nas categorias que são extraídas dos casos de uso.

3.4.6 Trabalho de Neto (2011)

(NETO, 2011) propôs uma ferramenta para gerar e gerenciar cenários de teste para Linhas de Produtos de Software. Ele utilizou um modelo de teste para gerar os artefatos de teste e suas dependências.

A geração dos testes se dá a partir de casos de uso. O método que faz a geração dividiu o objetivo de um caso de uso em sub-objetivos. Como resultado dessa operação, cenários de teste são criados para cada fragmento (sub-objetivo), assegurando, assim, a cobertura de um caso de uso.

3.4.7 Trabalho de Santos (2013)

O trabalho de (SANTOS, 2013) é a base para este trabalho, e por isso será um pouco mais detalhado neste capítulo. Ele definiu um template de caso de uso, chamado CAPLUC (Context Aware software Product Line Use Case template), para descrever linhas de produtos de software sensíveis ao contexto. Também foi definido por ele um método, o ChAPTER (ContextAware software Product line TEsting geneRation method), para gerar cenários de teste para uma LPSSC.

3.4.7.1 CAPLUC

O CAPLUC é um template de caso de uso para descrever Linhas de Produtos de Software Sensíveis ao Contexto. Ele foi definido em formato tabular e é composto por uma tupla de doze elementos: [Nome, Caso de Uso Estendido, Ponto de Extensão, Categoria de Reuso, Restrição de Contexto, Resumo, Atores, Pré-condição, Pós-Condição, Passos, Fluxos Alternativos, Sumário de Variações]. Além disso, é possível definir variáveis globais e locais. É possível ver na Figura 11 uma representação do template com as descrições dos principais campos.

Os campos específicos para associar informações de linhas de produtos de software e informações de sensibilidade ao contexto são melhor detalhados abaixo:

- **Categoria de Reuso:** especifica se um caso de uso é obrigatório, opcional ou alternativo;
- **Restrição de Contexto:** define o contexto no qual o caso de uso é aplicado;
- **Passos:** é possível indicar a variabilidade da linha através da numeração dos passos. Além disso, é necessário indicar qual o ponto de variação a que um passo alternativo está associado;
- **Sumário de Variações:** identifica as variações que afetam o caso de uso.

3.4.7.2 ChAPTER

O ChAPTER (ContextAware software Product line TEsting geneRation method) é um método para geração de cenários de teste para Linhas de Produtos de Software Sensíveis ao Contexto. Essa geração se dá a partir de descrições textuais de casos de uso. O método foi criado como uma extensão do método PLUTO, citado na Seção 3.4.5, para tratar LPSSC, pois o PLUTO gera cenários de teste para LPS sem a necessidade de um modelo intermediário.

O ChAPTER gera testes para as etapas de Engenharia de Domínio e Engenharia da Aplicação, descritas na Seção 2.2. Na etapa de Engenharia de Domínio, são gerados cenários de teste a partir de descrições de casos de uso próprias da linha, chamados SLICEs (Software product LIne Contextual use casE). A partir dos SLICEs, são gerados os cenários de teste da linha. A Figura 12 apresenta um cenário de teste gerado pelo ChAPTER para o caso de uso Mostra Documentos, que será apresentado no Capítulo 4, pertencente a linha Mobiline.

No cenário de teste apresentado na Figura 12, é possível perceber o caso de uso associado ao cenário de teste, bem como as categorias identificadas pelo método. A primeira

Figura 11 – Template de caso de uso definido em (SANTOS, 2013)

Elemento		**Descrição**	
Nome		*Nome do caso de uso. O nome deve refletir o objetivo do caso de uso*	
Caso de Uso Estendido		*Nome do caso de uso cujo comportamento é estendido por este caso de uso*	
Ponto de Extensão		*Local do caso de uso estendido onde este caso de uso atua*	
Categoria de Reuso		*Especificar o quanto o caso de uso é Obrigatório, Opcional ou Alternativo {colocar o nome dos casos de uso alternativos aqui}*	
Restrição de Contexto		*Descrição da condição de contexto que precisa ser verdadeira para que este caso de uso seja passível de execução*	
Resumo		*Resume o objetivo do caso de uso*	
Atores		*Descreve os atores do caso de uso, tanto os primários (que iniciam o caso de uso) quanto os secundários (que podem participar do caso de uso)*	
Pré-condição		*Especifica uma ou mais condições que devem ser verdadeiras no início do caso de uso*	
Pós-condição		*Especifica a condição que sempre é verdadeira ao final do caso de uso se a sequencia principal foi seguida*	
Passo		**Usuário**	**Sistema**
[Nome do Ponto de Variação] (Alt) [Nome da Variante]	*Número do Passo*	*Ação do Usuário*	*Resposta do Sistema*
Fluxos Alternativos			
Restrição para a execução do fluxo alternativo. No caso do produto, aqui também podem ser colocadas restrições de contexto para influenciar o comportamento da aplicação		*Descrição dos passos alternativos*	
Sumário das Variações Alternativas			
[Nome do ponto de variação alternativo]: *Questão descritiva*		***[Nome da Variante Alternativa 1]*** *(Restrição de Contexto: contexto que deve ser verdadeiro para a variante ser passível de escolha)*	*Identificação do passo na forma "Passo X"*
		[Nome da Variante Alternativa 2] *(Restrição de Contexto: contexto que deve ser verdadeiro para a variante ser passível de escolha)*	*Identificação do passo na forma "Passo X"*
		[Nome da Variante Alternativa N] *(Restrição de Contexto: contexto que deve ser verdadeiro para a variante ser passível de escolha)*	*Identificação do passo na forma "Passo X"*
Sumário das Variações Opcionais			
[Nome do ponto de variação opcional]: *Questão descritiva*		***[COM Nome da Variante Opcional]*** *(Restrição de Contexto: contexto que deve ser verdadeiro para a variante ser passível de escolha)*	*Identificação do passo na forma "Passo X"*
		[SEM Nome da Variante Opcional] *(Restrição de Contexto: contexto que deve ser verdadeiro para a variante ser passível de escolha)*	*Identificação do passo na forma "Passo X"*

Fonte: Santos (2013).

categoria trata dos cenários (ou fluxos) do caso de uso. A segunda indica a presença do ponto de variação "Imagem" e como contexto necessário para a execução desse cenário a "BATERIA_MEDIA OU BATERIA_ALTA". É possível perceber associações semelhantes nas outras categorias identificadas.

Na Engenharia de Aplicação, os casos de uso da aplicação são conhecidos como PiECES (Product contextual usE CasEs). Quando um PiECE é modicado, escolhendo uma feature entre features alternativas, novos cenários de teste são gerados. Esses cenários de teste são especícos do produto. Quando a feature é única para toda a linha, os cenários de teste são gerados a partir dos SLICEs.

Como última etapa do ChAPTER, estados são gerados e representam todas as congurações possíveis do produto. Essas congurações correspondem a identicação de features

Figura 12 – Cenário de teste gerado pelo ChAPTER

Comment		
CenarioTeste1		
Slice: Mostra Documentos		
Contexto: null		
Categoria	Opcao	Contexto
Cenarios	Principal	
Imagem	Imagem	BATERIA_MEDIA OU BATERIA_ALTA
Vídeo	Vídeo	BATERIA_ALTA
Texto	Sem formatação	

Fonte: Santos (2013).

ativas e a transição de estados é realizada por mudanças de contexto. Um estado é selecionado e com base nas features ativas é feito um recorte dos testes do produto, que dão origem à cenários de teste especícos da conguração. Assim, cada conguração possui um conjunto de cenários de teste associado.

3.4.8 Comparação entre os Métodos para Geração de Teste a partir de Caso de Uso

A Tabela 2 apresenta um comparativo entre os oito trabalhos encontrados. Todos apresentam caso de uso como insumo para geração de testes, porém quatro precisam de outros artefatos para auxiliar a geração dos testes a partir do caso de uso. Dos oito trabalhos, três possuem como objetivo gerar testes, que podem ser cenários de teste ou casos de teste para uma LPS ou LPSSC. Os outros trabalhos geram testes para aplicações convencionais.

Tabela 2 – Comparação entre os métodos de teste

	Insumo para geração dos testes	Descrição dos Casos de Uso	Geração automática de testes	LPS
Bertolino e Gnesi (2003)	Caso de uso	Linguagem Natural	Cenários de Teste	LPS
Nebut et al. (2006)	Caso de uso e Simulador de caso de uso	Linguagem Natural	Cenários de Teste	Não se aplica
Gois (2010)	Caso de uso e scripts de teste	Linguagem Natural	Caso de Teste	Não se aplica
Chen e Li (2010)	Caso de uso e autômatos	Linguagem Natural	Caso de Teste	Não se aplica
Siqueira (2010)	Caso de uso e scripts de teste	Linguagem Natural	Caso de Teste	Não se aplica
Barros (2011)	Caso de uso	Linguagem Natural Controlada	Não se aplica	Não se aplica
Neto (2011)	Caso de Uso	Linguagem Natural	Cenários de Teste	LPS
Santos (2013)	Caso de uso	Linguagem Natural	Cenários de Teste	LPSSC

Fonte: O autor.

Portanto, é possível perceber que nenhum dos trabalhos relacionados apresenta um método ou abordagem que possua como objetivo automatizar a geração dos testes para LPSSC de maneira que os testes gerados diminuam o esforço do testador no processo de criação dos mesmos, gerando-os mais próxima da execução. Assim, é possível notar que ainda é necessário um esforço para a geração de casos de testes para Linhas de Produtos de Software Sensíveis ao Contexto que, como apresentado no Capítulo 2, é uma linha de produtos de software com o intuito de desenvolver aplicações que se adaptam ou proveem um novo serviço baseado nas informações contextuais do usuário.

3.5 Conclusão

Este capítulo apresentou os conceitos associados a teste de software e casos de uso. Além disso, este capítulo tratou da especificação de casos de uso textuais utilizando uma LNC e os métodos para geração de testes a partir de casos de uso. Esses trabalhos relacionados foram encontrados em buscas na literatura que procurou por trabalhos que definiam linguagens naturais controladas ou definiam métodos para geração de testes a partir de casos de uso.

Com relação aos testes que são gerados, muitos geram cenários de testes a partir de casos de uso descritos em linguagem natural. Para alcançar um rigor maior na definição dos testes no que diz respeito a valores a serem utilizados, é necessário utilizar outros artefatos ou uma linguagem natural controlada.

4 CARNAUBA

Neste capítulo é descrita a linguagem natural controlada proposta neste trabalho, a CARNAUbA (Controlled nAtuRal laNguAge for context-aware software product lines Use cAses). Este capítulo está dividido assim: na Seção 4.1 é apresentada uma visão geral da linguagem; a Seção 4.2 trata da análise sintática executada na linguagem; na Seção 4.3 é apresentada a análise semântica; na Seção 4.4 são apresentados dois exemplos de uso; a Seção 4.5 aborda as limitações da linguagem; e na Seção 4.6 são apresentadas as conclusões.

4.1 Visão Geral

A linguagem descrita neste capítulo foi denominada CARNAUbA, acrônimo para *Controlled nAtuRal laNguAge for context-aware software product lines Use cAses*. Ela foi definida com base no template de caso de uso CAPLUC apresentado no Capítulo 3. Assim, cada elemento presente no caso de uso possui uma restrição na forma como deve ser preenchido ao seguir as definições da CARNAUbA.

A linguagem CARNAUbA difere das demais LNCs, pois trata questões associadas a LPSSC, o que não foi encontrado na literatura, e também por ser em português, o que também não foi encontrado na literatura. Como a linguagem se baseia no CAPLUC e ele pode ser utilizado tanto para LPS como para LPSSC, a linguagem também pode ser utilizada para LPS ou LPSSC.

Como a CARNAUbA é uma linguagem livre de contexto, ela possui regras de produção e tuplas com 4 valores para descrevê-la. Os valores da tupla representam variáveis, terminais, as regras de produção existentes e o símbolo inicial da linguagem. Todos os valores das tuplas podem ser encontrados nas regras de produção. Os terminais são símbolos que representam o fim de uma sequência de operações. As variáveis, por sua vez, são símbolos que permitem alcançar estados aumentando, assim, o número de palavras aceitas em uma determinada linguagem. As regras de produção representam as transições que podem existir entre as variáveis e os terminais. O símbolo inicial representa a variável que dá início à linguagem.

Sabendo disso, foi definida na CARNAUbA uma tupla para cada elemento existente. As tuplas definidas serão mostradas na seção de análise sintática deste capítulo. Em seguida, é apresentada a análise semântica das frases, assim como a lista de verbos aceitas na linguagem.

4.2 Análise Sintática

A análise sintática é responsável por indicar como devem ser as construções das frases em uma linguagem. No caso deste trabalho, em uma LNC. Logo, nesta seção são descritas as sintaxes de cada elemento presente no CAPLUC apresentado na Seção 3.4.7.1.

4.2.1 Nome do Caso de Uso

Esse elemento deve conter o nome do caso de uso que deve refletir o objetivo do mesmo. Assim, ele foi definido para seguir a estrutura apresentada na Tabela 3.

Essa estrutura diz que o caso de uso deve começar seguindo a definição da variável "NOME" que diz que o elemento deve iniciar com uma letra maiúscula e que em seguida podem existir letras minúsculas, maiúsculas, números, underline e hífen. Conforme apresentado na Tabela 3, um nome de caso de uso que pode ser especificado pela linguagem é "Autenticação".

Tabela 3 – Definição da sintaxe do elemento "Nome do caso de uso"

Elemento	Descrição
Nome do Caso de Uso	*Caso de Uso: NOME (ID | INT) ** *NOME: ('A'..'Z') ('a'..'z'|'A'..'Z'|'0'..'9'|'_'|'-') ** *ID: ('a'..'z' | 'A'..'Z'|'_') ('a'..'z'|'A'..'Z'|'0'..'9'|'_'|'-') ** *INT:'0'..'9'+*
Exemplos	Autenticacao Acesso ao Contexto Mostra Documentos Mostra_Textos

Fonte: O autor.

Continuando a analisar a estrutura, ela diz que após encerrar a variável "NOME", podem existir duas opções: um "ID" ou um "INT". Essas variáveis produzem símbolos terminais diferentes da variável "NOME". Isso significa que após o nome podem vir números ou outras expressões que, nesse momento, podem iniciar com letras minúsculas, maiúsculas ou underline e são continuadas por letras minúsculas, maiúsculas, números, underline e o hífen. Seguem alguns exemplos de nomes de caso de uso que podem ser obtidos com essas regras de produção e que também são apresentados na Tabela 3: "Acesso ao Contexto", "Mostra Documentos", "Mostra_Textos".

4.2.2 Caso de Uso estendido

Esse elemento tem como objetivo identificar o caso de uso que tem o comportamento estendido por um caso de uso já descrito. Como ele deve identificar o nome de outro caso de uso existente, a gramática a ser seguida deve ser a mesma utilizada para expressar o nome de um caso de uso apresentada na Seção 4.2.1.

4.2.3 Ponto de Extensão

O intuito desse elemento é identificar o termo que indica a chamada para o caso descrito, considerando que ele estende outro caso de uso. Para isso, esse elemento deve seguir as restrições contidas na Tabela 4.

A estrutura desse elemento diz que ele deve ser iniciado com um "<". Em seguida, deve existir uma letra minúscula acompanhada de outras letras minúsculas ou de números ou de *underline*. Por fim, o elemento deve ser encerrado com um ">". O uso dos sinais "<" e ">" foi definido baseado no template definido por (GOMMA, 2004). Um exemplo de expressão gerada por essas regras de produção é "<tipo_de_midia>", como apresentado na Tabela 4.

Tabela 4 – Definição da sintaxe do elemento "Ponto de extensão"

Elemento	Descrição
Ponto de Extensão	*Ponto de extensão: "<" ID ">"* *ID: ('a'..'z') ('a'..'z'|'0'..'9'|'_') **
Exemplo	<tipo_de_midia>

Fonte: O autor.

4.2.4 Categoria de Reuso

Como são casos de uso de uma linha de produtos de software, eles podem ser obrigatórios, opcionais e alternativos, variando de acordo com a presença nos produtos desenvolvidos. Um caso de uso obrigatório deve existir em todos os produtos da linha. Os opcionais podem ou não existir nos produtos gerados e os alternativos indicam que casos de uso podem descrever ações semelhantes e, por isso, um ou mais casos de uso dentro de um grupo de casos de uso alternativos podem ser selecionados.

Portanto, o elemento categoria de reuso deve ser apenas preenchido com alguns desses três valores "Obrigatório", "Alternativo" e "Opcional" representando as opções descritas

anteriormente. Os valores são apresentados na Tabela 5.

Tabela 5 – Definição da sintaxe do elemento “Categoria de Reuso”

Elemento	Descrição
Categoria de Reuso	*Categoria: “Obrigatório” | “Alternativo” | “Opcional”*
Exemplos	Obrigatório Alternativo Opcional

Fonte: O autor.

4.2.5 Restrição de Contexto

Segundo a definição do CAPLUC, a restrição de contexto associada ao caso de uso deve descrever a condição de contexto que precisa ser verdadeira para que o caso de uso possa ser executado. Assim, para descrever essas condições, a estrutura definida na CARNAUbA segue as restrições apresentadas na Tabela 6.

As regras de produção iniciam com a variável da gramática livre de contexto “Restrição de Contexto”. A partir dela, é possível definir uma variável, que é composta de “IDs” e pode ser iniciado com letra maiúscula ou minúscula, além do caractere underline. Em seguida, essa variável pode ter outras letras, números e também o caractere underline. Assim, uma restrição de contexto pode ser “BATERIA_ALTA”, como apresentado na Tabela 6.

Tabela 6 – Definição da sintaxe do elemento “Restrição de Contexto”

Elemento	Descrição
Restrição de Contexto	*Restrição de Contexto: '(' variável (sinal val)? exp? ')' (exp)? | variável (sinal val)? (exp)?* *exp: (unário | binário) Restrição de Contexto* *unário: 'e' | 'ou'* *binário: '!'* *variável: ID ID** *sinal: '<' | '>' | '<=' | '>=' | '='* *val: ID | INT* *ID: ('a'..'z' | 'A'..'Z' | '_') ('a'..'z'|'A'..'Z'|'0'..'9'|'_')** *INT : '0'..'9'+*
Exemplos	BATERIA_ALTA temperatura > 50 (temperatura > 50) e (umidade > 30)

Fonte: O autor.

Podem existir também expressões lógicas iniciadas com uma variável e continuadas

com “>”, “<”, “<=”, entre outros caracteres. Após esses caracteres, devem existir valores ou outras variáveis. Portanto, pode ser especificada a seguinte restrição contextual: “temperatura > 50”.

É importante ressaltar que as expressões podem ser separadas por “(” e “)” e associadas pelos operadores lógicos “e”, “ou” e o “!”, que representa a negação. A expressão “(temperatura > 50) e (umidade > 30)” é um exemplo de restrição de contexto, como apresentado na Tabela 6.

4.2.6 Resumo

O Resumo permite que seja feita uma descrição do objetivo do caso de uso. Assim, é possível identificar mais claramente quais interações o caso de uso descreve. Portanto, devem ser descritas seguindo as restrições definidas para uma ação no elemento que descrevem os passos a serem executados no caso de uso, segundo é explicado na Seção 4.2.10.3.1.

4.2.7 Atores

Esse elemento permite identificar os atores que interagem com o caso de uso, além de identificá-los como ator primário, responsável por disparar o caso de uso, ou secundário, interage em outros momentos do caso de uso. Para entender como os atores devem ser especificados, as definições são demonstradas na Tabela 7.

Tabela 7 – Definição da sintaxe do elemento “Atores”

Elemento	Descrição
Ator	*Atores: (ID ID* '(' (“Primário” | “Secundário”) ')') ** *ID:('A'..'Z') ('a'..'z'|'A'..'Z'|'0'..'9'|'_') **
Exemplos	Usuário (Primário) Servidor (Secundário)

Fonte: O autor.

Seguindo as especificações, os nomes dos atores devem ser iniciados com letras maiúsculas, e podem ser seguidos de letras minúsculas, maiúsculas, números ou do caractere *underline*. Logo em seguida, deve ser especificado o tipo do ator “Primário” ou “Secundário” dentro de parênteses. Caso exista mais de um ator, eles também podem ser descritos sequencialmente e seguindo as mesmas definições do primeiro ator. Exemplos de termos gerados pela linguagem são encontrados na Tabela 7.

4.2.8 Pré-condição

Esse elemento descreve as condições necessárias em que o sistema deve estar para que o caso de uso possa ser executado. O objetivo desse campo é especificar uma ou mais condições que devem ser verdadeiras antes do início da execução do caso de uso.

A pré-condição deve seguir as restrições sintáticas para descrever uma ação definidas para os passos que é descrita na Seção 4.2.10.3.1.

4.2.9 Pós-condição

Na pós-condição é descrita a situação em que o sistema deve estar após a execução da sequência principal do caso de uso. Assim como a pré-condição, ela segue a estrutura dos elementos que descrevem uma ação nos passos de um caso de uso. Portanto, as regras de produção desse elemento seguem as regras apresentadas na Seção 4.2.10.3.1.

4.2.10 Passo

Esse elemento possui outros elementos associados, além de possuir três interpretações diferentes que variam de acordo com o tipo de frase descrita no caso de uso. Elas podem ser frases de ação, de condição ou de repetição. Essas formas de descrever esse elemento e as interpretações associadas são tratados nas próximas subseções.

4.2.10.1 Ponto de Variação e Variante

É possível associar um passo do caso de uso a um ponto de variação e a uma variante. Esses elementos são importantes, pois permitem inserir a variabilidade presente na linha dentro do caso de uso. Por exemplo, um passo que descreve a exibição de imagens pode estar associado ao ponto de variação “Mostra Documentos” e a variante “Imagens”.

Dessa forma, para inserir informações sobre o ponto de variação e a variante associados a um passo, o usuário deve seguir as regras de produção apresentadas na Tabela 8.

De acordo com essas regras, para descrever um ponto de variação é necessário inserir, inicialmente, o caractere “[”. Em seguida deve ser informado o nome do ponto de variação que deve iniciar com letra maiúscula seguida de outras letras, números e do caractere *underline*. A definição do ponto de variação é encerrada com a inserção do caractere “]”. Por exemplo, [Mostra Documentos] na Tabela 8 identifica o ponto de variação associado a um passo.

Tabela 8 – Definição da sintaxe do elemento “Ponto de Variação e Variante”

Elemento	Descrição
Ponto de variação e Variante	*Ponto de Variação: “[” ID ID* “]” Variante* *Variante: “[”ID ID* “]”* *ID: ('A'..'Z') ('a'..'z'|'A'..'Z'|'0'..'9'|'_') **
Exemplos	[Mostra Documentos] [Mostra Textos]

Fonte: O autor.

Após definir o ponto de variação, é necessário definir a variante associada a ele. Para defini-la, o usuário deve seguir a mesma sequência de caracteres utilizados para descrever um ponto de variação. Desse modo, a identificação da variante deve iniciar com o caractere “[” que deve ser acompanhado de uma letra maiúscula seguida por letras minúsculas, maiúsculas, números ou o caractere *underline*. Para encerrar a identificação da variante, o caractere “]” deve ser inserido. Como exemplo, tem-se [Mostra Textos], exibido na Tabela 8.

4.2.10.2 Número do Passo

Esse elemento não informa apenas o número do passo descrito, mas fornece informações sobre a obrigatoriedade do mesmo, além de identificar se o passo é alternativo, opcional ou está associado a uma informação de contexto, conforme descrito no CAPLUC (SANTOS, 2013). Essas indicações facilitam a percepção no que diz respeito a adequação da CARNAUbA para a descrição de casos de uso para uma LPSSC. Vale ressaltar que é possível perceber em outros elementos do CAPLUC a adequação da linguagem para descrever casos de uso para uma LPSSC. A sintaxe do elemento “Número do Passo” é apresentada na Tabela 9.

Tabela 9 – Definição da sintaxe do elemento “Número do Passo”

Elemento	Descrição
Número do Passo	*Número do passo: ('(') INT (')') ('*')? | INT ('*')?* *INT: '0'..'9'+*
Exemplos	1 2* (3) (4)*

Fonte: O autor.

A identificação de um número de um passo pode ser feita apenas por um numeral ou pode ser feita por um numeral seguido por um “*” que informa que o passo contém restrições de contexto para ser executado. O passo também pode ser identificado por um número entre parênteses para indicar que este passo é opcional. Um passo também pode ser opcional e com

restrições de contexto. Desse modo, o número existente entre os parênteses é acompanhado por um "*". Exemplos de números que podem ser especificados com a linguagem são encontrados na Tabela 9. Caso o número seja repetido, ele indica que tais passos são alternativos, representando assim a escolha de um deles.

4.2.10.3 Ações do Ator e do Sistema

Nesses elementos são descritas as ações dos atores que interagem com o sistema descrito nos casos de uso, bem como a resposta do sistema diante dos estímulos produzidos pelos atores. Existem quatro formas de se descrever uma ação do usuário ou do sistema. A primeira delas refere-se a uma ação direta do ator ou do sistema. A segunda é quando se refere a um ponto de extensão. A terceira define como devem ser escritos condicionais usando a linguagem e a quarta diz como devem ser escritos laços de repetição.

4.2.10.3.1 Ação

Nesta seção é apresentada a forma de se descrever uma ação em um passo de um caso de uso. Ressaltando que uma não se contém o comportamento condicional ou de repetição, sendo essas formas descritas posteriormente. Essa ação também não representa um ponto de extensão. As regras de produção são descritas na Tabela 10.

Dessa maneira, toda frase que representa uma ação descrita seguindo as restrições sintáticas da CARNAUbA deve conter um sujeito e um predicado assim como na língua portuguesa. Porém, o sujeito não pode ser oculto e nem indeterminado, devendo ser um ator ou o sistema. É obrigatória a presença de um artigo antes do sujeito. Também pode existir um complemento para o sujeito, e esse complemento deve ser iniciado por uma preposição. Alguns exemplos de sujeitos aceitos pela linguagem: "O Usuário", "O Administrador do Sistema". A frase definida com essa gramática é estruturada da seguinte maneira:

Sentença -> Sujeito + Predicado

Sujeito -> Artigo + Núcleo do Sujeito + {Complemento}

Predicado -> {Negação} + Verbo + Complemento

O predicado, por sua vez, pode conter um "não" para indicar uma negação, e é obrigatória a presença de um verbo, além de necessitar de um complemento. Só podem ser utilizados os verbos aceitos pela linguagem. A lista de verbos pode ser vista no apêndice "A". O complemento deve conter pelo menos um artigo e um nome. Por exemplo, "o login". Mas,

Tabela 10 – Definição da sintaxe de uma “Ação” do elemento “Passo”

Elemento	Descrição
Ação do Ator e do Sistema	*Sentença: Sujeito Predicado “.”* *Sujeito: Artigo Núcleo_do_Sujeito (Complemento)?* *Artigo: Artigo_Definido | Artigo_Indefinido* *Artigo_Definido: “O” | “A” | “Os” | “As”* *Artigo_Indefinido: “Um” | “Uns” | “Uma” | “Umas”* *Núcleo_do_Sujeito: ID_SUJEITO** *Complemento: ID_COMPLEMENTO ** *Predicado: Verbo Complemento_Predicado* *Verbo: ID_VERBO** *Complemento_Predicado: ID_COMPLEMENTO ** *ID_SUJEITO: ('A'..'Z') ('a'..'z'|'A'..'Z'|'0'..'9'|'_') ** *ID_COMPLEMENTO: ('a'..'z') ('a'..'z'|'A'..'Z'|'0'..'9'|'_') ** *ID_VERBO: ('a'..'z') **
Exemplos	O Usuário digita o login. O Sistema exibe uma mensagem de sucesso. O Usuário digita o login e a senha.

Fonte: O autor.

também podem existir mais de uma informação como “o login e a senha” ou ainda “o login, a senha e o captcha”.

Os verbos aceitos na linguagem foram definidos com base na análise de casos de uso de linha de produtos de software reais (Arcade Game e GoPhone), além da LPSSC Mobiline. Cada verbo aceito contém um nome, um significado, além de comandos a serem utilizados na geração dos testes, conforme apresentado no Capítulo 5, e informações sobre complementos que podem ser extraídos para a geração dos testes.

Além disso, os verbos estão conjugados sempre na terceira pessoa do singular, considerando que as frases são sempre descritas com o sujeito nessa pessoa. Por exemplo, “O Usuário” ou “O Sistema” sempre representam sujeitos na terceira pessoa do singular como existente no português. Dessa forma, não são aceitos verbos conjugados em outras pessoas.

4.2.10.3.2 Ponto de Extensão

O objetivo desse elemento é identificar, no caso de uso que está em execução, o local e o momento em execução em que o outro caso de uso, que vai estendê-lo, deve ser chamado para executar. São encontradas na Tabela 11, as regras de produção para auxiliar a identificar o local do ponto de extensão.

As regras de produção descritas acima mostram que a identificação do ponto de

Tabela 11 – Definição da sintaxe de uma “Ponto de Extensão” do elemento “Passo”

Elemento	Descrição
Ponto de Extensão	*Ponto de Extensão: “Ponto” “de” “Extensão” “:” “<” ID “>”* *ID: ('a'..'z') ('a'..'z'|'0'..'9'|'_') **
Exemplos	Ponto de extensão: <tipo_de_midia>

Fonte: O autor.

extensão deve iniciar com o nome “Ponto” seguido da preposição “de”, do substantivo “Extensão” e dos caracteres “:” e “<”. Eles foram definidos separadamente para não obrigar a existir apenas um “espaço” entre as palavras.

Logo em seguida, devem ser inseridos os nomes dos pontos de extensão. Esses nomes devem ser iniciados com letras minúsculas, seguido de outras letras minúsculas, números ou *underline*. Após encerrar a definição do nome, deve existir outro caractere, o “>”, e, assim, identificar que o ponto de extensão foi definido. Por exemplo, “Ponto de extensão: <tipo_de_midia>” é uma expressão que pode ser gerada pelas regras apresentada na Tabela 11.

4.2.10.3.3 Condição

Na descrição dos passos de um caso de uso também pode ser descrita uma condição para que uma determinada ação seja executada. Assim, é possível descrever verificações e um comportamento do sistema mais próximo do real. Para fazer isso utilizando a CARNAUbA, o usuário deve seguir as restrições apresentadas na Tabela 12.

Tabela 12 – Definição da sintaxe de uma “Condição” do elemento “Passo”

Elemento	Descrição
Condicional	*Condicional: “Se” Condição “,” “então” ação* *Condição: ID ID** *ID: ('a'..'z') ('a'..'z'|'0'..'9'|'_') **
Exemplos	Se temperatura > 50, então o sistema exibe uma mensagem de alerta

Fonte: O autor.

Desse modo, seguindo as definições da Tabela 12, é possível perceber que uma frase que indica condição é iniciada com um “Se” seguida da condição associada a uma vírgula. Para representar a ação que deve ser executada por um ator ou pelo sistema, a frase é identificada pela ocorrência da palavra “então”. Por fim, uma ação deve ser descrita conforme apresentado no item 4.2.10.3.1. Por exemplo, “Se a temperatura > 50, então o sistema exibe uma mensagem de alerta.”, como pode ser visto na Tabela 12.

4.2.10.3.4 Repetição

Existe a necessidade de serem inseridas frases que representem o comportamento repetitivo do sistema quando ele se encontrar dentro de algumas condições. Para atingir tal objetivo, na linguagem CARNAUbA foi especificada uma forma para descrever esse tipo de frase. As restrições para esse elemento são encontradas na Tabela 13.

Dessa forma, para indicar a repetição, a frase deve iniciar com o verbo "Repetir" seguido da indicação dos passos ("o passo" ou "os passos") que serão repetidos quando uma condição for verdadeira. Essa condição é descrita na sequência. A condição pode ocorrer um número exato de vezes ou por condições como as descritas no item anterior ("temperatura > 50"). Portanto, como exemplo de frase aceita pela CARNAUbA, tem-se "Repetir o passo 3 10 vezes". Mais exemplos são descritos na Tabela 13.

Tabela 13 – Definição da sintaxe de uma "Repetição" do elemento "Passo"

Elemento	Descrição
Repetição	*Repetição: "Repetir" ("o" "passo" INT | "os" "passos" INT a INT) (INT "vezes" | "até" condição)* *INT: '0'..'9'+*
Exemplos	Repetir o passo 3 10 vezes Repetir os passos 3 a 5 até que o sistema identifique a localização do usuário

Fonte: O autor.

4.3 Análise Semântica

A análise semântica de uma frase representa a interpretação da mesma para extrair informações. Ao se realizar uma análise semântica, uma pessoa está tentando entender a informação contida na frase. Esse mesmo procedimento pode ser automatizado, incluindo também a análise semântica de LNCs.

O processo para realizar a análise semântica da CARNAUbA procura analisar o verbo, bem como a relação com o sujeito da frase e a sua transitividade. A análise desses fatores contribui para facilitar a interpretação da frase. Porém, apenas o elemento que contém os passos a serem executados pelo caso de uso possuem relevância semântica, pois eles influenciam diretamente nas ações a serem executadas e na extração das informações para os testes. Assim, a análise semântica do elemento "Passo" da CANAUbA segue o processo apresentado na Figura 13.

Na análise semântica é feita uma verificação do tipo de frase que foi descrita no caso de uso. Primeiramente, é feita uma verificação para indicar se a frase contém um ponto de extensão. Caso seja verdadeiro, o nome do ponto de extensão é extraído e a análise nesse passo se encerra. Caso contrário, é verificado se a frase representa um laço, que é uma estrutura de repetição. Caso essa condição seja verdadeira, a primeira ação a ser realizada é a identificação dos passos a serem repetidos. Em seguida, são indentifcadas as condições de parada. As ações a serem realizadas são extraídas como uma ação que executa sem condição.

Se o tipo da frase não for o de uma estrutura de repetição, deve ser verificado se a frase indica uma ação condicional. Logo, se a frase for um condicional, o primeiro passo a ser executado é a extração da condição para se executar a ação. Essa informação deve ser armazenada e a continuação da frase que indica a ação a ser executada passa a ser verificada. Esse procedimento é o mesmo que é executado quando um passo descreve uma ação sem condicional.

O primeiro passo para verificar uma ação conforme apresentado na Figura 13 é a identificação da existência de um verbo na oração. Esses verbos foram pré-cadastrados, mas é possível aumentar o número deles cadastrando-os junto com as informações necessárias para a análise semântica.

Se não existir um verbo, o processamento é interrompido porque não é possível fazer a análise semântica por não seguir a estrutura sintática do elemento “Passo”. Caso exista, o sujeito da oração é identificado permitindo, assim, descobrir o seu papel no caso de uso, se é um ator ou o sistema quem está interagindo. Se for o sistema que está reagindo a uma ação de um ator, o verbo pode ser interpretado de maneira diferente da forma para uma ação de um ator.

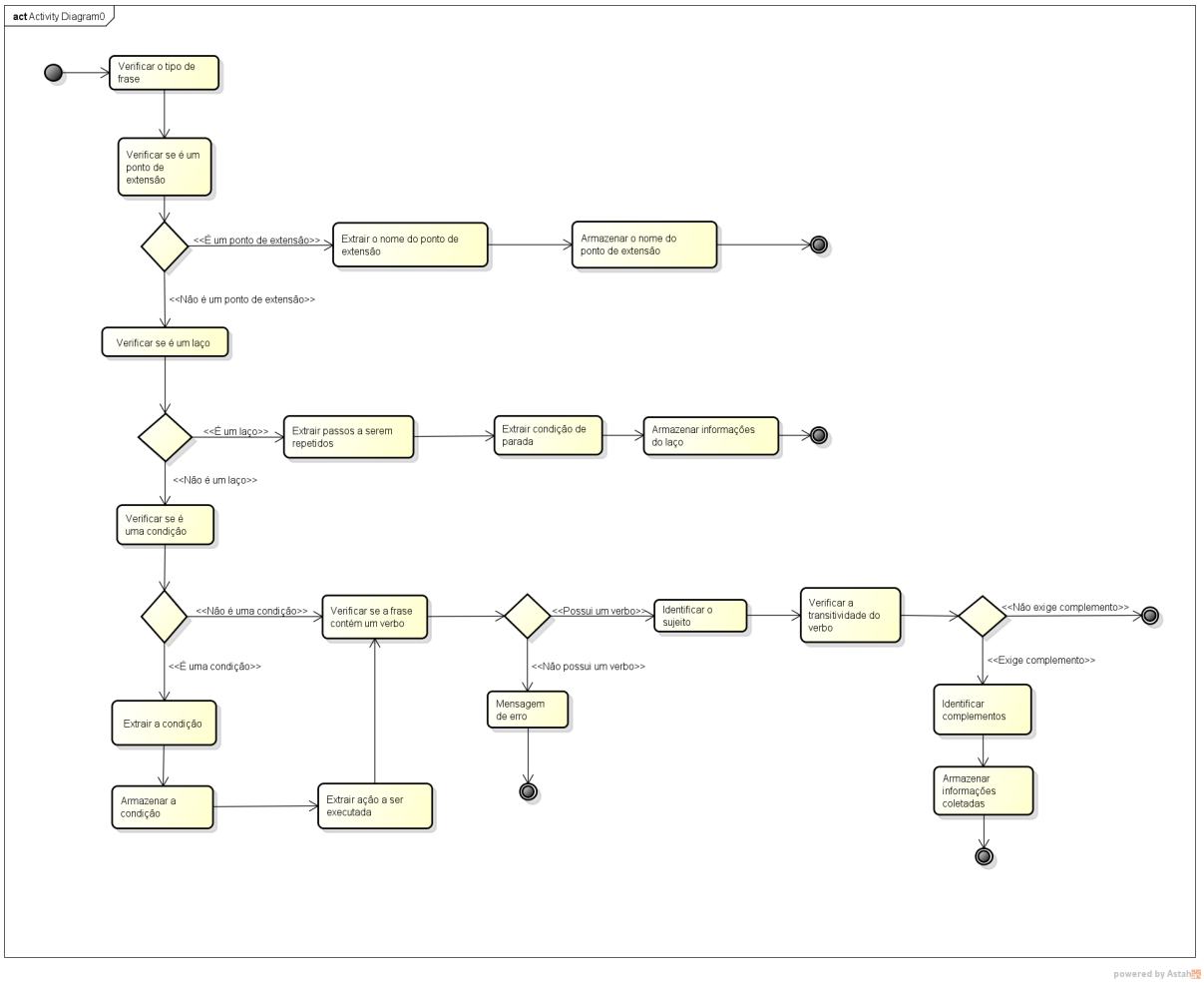
Após a identificação do sujeito, é verificada a transitividade do verbo, pois o mesmo pode ter transitividade diferente dependendo do sujeito da oração. Por isso, esse processo vem após a identificação do sujeito. Por exemplo, o verbo “exibe” associado a um ator indica que não possui complemento, porém associado ao sistema indica que existe a apresentação de uma informação para o usuário.

Em seguida, caso a análise indique a necessidade de complementos, é possível identifica-los como elementos a serem preenchidos na aplicação (entradas) ou como respostas do sistema (saídas). Essa identificação ocorre ao se analisar o sujeito da oração. Se for uma ação de um ator, o complemento é atribuído como elemento de entrada. Caso contrário, é uma resposta do sistema.

É possível ainda identificar valores para os elementos de entrada ou saída. Isso pode

ocorrer quando a oração descrita possui valores posteriores aos elementos. Para saber qual o nome de um elemento ou se existem muitos elementos é preciso fazer a análise sintática. Por exemplo, “O Usuário digitou o login e a senha” ou “O Sistema exibe a tela inicial da aplicação”. No primeiro caso, são identificados dois elementos para a entrada, o login e a senha. No segundo caso, o complemento seria “a tela” e o que aparece na sequência seria um possível valor para o teste.

Figura 13 – Processo de análise semântica



Fonte: O autor.

4.4 Exemplo de Uso

Nesta Seção são apresentados dois exemplos de casos de uso escritos seguindo as restrições da CARNAUbA. O primeiro exemplo é um caso de uso escrito originalmente na linguagem e o segundo é um exemplo de conversão de um caso de uso existente para a linguagem.

O caso de uso que será escrito utilizando a CARNAUbA é um caso de uso do GREat

Tour, produto de software desenvolvido a partir da linha Mobiline, conforme apresentado no Capítulo 2. O segundo é um caso de uso da LPS, pois não utiliza informações contextuais, Arcade Game Maker (MAKER, 2014) que já havia sido criado pelo *Software Engineering Institute* e pela Universidade de *Carnegie Mellon*.

4.4.1 Caso de Uso: Mostra Documentos

Esse caso de uso é um caso de uso da linha Mobiline que tem como objetivo descrever as ações a serem executadas para que sejam exibidos textos, imagens e vídeos em um guia de visitas móvel e sensível ao contexto. Na descrição dos passos desse caso de uso existe um ponto de extensão que deve chamar outros casos de uso (Mostra Textos, Mostra Imagens e Mostra Vídeo) para serem executados. Assim, esse caso de uso é estendido por outros três casos de uso.

O primeiro elemento a ser preenchido é o nome do caso de uso. Seguindo as restrições sintáticas impostas pela CARNAUbA, o elemento tem que iniciar com letra maiúscula e em seguida podem ser inseridos outros caracteres. Portanto, o nome definido é "Mostra Documentos".

Os elementos que associam um caso de uso a outro caso de uso, "Caso de Uso Estendido" e "Ponto de Extensão", não são preenchidos nesse exemplo, pois o caso de uso descrito não estende outro caso de uso. Como o caso de uso também não necessita de restrição de contexto, esse elemento não será preenchido.

A categoria de reuso de um caso de uso está associada a *feature* a qual ele possa estar associado. Nesse caso, como o caso de uso está associado a *feature* "Show Documents" apresentada na Figura 4 e ela é obrigatória, o caso de uso é obrigatório em todos os produtos da linha.

Os atores que interagem com o sistema descrito nos casos de uso também devem ser definidos. Nesse caso, quem interage com o sistema é o visitante e ele é o ator primário. Com base nessas informações e nas restrições sintáticas definidas na Seção 4.1.7, o ator desse caso de uso é "Visitante (Primário)".

O resumo desse caso de uso segue as restrições sintáticas do elemento "Passo", porém não é realizada uma análise semântica no mesmo. Assim, o resumo poderia ser: "O caso de uso contém os passos para exibição dos documentos associados ao ambiente onde o visitante se encontra."

Os elementos responsáveis pela pré-condição e pela pós-condição, assim como o

resumo, devem seguir as restrições sintáticas do elemento “Passo”, porém não são analisados semanticamente. Portanto, o elemento pré-condição foi definido assim: “O Sistema exibe o mapa do ambiente em que o Visitante se encontra.”. E a pós-condição deve ficar assim: “O Sistema exibe as informações do ambiente em que o Visitante se encontra.”.

Após o preenchimento desses elementos, o elemento “Passo”, bem como os elementos associados a ele devem ser descritos. O primeiro passo deve sempre ser uma ação do ator primário, pois nesse momento o caso de uso é disparado para ser executado. O caso de uso deve iniciar com a seleção de uma opção para visualizar as mídias do ambiente em que o visitante se encontra. Portanto, o primeiro passo deve ser “O Visitante seleciona a opção para visualizar as mídias do ambiente onde ele se encontra”.

Como resposta a essa interação, o sistema deve exibir algumas opções de informações, como textos sem formatação, textos na forma de nuvem de tags, imagens ou vídeos. O ponto de variação das opções que exibem textos é igual, porém as variantes são diferentes. E no caso das imagens e vídeos só existe ponto de variação, pois as mesmas são opcionais. Todas essas informações estão associadas à *features* da linha Mobiline (MOBILINE, 2014).

Cada uma dessas informações deve ser colocada em um passo separado e associada ao seu respectivo ponto de variação e variante, quando essa existir. No caso dos passos que contém pontos de variação iguais, eles devem ser passos alternativos, e, portanto, devem possuir numeração igual. Como os passos associadas a imagens e vídeos são opcionais, os números dos passos deles devem vir entre parênteses, conforme apresentado na Seção 4.2.10.2. Esses passos devem ficar como apresentados na Tabela 14.

O último passo desse caso de uso é um ponto de extensão. Isso ocorre porque, após o sistema exibir as opções de informações para o usuário, um caso de uso mais específico para cada uma das opções deve ser apresentado. Assim, o último passo desse caso de uso deve ser: “Ponto de Extensão: “<tipo_de_midia>”.

Os sumários apresentados no caso de uso são descritos seguindo as restrições sintáticas especificadas nas seções anteriores e descrevem os pontos de variação e variantes presentes no caso de uso “Mostra Documentos”. Conforme apresentado anteriormente, há três pontos de variação, sendo dois deles opcionais, e quatro variantes, sendo duas deles filhas de um mesmo ponto de variação.

Tabela 14 – Caso de uso Mostra Documentos construído

Nome do Caso de Uso	Mostra Documentos		
Caso de Uso Estendido			
Ponto de Extensão			
Categoria de Reuso	Obrigatório		
Restrição de Contexto			
Atores	Visitante (Primário)		
Resumo	O caso de uso contém os passos para exibição dos documentos associados ao ambiente onde o Visitante se encontra		
Pré-condição	O Sistema exibe o mapa do ambiente em que o Visitante se encontra		
Pós-Condição	O Sistema exibe as informações do ambiente em que o Visitante se encontra		
Passos		**Ações do Ator**	**Ações do Sistema**
	1	O Visitante seleciona a opção para visualizar as mídias do ambiente onde ele se encontra.	
[Texto] [Sem Formatação]	**2**		O Sistema exibe a opção "textos sem formatação".
[Texto] [Tags]	**2**		O Sistema exibe a opção "textos na forma de nuvem de *tags*".
[Imagem]	**(3)**		O Sistema exibe a opção "imagens".
[Video]	**(4)**		O Sistema exibe a opção "vídeos".
	5	Ponto de extensão: < tipo_de_midia >	
Pontos de Variação Alternativos e Obrigatórios			
Ponto de Variação	**Variante**	**Restrição de Contexto**	**Passo**
[Texto]: "Qual o formato dos textos que serão exibidos ao Visitante?"	**Sem Formatação**		Passo 02
	Tags		Passo 02
Sumário de Variações Opcionais			
Ponto de Variação	**Restrição de Contexto**		**Passo**
[Imagem]: "Imagens serão exibidas ao Visitante?"	BATERIA_MEDIA OU BATERIA_ALTA		Passo 03
[Video]: "Vídeos serão exibidos ao Visitante?"	BATERIA_ALTA		Passo 04

Fonte: O autor.

4.4.2 Caso de Uso: Play Brickles

Como descrito anteriormente, o caso de uso utilizado para exemplificar a conversão de um caso de uso existente para o CAPLUC, ainda considerando as restrições da linguagem, que

é o foco deste trabalho, foi desenvolvido pelo *Software Engineering Institute* e pela Universidade de *Carnegie Mellon*.

A linha de produtos desenvolvida por essas duas instituições foi denominada Arcade Game Maker (MAKER, 2014) e seu objetivo é desenvolver jogos, cujo propósito é acumular pontos ao passar por obstáculos, que permitem apenas um jogador. O objetivo do jogo descrito no caso de uso, o Play Brickles, é eliminar todos os "tijolos" acertando-os com o disco que é direcionado por uma barra horizontal na parte inferior da tela. Tal meta deve ser atingida antes que o número de vidas acabe, o que ocorre quando o jogador não consegue mover a barra horizontal para direcionar o disco e o disco cai no "piso".

Após conhecer o funcionamento do jogo, é possível entender melhor o que o caso de uso descreve. A Figura 14 contém o caso de uso original, que está escrito em inglês, com o nome dos atores envolvidos, pré-condição, pós-condição e os passos executados pelos atores e as respostas sistema a cada interação.

Figura 14 – Caso de Uso Play Brickles

Actor: GamePlayer or GameInstaller

Preconditions: AGM011: Install game has completed successfully.

Detailed Description

Trigger:

Actor	System Response
Selects PLAY from the menu	Initializes the game and displays the gameboard
Left-click to begin play	Starts game action
Repeat the following for 10 frames plus a bonus throw	
Positions the mouse and left-clicks to send ball down alley	Moves the ball down the alley using a randomly selected algorithm. If collisions result when the ball reaches the pins, moves pins as determined by the physics of the collision.
	Counts number of pins knocked down
Positions the mouse and left-clicks to send ball down alley	Moves the ball down the alley using a randomly selected algorithm. If collisions result when the ball reaches the pins, moves pins as determined by the physics of the collision
	Computes score

Postconditions: Game has been played.

Fonte: Maker (2014).

Os atores responsáveis por interagir com o caso de uso são o "Jogador" ou o "Responsável pela instalação do jogo". Eles são considerados atores primários, pois iniciam a execução do caso de uso. Para facilitar a compreensão será adotado apenas um ator, o "Jogador", com ator

principal. Seguindo as restrições sintáticas definidas na Seção 4.1.7, o nome do ator deverá ser "Jogador (Primário)".

A pré-condição informada pelos autores do caso de uso diz que o jogo deve ter sido instalado com sucesso. A frase, como pode ser percebido na Figura 14, não se encontra na sintaxe definida pela CARNAUbA, pois possui mais de um verbo e precisa de um ator ou sistema para realizar o papel de sujeito da oração. Dessa forma, a frase não pode ser traduzida literalmente, necessitando, assim, de ajustes. Portanto, transformando a frase escrita em linguagem natural para a CARNAUbA, tem-se a seguinte oração: "O Jogador instalou o jogo com sucesso".

A pós-condição, em tradução literal, diz que "O jogo foi jogado". Assim como no caso anterior, a frase não se enquadra na sintaxe estabelecida pela CARNAUbA devido à existência de dois verbos e nenhum complemento. Portanto, alguns ajustes também são necessários para adequar a frase à sintaxe da linguagem definida neste trabalho. Com as alterações, a frase deve ficar como: "O Jogador encerrou o jogo".

Analisando os passos dos casos de uso, tem-se que a primeira ação executada pelos atores é um clique no menu "Play". Conforme pode ser visualizado na Figura 14, a frase se encontra quase totalmente adequada à sintaxe da CARNAUbA, necessitando ajustar apenas o sujeito que está oculto. Assim, após a conversão, a frase ficaria da seguinte maneira: "O Jogador clica no menu 'Play'".

Como resposta do sistema a primeira interação do ator, tem-se, no original, "Inicia o jogo e exibe o tabuleiro". Assim como nas outras situações é necessário realizar alguns ajustes para que a frase se enquadre nas restrições sintáticas da CARNAUbA. Dessa forma, essa oração deve ser convertida em outras duas orações o que acarreta no aumento do número de passos: "O Sistema inicia o jogo" e "O Sistema exibe o tabuleiro".

O procedimento deve ser realizado para todos os outros passos do caso de uso. Como algumas das frases são semelhantes, o processo será detalhado para o caso de existir um condicional em uma frase, situação ainda não explicada. Na maior sequência de ações do caso de uso apresentado na Figura 14, é possível perceber condições. A primeira delas diz: "Se o disco colide com o teto, ele reflete de volta para a área de jogo". Nessa situação, são necessários apenas alguns ajustes. Convertendo para a linguagem, a frase deveria ser escrita da seguinte maneira: "Se o disco colide com o teto, então o disco volta para a área de jogo." As alterações que foram realizadas foram de inserção de palavras (então), além da mudança do sujeito que era um pronome, o que não é permitido pela linguagem e a redução do número de verbos.

Por fim, o caso de uso após o procedimento de conversão deve ficar semelhante ao que é apresentado na Tabela 15.

Tabela 15 – Caso de uso Play Brickles convertido para a CARNAUbA

Elemento do Caso de Uso		**Descrição**	
Nome		Play Brickles	
Caso de Uso Estendido		--	
Ponto de Extensão		--	
Categoria de Reuso		Obrigatório	
Restrição de Contexto		--	
Resumo		Acertar as barras com o disco	
Atores		Jogador	
Pré-condição		O Jogador instalou o jogo com sucesso.	
Pós-condição		O Jogador finalizou o jogo.	
Passo		**Ações do Usuário**	**Ações do Sistema**
	1	O Jogador clica no menu "Play".	
	2	--	O Sistema exibe o tabuleiro.
	3	O Jogador clica no botão esquerdo do mouse.	
	4	--	O Sistema inicia o jogo.
	5	O Jogador clica no botão esquerdo do mouse.	
	6	O Jogador clica nas setas direcionais.	
	7		O Sistema move a barra horizontalmente seguindo o movimento do mouse.
	8		O Sistema verifica a existência da colisão do disco com outro objeto.
	9		Se o disco colide com o teto, então O Disco volta para a área de jogo.
	10		Se o disco colide com a parede, então O Disco volta para a área de jogo.
	11		Se o disco colide com o piso, então O Disco deixa de existir.
	12		Se o número máximo de chances não for alcançado, então O Sistema exibe um novo disco na tela.
	13		Se o número máximo de chances for atingido, então O Sistema exibe uma caixa de diálogo de derrota.
	14		Se o disco colide com um "tijolo", então O Sistema define uma ação pelo tipo de tijolo.
	15		Se o disco colide com o último "tijolo", então O Sistema exibe a caixa de diálogo de vitória.
	16	O Jogador clica no botão da caixa de diálogo.	
	17		O Sistema exibe a tela inicial.

Fonte: O autor.

4.5 Limitações

A linguagem natural controlada foi definida neste trabalho, mas ainda existem limitações na mesma que podem ser superadas em uma nova versão. Algumas dessas limitações são listadas a seguir:

- A descrição de condicionais aninhados (se então) não é possível quando se utiliza a linguagem. Logo, não se pode descrever no caso de uso condições (se não) que só serão executadas quando outra condição (se então) for verdadeira;
- A CARNAUbA só aceita o português, foco deste trabalho. Portanto, não é possível descrever os casos de uso em outras línguas;
- A linguagem não permite a descrição das frases com mais de um verbo. Isso acarretaria em mais expressividade na linguagem, aumentando, assim, a possibilidade de frases com mais informações o que pode melhorar a interpretação.

4.6 Conclusão

Neste capítulo foi apresentada a linguagem natural controlada proposta neste trabalho para auxiliar a geração automática de teste a partir de casos de uso. A linguagem engloba restrições sintáticas para cada elemento do CAPLUC e possui também restrições semânticas para o campo que descreve a interação do ator com o sistema. Também foram apresentados exemplos de uso da linguagem, bem como as limitações da mesma.

Para auxiliar na definição da sintaxe dos elementos do template de caso de uso, foram estudados casos de uso de algumas linhas com o intuito de descobrir qual a melhor sintaxe para cada elemento do caso de uso. Isso foi possível ao se comparar as frases em cada caso de uso.

O processo para realizar a análise semântica foi definido após o estudo da sintaxe do elemento “Passo”, pois após a definição da sintaxe ficou definido quais os tipos de frases que poderiam ser produzidas pela linguagem. Como apresentado na Seção 4.2, em um passo podem ser descritas ações, condições ou repetições.

Apesar das limitações descritas e da necessidade de treinamento, acredita-se que a linguagem possa ser utilizada para descrever os casos de uso, pois após a compreensão da sintaxe, o uso e a geração de testes deve ser facilitado.

5 O MÉTODO E A FERRAMENTA PARA ESTENDER O CHAPTER

Este capítulo trata da definição da extensão do método ChAPTER. Na seção 5.1 são apresentadas as limitações do ChAPTER e que serviram de motivação para o desenvolvimento desta dissertação; na Seção 5.2 é apresentado o método que estende o ChAPTER; na Seção 5.3 é descrita a ferramenta de apoio ao método proposto; a Seção 5.4 contém a arquitetura da ferramenta; na Seção 5.5 são especificadas as limitações do método; e a Seção 5.6 contém as conclusões.

5.1 Limitações do ChAPTER

O ChAPTER possui algumas limitações, como a geração apenas de cenários de teste para LPSSC, e, por isso, foi proposta uma extensão. Essa extensão para o método ChAPTER tem como objetivo gerar casos de teste, bem como a estrutura necessária para sua execução. Essa estrutura é composta por classes que são utilizadas para associar os casos de teste com a linha ou aplicação sob teste.

Para a definição da extensão foram estudadas abordagens de geração de testes a partir de casos de uso. Para complementar o estudo e gerar os testes com o auxílio de técnicas conhecidas foram também pesquisadas as técnicas de teste conhecidas como partição de equivalência e análise do valor limite.

5.2 Funcionamento da extensão proposta

O funcionamento da extensão tem início com a descrição dos casos de uso. Essa descrição deve ser realizada com base nas restrições sintáticas da CARNAUbA especificadas na Seção 4.2. Após efetuar a descrição, os casos de uso devem ser submetidos ao interpretador da linguagem para verificar a estrutura sintática e semântica das frases dos passos do caso de uso.

É importante ressaltar que o template de caso de uso utilizado para a geração dos testes é o template definido por (SANTOS, 2013), o CAPLUC, que foi apresentado na Seção 3.4.7.1. Logo, todas as informações obtidas para a geração dos testes estão de acordo com as definições desse template.

O interpretador da linguagem é composto por duas partes: o analisador sintático e o analisador semântico. O analisador sintático é o responsável pela verificação da estrutura das expressões ou frases dos casos de uso. Ele deve informar se os elementos preenchidos estão de

acordo com as sintaxes estabelecidas pela CARNAUBA e que foram apresentadas no Capítulo 4.

O analisador semântico, por sua vez, é responsável por identificar os termos contidos nas frases dos passos do caso de uso, observando a sua composição. Ele é responsável por identificar o sujeito, o verbo e os possíveis complementos que possam existir nas frases. Uma análise mais detalhada do analisador semântico é feita na Seção 4.3.

O processo de analisar as frases sintática e semanticamente é executado em paralelo, e, assim, é possível verificar a corretude das sintaxes das orações, bem como identificar quais os sujeitos, os verbos e complementos inseridos em cada frase. Isso facilita a análise das frases, diminuindo o processamento das orações e podendo gerar mais facilmente os testes.

Essas informações são repassadas para o ChAPTER e para o gerador de testes simultaneamente, pois o método proposto por (SANTOS, 2013) deve gerar os cenários, enquanto o gerador de testes proposto identifica as entradas e saídas esperadas. No caso das entradas, mais informações, como tipo da entrada (número inteiro, real, texto), podem ser adicionadas o que contribui para uma geração parcial dos testes mais próxima dos testes executáveis.

O gerador de testes, ao receber as informações obtidas pelos analisadores sintáticos e semânticos, procura pelos verbos para interpretá-los como comandos a serem inseridos em uma classe que deve conter os procedimentos de testes a serem executados. Em seguida, ele deve verificar se existem campos a serem preenchidos durante a execução do teste, e, caso existam, deve descobrir quais são.

Após analisar as frases e descobrir quais campos foram extraídos dos passos, o gerador de testes os classifica em campos de entrada e saída baseado no sujeito da frase. Como apresentado na Seção 4.2, se o sujeito for um ator, ele representa um campo de entrada, mas se for o sistema, o campo é de saída.

Em relação aos campos de entrada, é possível, ainda, adicionar mais informações que contribuem para uma melhor geração dos testes. Essas informações estão associadas ao nome do campo a ser exibido no teste, o tipo do campo, limites inferiores e superiores e restrições nos valores a serem gerados. Um novo nome para o campo é importante, pois o nome identificado pelo método pode ser muito genérico e o testador deseja que o nome do campo seja algo mais específico. Para o passo “O Visitante seleciona a opção ‘Texto’”, opção é um termo genérico, e o testador pode colocar um nome mais específico para esse campo, como “menu” ou “item de menu”, entre outros.

O tipo do campo define se o mesmo é um texto (string), um número inteiro, um

número real (float), um valor de localização outdoor (GPS) ou indoor e ainda se é um arquivo. A relevância dessa informação é que valores mais próximos dos reais são gerados pelo método contribuindo ainda mais para a aproximação dos testes da execução.

Os limites inferiores e superiores indicam qual o número mínimo e máximo de caracteres que um texto deve ter, ou o número máximo e mínimo que um inteiro ou real podem assumir. Com base nessas informações, os dados de teste podem ser gerados utilizando as técnicas de partição de categorias e análise do valor limite (MYERS *et al.*, 2011). Dessa forma, para um campo ao qual são especificados os limites inferior e superior são gerados no mínimo quatro valores: i) um valor igual ao limite inferior; ii) um valor maior em uma unidade do limite inferior; iii) um valor igual ao limite superior; e iv) um valor menor em uma unidade ao limite superior. Se não for informado nenhum limite, apenas um valor aleatório será gerado.

O campo que define restrições tem como objetivo informar valores que não possam ser utilizados nas entradas. Essa informação tem relevância quando se deseja que um campo, por exemplo, não aceite o caractere "%", assim, os valores gerados são diferentes do caractere "%".

Enquanto o gerador de testes recebe esses detalhes sobre os campos de entrada, o ChAPTER separa os testes em categorias que vão ser utilizadas para definir os cenários de teste. Esses cenários são repassados para o gerador de testes proposto, conforme apresentado na Figura 15, para que sejam verificados os campos presentes em cada cenário de teste. Após esse procedimento, o gerador de teste deve fazer uma combinação das possibilidades existentes. Para isso, ele verifica quantos valores foram gerados para cada entrada, e, em seguida, ele combina todos os valores de uma entrada com os valores das outras entradas. Supondo que existam duas entradas e para cada entrada sejam gerados quatro valores, cada valor de uma entrada deve ser testado com um valor da outra entrada. Logo, tem-se dezesseis casos de teste gerados parcialmente para uma situação.

Esses valores gerados são inseridos na ferramenta de teste FitNesse (FITNESSE, 201e). A FitNesse é uma ferramenta Web que permite uma fácil alteração dos dados pelos usuários, onde são inseridas tabelas para representar os casos de teste. Pelo fato dela ser uma ferramenta Web, possui páginas, e, por isso, cada teste deve ser inserido em uma página. O ChAPTER gera cenários de teste em páginas na FitNesse e a extensão proposta gera os casos de testes para cada cenário definido pelo ChAPTER.

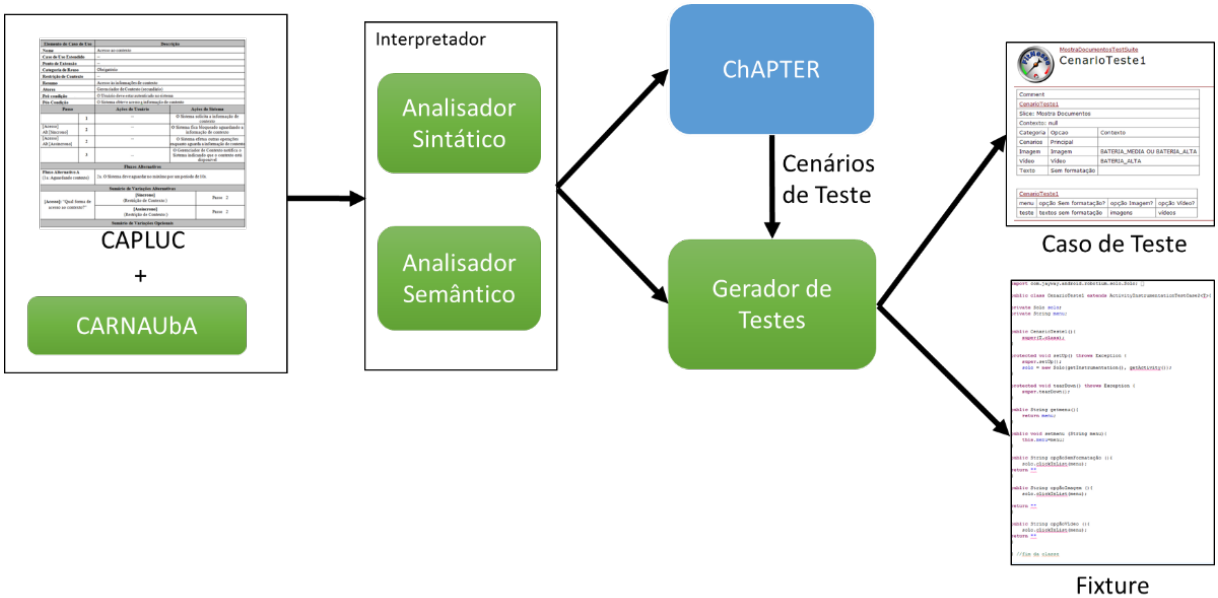
Para utilizar os valores dos testes, é necessária uma classe que faça a ligação entre os valores a serem testados e os sistemas ou componentes sob teste. Essa classe é chamada *Fixture*

e pode ser desenvolvida em várias linguagens, no caso deste trabalho foi adotada a linguagem Java. É nela que estão contidos os procedimentos de teste a serem executados nas partes a serem testadas.

Na *Fixture*, as entradas identificadas na FitNesse são representadas por atributos de classe que precisam dos métodos "get" e "set". As saídas são representadas por métodos que possuem um retorno. Se o campo for um texto, o retorno deve ser uma string. Se for um número, o tipo do retorno tem que ser igual ao dos valores inseridos. Por exemplo, se for um campo onde só sejam apresentados números inteiros, o retorno deve ser desse tipo. Caso seja um número real, o tipo do retorno do método deve real. O mesmo procedimento deve realizado para todos os tipos primitivos do Java.

Conforme observado na Figura 15, o gerador de testes também possui outra saída além dos casos de teste na FitNesse. Ele gera um esboço da *Fixture* seguindo o padrão identificado acima. Todos os campos de entrada são representados por variáveis globais com seus respectivos métodos "*get*" e "*set*". Os campos de saída são definidos como métodos. Os comandos, que representam os procedimentos de teste, associados aos verbos encontrados, devem ser inseridos nos métodos gerados. Um exemplo de uso completo da extensão é apresentado no Capítulo 6.

Figura 15 – Visão Geral da extensão do ChAPTER



Fonte: O autor.

5.3 Ferramenta de Apoio ao Método

(SANTOS, 2013) desenvolveu uma ferramenta, a ToChAPTER, que tem como objetivo a implementação do método ChAPTER. Portanto, assim como o método foi estendido de (SANTOS, 2013), a ferramenta desenvolvida também foi estendida, dessa forma, a ferramenta é denominada ToChAPTER 2.0.

Na ferramenta existente é possível cadastrar linhas de produtos de software sensíveis ao contexto, bem como os seus produtos, além de permitir fazer o download da ferramenta da mesma para que possa ser utilizada pelos testadores em seus projetos específicos e exibir informações sobre a mesma e sobre o seu desenvolvimento. Essas funcionalidades são melhores descritas em (SANTOS, 2013).

Para estender a ferramenta, foram inseridas duas telas associadas a proposta desta dissertação. Uma para tratar da linguagem aqui definida, e a outra que é necessária para a execução do método, pois a função dela é receber as informações extras sobre os campos de entrada. As telas da ferramenta responsáveis por essas partes são apresentadas nas figuras 16 e 17.

Na ferramenta é implementada a extensão proposta neste trabalho, e, por isso, ela é capaz de gerar casos de teste na ferramenta FitNesse e as fixtures necessárias para a execução dos testes, conforme apresentado na Seção 5.2.

5.4 Arquitetura da Ferramenta

Nesta seção é apresentada a arquitetura da ferramenta desenvolvida por (SANTOS, 2013) e estendida neste trabalho. Dessa forma, são apresentados os diagramas de pacote, de classe e de sequência associados ao método proposto.

5.4.1 Diagrama de Pacotes

Nesta subseção é apresentado o diagrama de pacotes da ferramenta considerando a extensão proposta. A Figura 18 exibe o diagrama com os pacotes e notas sobre o papel de cada um. No ChAPTER, existiam os pacotes de controladores, persistência e o ChAPTER, como apresentado em (SANTOS, 2013). Esses pacotes também são explicados nesta subseção.

No diagrama é possível perceber que a ferramenta é composta de seis módulos. O primeiro deles, o "*Controllers*", representa os controladores cuja função é receber e prover

Figura 16 – Tela da ferramenta que exibe informações da CARNAUbA

Fonte: O autor.

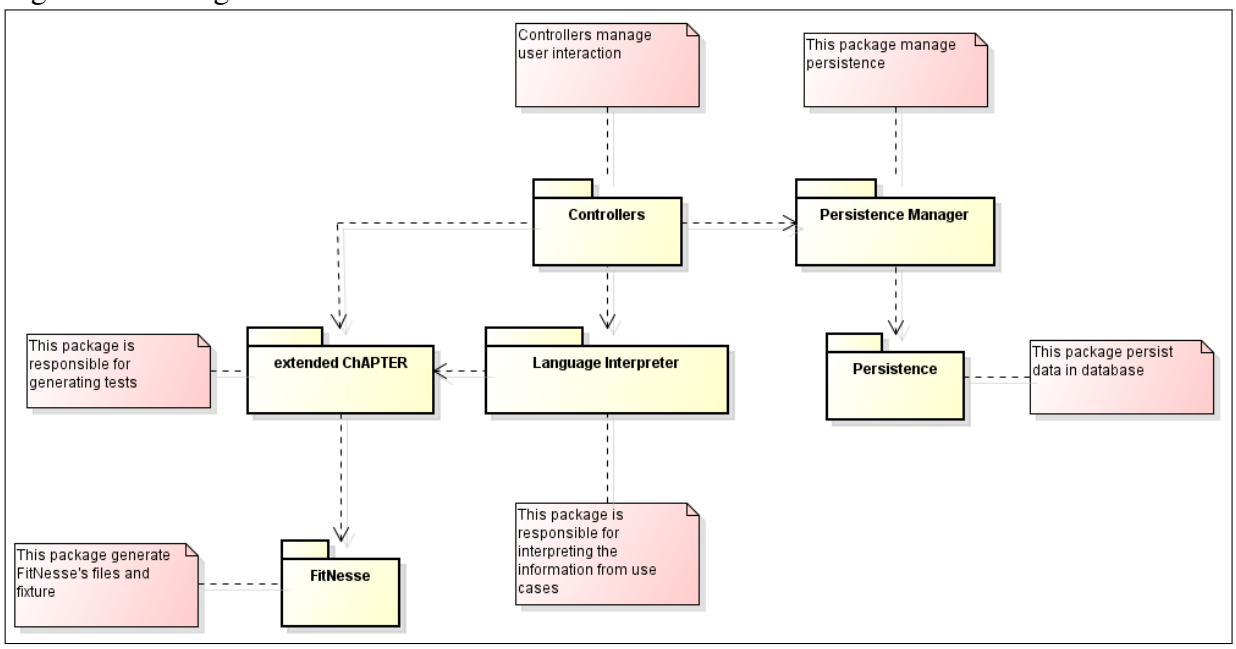
Figura 17 – Inserção de detalhes para as entradas

Fonte: O autor.

informações aos usuários. Os controladores se comunicam com três pacotes. Um deles é o "*Persistence Manager*", cuja função é gerenciar as informações a serem armazenadas no banco de dados. Porém, ele ainda não armazena os dados, sendo necessário para isso outro pacote, o "*Persistence*".

Os controladores ainda se comunicam com o "*Language Interpreter*" que recebe as informações sobre os casos de uso a serem testados. Esse pacote interpreta as informações contidas neles e envia essas informações para outro pacote, o "extended ChAPTER". Esse último pacote também se comunica com os controladores recebendo desses as instruções sobre os casos

Figura 18 – Diagrama de Pacotes



Fonte: O autor.

de uso a serem testados, bem como outras informações necessárias para o teste.

Após isso, os testes são gerados pelo “extended ChAPTER” e são repassados para o módulo “FitNesse” cujo objetivo é gerar os arquivos de teste, bem como a Fixture necessária para a execução do teste a partir das informações enviadas pelo “extended ChAPTER”.

5.5 Limitações

Como limitações encontradas no método, é importante citar as seguintes, destacando que as mesmas podem ser solucionadas em novas versões do método:

- Geração de passos nos métodos da Fixture que representam apenas a execução dos passos do caso de uso até o momento em que as saídas esperadas foram encontradas. Por exemplo, supondo que os passos 3 e 6 de um caso de uso possuam saídas, os comandos inseridos nos procedimentos da fixture para cada um desses passos devem ser diferentes, pois entre o passo 3 e o 6 podem existir passos a serem executados que não devem estar presentes no método da fixture referente ao passo 3, pois da forma como a extensão está atualmente implementada, o método referente ao passo 3 contém ações a serem executadas até o passo 6;
- Especificação mais detalhada das categorias a serem utilizadas para geração dos valores das entradas. É importante que o testador possa identificar mais detalhes sobre as entradas, como várias categorias para a geração dos testes;

- Geração de valores específicos para os fluxos alternativos. Atualmente, o método utiliza os valores gerados para o fluxo principal, mesmo que seja um fluxo alternativo. Portanto, é importante gerar valores específicos para os fluxos alternativos considerando as condições para que um fluxo alternativo seja chamado para executar;
- Simulação de contexto durante a execução dos testes. O método ainda não permite a simulação de contexto necessária para a execução dos testes, o qual estava fora do escopo deste trabalho. O contexto na extensão proposta deve estar igual ao presente no elemento "Restrições de Contexto";
- Executar um ponto de extensão quando encontrado. A extensão não é capaz de gerar testes sequenciais quando encontra um ponto de extensão no caso de uso.

5.6 Conclusão

Neste capítulo foram apresentadas a extensão para o método ChAPTER juntamente com a extensão para sua ferramenta, assim como as limitações propostas para o método. O exemplo de uso da extensão proposta e da ferramenta é apresentado no Capítulo 6.

A extensão proposta difere dos demais métodos no que diz respeito a geração da estrutura para a execução de teste em linha de produtos de software sensíveis ao contexto. Essa estrutura é composta de casos de teste e de classes necessárias para a execução dos testes.

Para definir a extensão foram executadas duas etapas. A primeira foi o estudo de outras abordagens de geração parcial de testes a partir de casos de uso. A segunda etapa foi a definição da sintaxe dos passos do caso de uso, pois após isso é que o método poderia ser definido.

Apesar das limitações descritas, acredita-se que o método na versão atual facilita a geração de parte dos casos de teste e das classes necessárias para a automação da execução dos testes.

6 APLICAÇÃO DA EXTENSÃO PROPOSTA

Neste capítulo é apresentada uma aplicação da extensão proposta e sua respectiva implementação na ferramenta ToChAPTER em dois estudos de caso. Essa aplicação consiste na demonstração do uso da ferramenta e do método proposto desde a modelagem dos casos de uso até a geração parcial dos casos de testes.

Este capítulo está dividido da seguinte maneira: na Seção 6.1 são apresentadas as linhas de produtos utilizadas para a avaliação preliminar; na Seção 6.2 é apresentada a aplicação da extensão na linha Mobiline; a Seção 6.3 aborda a aplicação da extensão na linha Arcade Game Maker; e a Seção 6.4 apresenta as conclusões do capítulo.

6.1 Mobiline e Arcade Game Maker

Neste capítulo são descritas duas aplicações para o método que estende o ChAPTER, mas para isso é necessário conhecer os casos de uso utilizados e as linhas a qual pertencem. A primeira linha utilizada é a linha Mobiline e a outra linha é a *Arcade Game Maker*. Esta seção é dedicada a apresentação dessas linhas.

As duas linhas foram criadas na academia, sendo a Mobiline (MOBILINE, 2014) criada em uma parceria entre a Universidade Federal do Ceará e a Universidade Federal do Rio de Janeiro enquanto que a *Arcade Game Maker* (MAKER, 2014) foi desenvolvida em uma parceria entre a Universidade de *Carnegie Mellon* e o *Software Engineering Institute* (SEI).

A linha Mobiline, como apresentada no Capítulo 2, é uma linha de produtos de software sensível ao contexto, cujo objetivo é desenvolver guias de visitas móveis e sensíveis ao contexto. O objetivo das aplicações desenvolvidas a partir dessa linha é guiar os visitantes dentro de lugares como um museu ou um laboratório de pesquisa. Dessa forma, durante a visita podem ser apresentadas informações como textos, imagens, vídeos e pessoas que trabalham no ambiente que está sendo visitado. Um exemplo de aplicação desenvolvida a partir desta linha é o GREat Tour que foi mencionado no Capítulo 2.

A outra linha, *Arcade Game Maker*, como mencionado no Capítulo 4, é uma linha de produtos de software para o desenvolvimento de jogos de arcade. Essa linha tem como alguns de seus produtos o jogo de boliche e o brickles, que foram descritos no Capítulo 4.

6.2 Aplicação da Extensão na LPSSC Mobiline

O primeiro passo para poder apresentar a aplicação da extensão proposta neste trabalho é conhecer o caso de uso, bem como o comportamento dos sistemas sob teste que ele descreve. O caso de uso escolhido é um caso de uso da linha e não de um produto gerado pela linha, pois mostra a geração parcial dos testes para uma linha, com os possíveis cenários para a mesma. O caso de uso selecionado da linha Mobiline foi o "Mostra Textos" que tem como objetivo descrever a interação do ator "Visitante" com o sistema para que esse exiba textos associados a um determinado ambiente. O caso de uso é apresentado na Tabela 16.

O caso de uso "Mostra Textos", conforme descrição contida na Tabela 16, estende o caso de uso "Mostra Documentos" apresentado no Capítulo 4. Ele é obrigatório e possui restrições de contexto associadas a ele, que são BATERIA_BAIXA ou BATERIA_MEDIA ou BATERIA_ALTA, e dois atores interagem nesse caso de uso, sendo o "Visitante" como ator primário e o "Servidor" como ator secundário. A pré-condição para a execução desse caso de uso é o fato do "Visitante" ter selecionado a opção para visualizar os textos associados a um determinado ambiente. Como pós-condição, tem-se que o sistema deve exibir os textos associados ao ambiente em que o visitante se encontra.

O caso de uso é disparado com a ação do "Visitante" de selecionar a opção para visualizar os textos do ambiente. Como resposta a essa ação, o sistema solicita para o servidor os textos. O "Servidor", que é um ator secundário, como apresentado anteriormente, retorna os textos solicitados. Por fim, o sistema pode exibir os textos de duas formas. A primeira é o texto simples, sem formatação. A segunda é a exibição em forma de nuvem de *tags*. Esses dois últimos passos são alternativos e estão associados ao mesmo ponto de variação, porém com variantes diferentes. O ponto de variação que esses passos estão associados é o "Texto" e como variantes tem-se "Sem Formatação" para o passo que exibe o texto simples e "*Tags*" para o passo que exibe a nuvem de *tags*. O sumário dos pontos de variação alternativos e obrigatórios resumem as informações acima.

Conhecido o caso de uso, é possível especificá-lo na ToChAPTER. Para isso, é necessário que a linha a qual o caso e uso pertence esteja cadastrada. Nesse caso, a linha cadastrada é a linha Mobiline, conforme apresentado na Figura 19. Uma linha pode ser editada, removida ou ter os seus casos de uso listados. O usuário deve clicar para visualizar os casos de uso existentes, e, assim, cadastrar um novo caso de uso. No cadastro de um caso de uso, é necessário inserir as informações básicas do caso de uso.

Tabela 16 – Caso de Uso Mostra Texto

Nome do Caso de Uso		Mostra Texto	
Caso de Uso Estendido		Mostra Documentos	
Ponto de Extensão		tipo_de_midia	
Categoria de Reuso		Obrigatório	
Restrição de Contexto		BATERIA_BAIXA OU BATERIA_MEDIA OU BATERIA_ALTA	
Atores		Visitante (Primário), Servidor (Secundário)	
Resumo		O Visitante visualiza os textos associados ao ambiente onde ele se encontra	
Pré-condição		O Visitante selecionou a opção para visualizar as mídias do ambiente onde ele se encontra	
Pós-Condição		O Sistema exibe os textos associados ao ambiente	
Passos		**Ações do Ator**	**Ações do Sistema**
	1	O Visitante seleciona a opção para visualizar os textos do ambiente em que se encontra.	
	2		O Sistema solicita os textos associados à localização atual do Visitante ao Servidor.
	3	O Servidor retorna os textos associados à localização atual do Visitante.	
[Texto] [Sem Formatação]	**4**		O Sistema exibe o texto selecionado pelo Visitante sem Formatação.
[Texto] [Tags]	**4**		O Sistema exibe o texto selecionado pelo Visitante na forma de nuvem de tags.
Pontos de Variação Alternativos e Obrigatórios			
Ponto de Variação	**Variante**	**Restrição de Contexto**	**Passo**
[Texto]: "Qual o formato dos textos que serão exibidos ao Visitante?"	**Sem Formatação**		Passo 04
	Tags		Passo 04

Fonte: O autor.

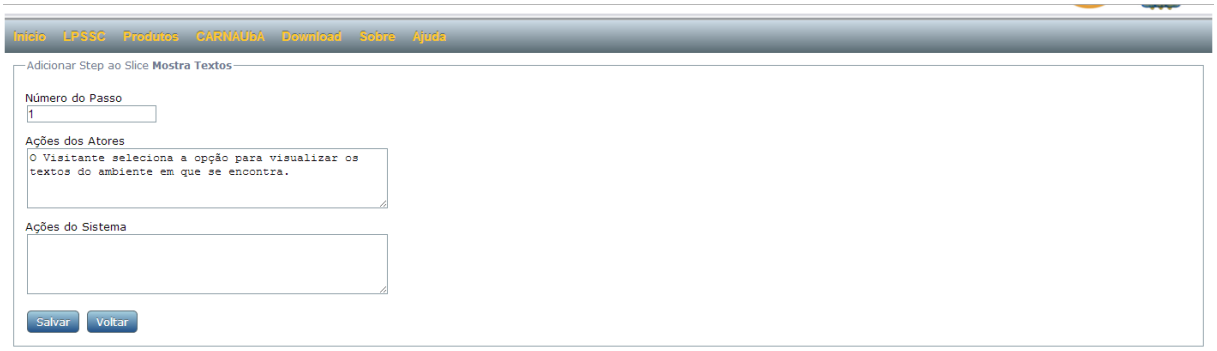
Em seguida, o caso de uso tem que ser cadastrado por completo, terminando com a inserção dos passos do mesmo. Vale ressaltar que um passo pode ser uma ação de um ator ou uma resposta do sistema. É importante ressaltar que as frases que descrevem os passos devem seguir a sintaxe descrita no Capítulo 4. Após o cadastro de cada passo do caso de uso, incluindo os passos alternativos, que possuem a mesma numeração, o usuário da ToChAPTER clica em concluir e o caso de uso está pronto para ser analisado para permitir a geração parcial de testes. A tela de cadastro de um passo no caso de uso é apresentada na Figura 20. É possível também definir pontos de variação e variante para os passos de um caso de uso, como explicado em (SANTOS, 2013).

Para gerar testes a partir de um caso de uso, o usuário deve selecionar essa opção na ferramenta. Em seguida, a ferramenta envia para o interpretador da linguagem o caso de uso para

Figura 19 – Lista de Linhas cadastradas

Fonte: O autor.

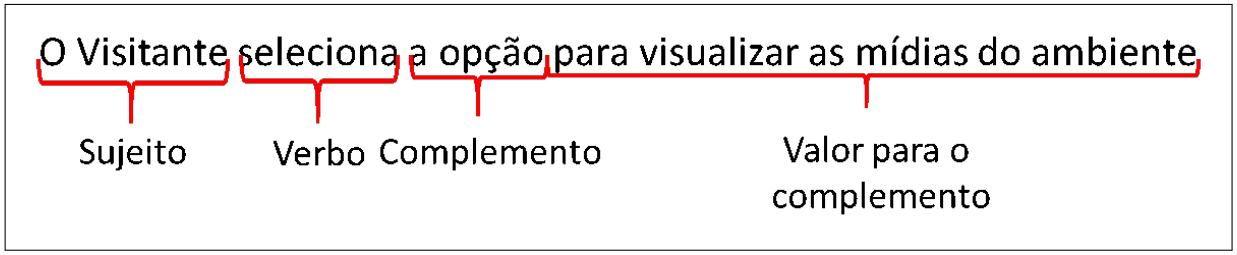
Figura 20 – Definição de um passo para o caso de uso Mostra Textos



Fonte: O autor.

ser analisado. A análise ocorre passo a passo conforme definido no Capítulo 4 e na Figura 21.

Figura 21 – Extração das informações do passo 1 do caso de uso Mostra Textos



Fonte: O autor.

O primeiro passo executado pelo interpretador é a verificação da existência de um verbo permitido pela linguagem. A lista completa pode ser encontrada no Apêndice A. Caso seja confirmada a presença de um verbo, o interpretador procura o sujeito da oração. O primeiro termo da frase deve ser sempre um artigo, e, portanto, o sujeito pode ser encontrado entre esse termo e o verbo.

A execução do interpretador continua com o intuito de identificar possíveis complementos para o verbo, mas para isso, precisa primeiro verificar dentro das regras associadas ao verbo a transitividade que trata sobre a existência ou não de complementos. Caso seja possível identificar um complemento, o interpretador analisa o conteúdo da frase que se encontra logo após o verbo.

O primeiro termo após o verbo também deve ser um artigo e o termo seguinte deve ser um complemento, que pode ser um campo de entrada ou saída que varia conforme o sujeito. O interpretador ainda verifica se a palavra após o primeiro complemento é uma preposição. Caso seja, ele identifica que o complemento é composto por três palavras e procura por mais complementos ou por valores que possam ser utilizados pelo campo na hora do teste. É importante ressaltar que a CARNAUbA não permite a existência de um complemento com mais de três palavras, o que é uma limitação dessa versão.

Caso não seja uma preposição, o interpretador verifica se existe outro complemento na frase. Essa verificação ocorre pela identificação de outros caracteres que sejam responsáveis pela indicação da presença de outro termo, como o caractere "," e a conjunção aditiva "e". Se houver um desses dois caracteres, o interpretador identifica os outros complementos da mesma forma que o primeiro complemento.

Ele deve procurar um artigo e o segundo termo é o núcleo do complemento. Se houver uma preposição após esse complemento, ele é composto por três palavras. Esse procedimento é executado até se encerrar a frase. É importante destacar que, no caso das saídas, elas são associadas aos pontos de variação e variantes existentes nos passos a que elas pertencem, o que contribui para identificar quais campos pertencem a um passo no momento da geração dos testes.

Se não houver outros complementos, o interpretador identifica que o restante da frase é um termo que pode ser utilizado como valor para o teste. Esse valor está associado ao campo identificado pelo interpretador da linguagem. Por exemplo, a expressão "associados à localização atual do visitante ao Servidor" que complementa o termo "textos" no passo 2 do caso de uso "Mostra Textos" deve ser utilizada como uma saída esperada do teste.

Ao final da interpretação da frase, os complementos são classificados em campos de entrada ou saída para que aqueles que representam entradas para o teste sejam informados para o usuário poder inserir mais dados sobre os mesmos. Essa etapa é apresentada na Figura 22. Dentre as opções que o usuário possui, ele pode informar um outro nome para identificar o campo no momento da geração do teste, bem como o tipo desse campo, os limites associados a

ele e algum caractere que não possa ser utilizado durante o teste. No caso da entrada “opção”, foi definido o nome “menu” para ser utilizado no teste.

No caso de uso “Mostra Textos” apenas um campo de entrada é identificado, que é o valor “opção”. Isso porque o verbo, quando associado a um passo de um ator, indica que não pode ter um complemento, conforme visualizado no Apêndice A. Um possível complemento para o campo “opção” é “para visualizar os textos do ambiente em que ele se encontra”, porém como é um campo de entrada esse valor não deve ser utilizado.

No caso das saídas, tem-se duas saídas com o nome “texto”, porém associadas a variantes diferentes. Uma delas está associada a variante “Sem Formatação” e a outra está associada a variante “*Tags*”. Assim como a entrada identificada anteriormente, existem valores que podem ser utilizados para descrever saídas esperadas para esses campos durantes os testes. No caso da saída “texto” associada a variante “Sem Formatação”, o valor esperado para a saída é “selecionado pelo Visitante Sem Formatação”. No caso da outra saída, o valor esperado é “selecionado pelo Visitante na forma de nuvem de *tags*”.

Figura 22 – Tela para inserir mais informações para uma entrada

Fonte: O autor.

Em seguida, essas informações são repassadas para o gerador de testes que as armazena para, posteriormente, gerar valores mais adequados para as entradas identificadas. Em paralelo, o ChAPTER espera que o usuário defina mais categorias a serem utilizadas no teste. Se nenhuma categoria for definida, o ChAPTER gera cenários de teste com base nas categorias que ele próprio identificou, conforme apresentado em (SANTOS, 2013).

O ChAPTER, quando analisa esse caso de uso, consegue identificar dois cenários de teste. O primeiro deles diz que os passos 1, 2, 3 e 4, sendo o passo 4 o que exibe textos sem formatação, são executados, conforme observado na Tabela 17. O segundo diz que os passos 1, 2, 3 e 4, sendo, nesse caso, o 4 associado à variante “*Tags*”, como apresentado na Tabela 18. É importante salientar que os cenários gerados devem seguir as restrições de contexto apresentadas

no caso de uso.

Após a definição dos cenários de teste, esses são repassados para a extensão proposta neste trabalho. A extensão é encarregada de analisar cada cenário e os passos que estão associados aos cenários para poder identificar no teste quais entradas e saídas devem ser geradas. Após essa identificação, os valores para as entradas são gerados com base nas informações definidas pelo usuário na etapa apresentada na Figura 22.

Tabela 17 – Cenário de Teste 1 para o caso de uso Mostra Textos

Cenário de Teste 1	
Contexto de execução	**BATERIA_BAIXA OU BATERIA_MEDIA OU BATERIA_ALTA**
Número do passo	**Passo**
1	O Visitante seleciona a opção para visualizar os textos do ambiente em que se encontra.
2	O Sistema solicita os textos associados à localização atual do Visitante ao Servidor.
3	O Servidor retorna os textos associados à localização atual do Visitante.
4	O Sistema exibe o texto selecionado pelo Visitante sem Formatação.

Fonte: O autor.

No caso de uso “Mostra Textos” é possível identificar os campos “opção” e “texto”. No primeiro passo desse caso de uso, que diz que o visitante seleciona a opção para visualizar os textos do ambiente em que se encontra, ele verifica que existe o campo “opção” e é possível encontrar na frase a palavra “textos” que contém o valor “texto”. Porém, essa palavra está dentro do trecho que pertence ao valor para o campo, além de representar uma saída e não uma entrada e a palavra não ser igual. Portanto, nesse momento, esse campo não é inserido no teste.

No passo 2, que diz que “o Sistema solicita ao servidor os textos associados a localização atual do Visitante”, os campos não são verificados, pois o verbo não exige um complemento segundo as definições inseridas para o mesmo. No passo 3, nenhum campo será extraído, pois assim como no passo 2, o verbo não exige complementos.

No caso do passo 4, existem duas opções de execução. A primeira opção diz respeito

a exibição de textos sem formatação. A entrada “opção” não deve ser inserida no teste por dois motivos. O primeiro deles é por já existir no teste e o segundo é porque é um campo de entrada e o passo analisado é uma ação do sistema que diz que os campos extraídos devem ser saídas para o teste. A saída “texto” com variante igual a “Sem Formatação” é encontrada no passo, mas para ser adicionada no teste, o método verifica o ponto de extensão e a variante associados ao campo. Caso seja igual ao ponto de variação e a variante do passo, o campo é adicionado. Nesse caso, a saída “texto” procurada possui o mesmo ponto de variação e a mesma variante que o passo analisado, conforme apresentado na Tabela 16. No caso da outra saída “texto”, ela é encontrada, mas como possui a variante diferente da variante do passo analisado, ela não é adicionada ao teste. O procedimento é executado no outro passo de número 4 quando ele estiver presente no cenário, como no caso do cenário de teste 2 apresentado na Tabela 18.

Tabela 18 – Cenário de Teste 2 para o caso de uso Mostra Textos

Cenário de Teste 2	
Contexto de execução	**BATERIA_BAIXA OU BATERIA_MEDIA OU BATERIA_ALTA**
Número do passo	**Passo**
1	O Visitante seleciona a opção para visualizar os textos do ambiente em que se encontra.
2	O Sistema solicita os textos associados à localização atual do Visitante ao Servidor.
3	O Servidor retorna os textos associados à localização atual do Visitante.
4	O Sistema exibe o texto selecionado pelo Visitante na forma de nuvem de tags.

Fonte: O autor.

Após a identificação dos campos presentes no cenário de teste, os valores para a entrada “opção” são gerados. Foi definido que a entrada “opção” deveria ter um outro nome para facilitar a identificação na hora do teste, e, por isso, foi atribuído o nome “menu”. Ela deve ser do tipo “string”, mas como ela representa um menu, não possui limites, pois o nome botão não deve mudar. Ela também não possui restrições de caracteres. Dentro dessas condições, o

método gerou um valor aleatório baseado na tabela ASCII. O valor definido para a entrada pode ser observado na Figura 23. Nesse caso, o valor extraído da frase não será utilizado.

Para compreender o teste apresentado na Figura 23, é importante conhecer um pouco a ferramenta utilizada na especificação dos testes. Ela é denominada FitNesse e permite a inserção de valores de entrada e saída para os testes em forma de tabelas. Na segunda linha da tabela são especificadas as entradas e saídas, essas diferenciadas pelo caractere "?". Feito isso, a FitNesse deve executar uma classe, denominada fixture, que é responsável por verificar o comportamento do sistema sob teste e retornar o resultado para a FitNesse para que a mesma informe para o testador se o teste passou ou não. A fixture deve conter os passos a serem executados na aplicação, como o preenchimento de campos e clique em botões. Para preencher os campos durante a execução do teste, ela deve receber os valores informados na tabela presente na FitNesse.

Figura 23 – Teste gerado na FitNesse

MostraTextosTestSuite

CenarioTeste1

Comment		
CenarioTeste1		
Slice: Mostra Textos		
Contexto: BATERIA_BAIXA OU BATERIA_MEDIA OU BATERIA_ALTA		
Categoria	Opcao	Contexto
Cenarios	Principal	
Texto	Sem formatação	

CenarioTeste1	
menu	texto Sem formatação?
VDfR	selecionado pelo Visitante sem Formatação

Fonte: O autor.

O método proposto, após gerar os valores para as entradas e as saídas do teste, começa a estruturar a tabela que representa o teste na FitNesse. Como apresentado anteriormente, na segunda linha da tabela devem ser preenchidos todos os campos, entradas e saídas. É importante ressaltar que a identificação das saídas é feita por uma "?" ao final do nome. As saídas que possuem uma associação com uma variante têm adicionado ao seu nome o nome da sua variante. Em seguida, são preenchidos os valores para a entrada. Por fim, são preenchidos os valores para as saídas. Como elas devem possuir apenas um valor, esse valor é repetido para todos os casos

de teste.

Depois de preenchida a tabela, a fixture é gerada. São definidos os pacotes a serem importados para serem utilizados na fixture. Esses pacotes são da biblioteca da ferramenta de teste Robotium (ROBOTIUM, 2014), que permite a execução dos testes e interage com a interface dos sistemas sob teste, e de uma classe a ser estendida, pois segue o padrão de teste em Android (ANDROID, 2014). A classe deve ser então iniciada. Nesse caso, o nome dela é associado ao cenário a que faz parte, que pode ser "CenarioTeste1", para o caso de ser utilizado o primeiro cenário de teste citado anteriormente. Ela deve estender a classe "ActivityInstrumentationTestCase2" e deve identificar a classe a ser testada. Como nesse processo não foi definida nenhuma classe como sendo a classe sob teste, ela não será informada.

Na fixture deve existir pelo menos um atributo de classe obrigatório, além da entrada. Esse atributo é um objeto do tipo "Solo" que é responsável pela execução dos testes e pela interação com as telas do sistema que será testado. Vale ressaltar que essa fixture gerada é um esboço, pois esse caso de uso é da linha. Quando os produtos forem desenvolvidos, eles podem reutilizar essa fixture.

Em seguida, são inseridas as entradas. Nesse caso, como existe apenas a entrada "opção" e ela, como definido anteriormente, é uma string, a mesma é adicionada como um atributo de classe pertencente a esse tipo. O construtor do método pode não receber nenhum parâmetro, mas deve ter uma chamada para superclasse a que pertence à fixture passando como parâmetro a classe sob teste.

Devem ser definidos também os métodos "*setUp*" e "*tearDown*". O primeiro é para ser executado antes do início dos testes e o segundo é para ser executado após a execução dos testes. No "*setUp*", deve existir uma chamada para o método "*setUp*" da superclasse e o atributo de classe "solo" deve ser instanciado. No "*tearDown*", é inserida uma chamada para o método "*tearDown*" da superclasse.

Por fim, são inseridos os métodos que representam as saídas na FitNesse. Para o caso de uso descrito, é inserido o método "texto", tanto no primeiro cenário de teste quanto no segundo, seguido do nome da variante associada ao passo a que ele pertence. Em cada método são inseridos comandos associados aos verbos utilizados nas frases para descrever os passos de um caso de uso. Caso não existam comandos associados, um passo não é descrito na fixture. A fixture gerada para o cenário de teste 1 é apresentada na Figura 24.

Figura 24 – Fixture gerada para o caso de uso Mostra Textos

```
import com.jayway.android.robotium.solo.Solo;
import android.test.ActivityInstrumentationTestCase2;

 public class CenarioTeste1 extends ActivityInstrumentationTestCase2<T>{

    private Solo solo;
    private String menu;

    public CenarioTeste1(){
        super(T.class);
    }

    protected void setUp() throws Exception {
        super.setUp();
        solo = new Solo(getInstrumentation(), getActivity());
    }

    protected void tearDown() throws Exception {
        super.tearDown();
    }

    public String getMenu() {
        return menu;
    }

    public void setMenu(String menu) {
        this.menu = menu;
    }

    public String textoSemFormatação(){
        solo.clickInList(menu);
        return ""
    }

} //fim da classe
```

Fonte: O autor.

6.3 Aplicação da Extensão na LPS Arcade Game Maker

O caso de uso escolhido dessa linha foi o caso de uso “Play Bowling” que descreve os passos de um jogo de boliche. Esse caso de uso é de um produto da linha e ao final da geração dos testes, os testes precisarão ser pouco alterados para poderem ser utilizados.

Como esse caso de uso já existe, é importante conhecê-lo primeiro antes da utilização do método. Além disso, para ser utilizado pela extensão proposta ele deve ser convertido para o padrão sintático definido pela linguagem CARNAUbA. Os passos executados para essa conversão são apresentados no Capítulo 4, e serão revisitados nesta seção.

O caso de uso original é apresentado na Figura 25. Nele é possível identificar que os atores podem ser o “Jogador” ou o “Responsável pela instalação”. Para facilitar a compreensão será utilizado o “Jogador” como único ator do caso de uso. O caso de uso possui como pré-condição a instalação completa e com sucesso do jogo desejado, nesse caso, é o boliche. Ele possui como pós-condição “O Jogo foi jogado”, o que podemos transformar em “O Jogador executou o jogo”.

Figura 25 – Caso de uso “Play Bowling”

Play Bowling

Use Case ID: AGM008

Use Case Level: System end-to-end

Scenario

Actor: GamePlayer or GameInstaller

Preconditions: AGM011: Install game has completed successfully.

Detailed Description

Trigger:

Actor	System Response
Selects PLAY from the menu	Initializes the game and displays the gameboard
Left-click to begin play	Starts game action
Repeat the following for 10 frames plus a bonus throw	
Positions the mouse and left-clicks to send ball down alley	Moves the ball down the alley using a randomly selected algorithm. If collisions result when the ball reaches the pins, moves pins as determined by the physics of the collision.
	Counts number of pins knocked down
Positions the mouse and left-clicks to send ball down alley	Moves the ball down the alley using a randomly selected algorithm. If collisions result when the ball reaches the pins, moves pins as determined by the physics of the collision
	Computes score

Postconditions: Game has been played.

Fonte: O autor.

Em relação aos passos, eles são iniciados com a seleção do menu “Play”. Portanto, na CARNAUbA ele deve ficar como “O Jogador clica no menu Play”. O sistema responde com a

inicialização do jogo e a exibição do ambiente do jogo. Como apresentado no Capítulo 4, essa frase deve ser convertida em duas na CARNAUbA. A primeira delas diz que “O Sistema inicia o jogo”. A segunda fica “O Sistema exibe o tabuleiro”.

Para jogar, o jogador deve clicar com o botão esquerdo do mouse e o sistema responde a essa ação com o início da ação do jogo. Assim, o sistema já se encontra pronto exibindo a tela para jogar. As frases para essas ações ficariam assim quando convertidas para a CARNAUbA: “O Jogador clica no botão esquerdo do mouse” e “O Sistema inicia o jogo”.

Em seguida, o passo encontrado indica repetição de outros passos. No caso de uso original, é apenas informado que os passos seguintes são repetidos dez vezes. Essa frase deve ser convertida para o padrão da CARNAUbA que exige a identificação dos passos a serem repetidos. Nesse momento ainda não é possível dizer quais os números dos passos, portanto, essa instrução será convertida por último.

A próxima ação do ator é posicionar o mouse para definir a direção que a bola de boliche deve seguir para derrubar os pinos. Assim, a frase definida para essa ação fica “O Jogador move o mouse para arremessar a bola”. O sistema, como resposta a essa ação do ator, move a bola. Ele também verifica se houve alguma colisão da bola com os pinos e caso existam colisões, o sistema derruba os pinos. Em seguida, ele conta o número de pinos que foram derrubados. A frase da ação do sistema deve ser dividida em outras três frases. A primeira delas diz que “O Sistema move a bola”. A segunda é uma condição que verifica a colisão da bola: “Se a bola atinge pelo menos um pino, então o sistema move os pinos”. A terceira representa a contabilização dos pinos derrubados: “O Sistema contabiliza o número de pinos derrubados”.

O jogador repete o movimento de arrastar o mouse para derrubar os pinos. O sistema executa o mesmo procedimento de mover a bola e verificar se houve alguma colisão. Há uma pequena mudança no final, pois o sistema deve computar a pontuação final do jogador após mais um lance. A única frase diferente dos procedimentos anteriores foi convertida para a seguinte oração: “O Sistema contabiliza a pontuação final do Jogador”. Assim, o caso de uso convertido para a CARNAUbA fica como apresentado na Tabela 19.

Como mencionado anteriormente, o passo que descreve a repetição deve ser convertido por último. Assim, é possível saber quantos passos existem ao todo no caso de uso e quais devem ser repetidos. Como pode ser observado na Tabela 19, o caso de uso possui 13 passos, sendo um deles o passo que trata da repetição. Os passos a serem repetidos são os passos 6 a 13, que indicam o processo de arremessar a bola na pista e a contagem dos pontos do jogador.

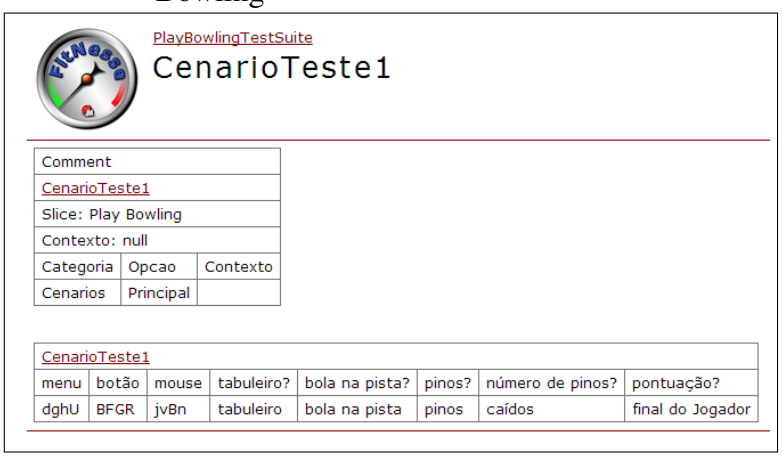
Após a conversão do caso de uso, ele deve ser adicionado na ToChAPTER. Mas, para isso a linha deve ser cadastrada. Em seguida, o testador deve clicar no menu que indica a exibição dos casos de uso da linha. O caso de uso também tem que ser cadastrado e para isso o procedimento executado é igual ao apresentado na Seção 6.2. Após inserir as informações, os passos devem ser cadastrados. Nesse caso de uso não há nenhum ponto de variação e variante associados aos passos.

Quando o testador clicar no menu indicado para iniciar o processo de geração de testes, a ferramenta vai procurar pelas entradas e saídas esperadas. Nesse caso, as entradas identificadas pelo método são “menu”, “botão” e “mouse”, pois as mesmas estão em frases que os verbos exigem complementos. Logo, o testador pode informar mais detalhes sobre esses campos. Porém, nesse caso, nenhuma informação foi adicionada, pois eles apresentam nomes que indicam claramente a sua função. Além disso, as entradas “menu” e “botão” possuem valores estáticos e a entrada “mouse” não deve possuir restrições, portanto, nenhuma informação foi adicionada e só foi gerado um valor para cada entrada, conforme observado na Figura 26.

As saídas identificadas são “tabuleiro”, “bola na pista”, “pinos”, “número de pinos” e “pontuação”. Esses campos foram identificados, pois nas frases em que foram encontrados, os verbos exigem complementos, incluindo nos passos que haviam condições.

O próximo passo a ser executado é a definição de categorias que possam contribuir para a geração de cenários de teste, mas como mencionado na Seção 6.2, essa etapa pertence ao método ChAPTER que é definido em (SANTOS, 2013). Em seguida, os cenários de teste são enviados para a parte responsável pela geração dos testes. O gerador de testes é encarregado de verificar a presença dos campos nos passos e assim gerar valores para serem utilizados no teste.

Figura 26 – Caso de teste na FitNesse para o caso de uso “Play Bowling”



Fonte: O autor.

Tabela 19 – Caso de uso “Play Bowling” convertido para a CARNAUbA

Nome		Play Bowling	
Caso de Uso Estendido		--	
Ponto de Extensão		--	
Categoria de Reuso		Obrigatório	
Restrição de Contexto		--	
Resumo		O objetivo é fazer a maior pontuação derrubando os pinos	
Atores		Jogador	
Pré-condição		O Jogador instalou o jogo com sucesso	
Pós-Condição		O Jogador executou o jogo	
Passo		**Ações do Usuário**	**Ações do Sistema**
	1	O Jogador clica no menu “Play”.	
	2		O Sistema exibe o tabuleiro.
	3	O Jogador clica no botão esquerdo do mouse.	
	4		O Sistema inicia o jogo.
	5	Repetir os passos 6 a 13 10 vezes	
	6	O Jogador move o mouse para arremessar a bola.	
	7		O Sistema move a bola na pista.
	8		Se a bola atinge pelo menos um pino, então O Sistema move os pinos.
	9		O Sistema contabiliza o número de pinos derrubados.
	10	O Jogador move o mouse para arremessar a bola.	
	11		O Sistema move a bola na pista.
	12		Se a bola atinge pelo menos um pino, então O Sistema move os pinos.
	13		O Sistema contabiliza a pontuação final do Jogador.

Fonte: O autor.

Por fim, os testes são estruturados para serem adicionados nos arquivos que compõem as páginas da FitNesse e a fixture a ser utilizada no teste. Os testes na FitNesse e a fixture gerada são apresentadas nas Figuras 26 e 27.

Existem situações na descrição da frase que não é possível extrair um valor para a saída. Nesses casos, é atribuído o mesmo nome da saída esperada. Essa situação ocorre no caso das saídas “tabuleiro”, “bola na pista” e “pinos”. É importante ressaltar que para que o teste seja executado, é necessário fazer alguns ajustes na fixture gerada, como a definição da classe a ser testada.

Figura 27 – Fixture gerada para o caso de uso “Play Bowling”

```
port com.jayway.android.robotium.solo.Solo;
port android.test.ActivityInstrumentationTestCase2;

ublic class CenarioTeste1 extends ActivityInstrumentationTestCase2<T>{

  private Solo solo;
  private String menu;
  private String botão;
  private String mouse;


  public CenarioTeste1(){
      super(T.class);
  }

  protected void setUp() throws Exception {
      super.setUp();
      solo = new Solo(getInstrumentation(), getActivity());
  }

  protected void tearDown() throws Exception {
      super.tearDown();
  }

  public String getmenu(){
      return menu;
  }

  public void setmenu (String menu){
      this.menu=menu;
  }

  public String getbotão(){
      return botão;
  }

  public void setbotão (String botão){
      this.botão=botão;
  }

  public String getmouse(){
      return mouse;
  }

  public void setmouse (String mouse){
      this.mouse=mouse;
  }

  public String tabuleiro(){
      solo.click(menu);
      solo.click(botão);
      solo.drag(mouse);
      solo.drag(mouse);
      return "";
  }

  public String bolaNaPista(){
      solo.click(menu);
      solo.click(botão);
      solo.drag(mouse);
      solo.drag(mouse);
      return "";
  }

  public String pinos(){
      solo.click(menu);
      solo.click(botão);
      solo.drag(mouse);
      solo.drag(mouse);
      return "";
  }

  public String númeroDePinos (){
      solo.click(menu);
      solo.click(botão);
      solo.drag(mouse);
      solo.drag(mouse);
      return ""
  }

  public String pontuação (){
      solo.click(menu);
      solo.click(botão);
      solo.drag(mouse);
      solo.drag(mouse);
      return ""
  }

} //fim da classe
```

Fonte: O autor.

6.4 Conclusão

Neste capítulo foi apresentada uma aplicação da extensão do método ChAPTER, proposta neste trabalho, e da ferramenta que a implementa. A aplicação da extensão consiste da explicação passo-a-passo do processo para se gerar parcialmente os testes, pois ainda é necessário um esforço para executar os testes.

Para assegurar a avaliação do método, foram utilizadas duas linhas. Uma delas é a linha Mobiline, sensível ao contexto, cujo objetivo da mesma é desenvolver aplicações móveis e sensíveis ao contexto para guiar visitantes em vários lugares, como museus e laboratórios.

A outra linha é uma linha de produtos de software convencional, diferente da linha Mobiline por causa do objetivo, que é a geração de jogos que não sejam sensíveis ao contexto. Os jogos desenvolvidos a partir desta linha são definidos como sendo de arcade, cujo é objetivo é acumular mais pontos, conforme apresentado em (MAKER, 2014).

A aplicação da extensão com essas duas linhas permite assegurar que o método e a linguagem propostos permitem a geração parcial dos casos de testes tanto para linhas de produtos convencionais, quanto para linhas de produtos de software sensíveis ao contexto. É importante salientar que as informações de contexto, quando existentes, não são utilizadas no teste, mas guiam o testador sobre a situação contextual que os produtos ou códigos existentes da linha

devem estar executando para que o teste possa ser realizado.

7 CONCLUSÃO

O objetivo deste capítulo é resumir os resultados alcançados com este trabalho e também dar diretrizes para os trabalhos futuros. Este capítulo está então assim dividido: na Seção 7.1 são apresentados o resumo dos temas envolvidos e os resultados alcançados; e na Seção 7.2 são apresentados os trabalhos futuros.

7.1 Resultados Alcançados

Com o conceito de Linha de Produtos de Software é possível reutilizar artefatos e reduzir os custos envolvidos no desenvolvimento de aplicações de um mesmo domínio. Quando o objetivo das linhas é desenvolver aplicações sensíveis ao contexto, tem-se as Linhas de Produtos de Software Sensíveis ao Contexto.

Em relação ao teste em uma LPS, ele é mais complexo do que em uma aplicação tradicional, pois a complexidade de se definir e desenvolver uma linha é maior do que a definição e desenvolvimento de um único produto (aplicação), uma vez que uma linha abrange todo um domínio de aplicações. Quando se trata de aplicações sensíveis ao contexto, a complexidade ainda é maior do que em uma aplicação tradicional, pois envolve informações de contexto que variam conforme o ambiente do usuário e podem ser localização obtida por GPS, carga da bateria, entre outras. Portanto, o teste em uma LPSSC agrega outros desafios que vão impactar na sua complexidade.

Para auxiliar a atividade de testes, incluindo os testes em LPS, e reduzir os custos envolvidos na criação dos mesmos, eles podem ser gerados automaticamente a partir de artefatos de requisitos, técnica conhecida como teste baseado em requisitos. Um artefato que pode ser utilizado é o caso de uso, pois descreve o comportamento do sistema. Isso fica mais claro quando se possui casos de uso textuais. Portanto, o teste pode ser gerado a partir das descrições textuais dos casos de uso.

Nesse cenário, esta dissertação de mestrado estende o trabalho de (SANTOS, 2013), o qual define um método para geração de cenários de teste para LPSSC a partir de casos de uso. Uma das dificuldades encontradas pelo referido autor para poder gerar testes mais próximos da execução foi o fato dos casos de uso serem descritos em linguagem natural, que pode ser ambígua e imprecisa. Esta dissertação tenta resolver isso com a criação de uma linguagem natural controlada.Além disso , três resultados desta dissertação podem então ser mencionados,

sendo um deles principal e outros dois secundários. O resultado principal é o método que estende o ChAPTER para permitir a geração de casos de teste para LPSSC. Para alcançar esse resultado, a linguagem CARNAUbA foi especificada e a extensão da ferramenta ToChAPTER foi desenvolvida. Estes últimos são então os dois resultados secundários deste trabalho.

O primeiro resultado alcançado neste trabalho foi a definição de uma Linguagem Natural Controlada (LNC) que possui restrições sintáticas, semânticas e de vocabulário em relação à uma linguagem natural. A linguagem definida nesta dissertação foi denominada CARNAUbA, acrônimo de Controlled nAtuRal laNguAge for context-aware software product lines Use cAses. A CARNAUbA definiu restrições sintáticas sobre os campos do template de caso de uso definido em (SANTOS, 2013), o CAPLUC. Ela também definiu restrições semânticas para o elemento "Passo" do template. As restrições de vocabulário ficaram mais definidas na parte dos verbos, que indicam as ações executadas nos passos.

Em seguida, o segundo resultado alcançado foi a definição de uma extensão para o método ChAPTER com o objetivo de permitir que fossem gerados partes dos casos de teste e a estrutura necessária para facilitar a execução dos testes. Além de gerar parte dos casos de testes e a estrutura para a execução dos mesmos, a extensão gera valores para as entradas com base em informações passadas pelos testadores. Isso é realizado após a extração das informações do caso de uso que foi descrito na CARNAUbA.

A extensão proposta recebe cenários de teste do ChAPTER para poder gerar os testes a serem executados. Em cada cenário são analisadas as entradas e saídas existentes, para assim, gerar parte dos testes para cada cenário definido. Vale ressaltar que a geração dos testes por completo, incluindo as saídas esperadas para cada passo, é algo bastante complexo e que necessita de mais investimento, uma vez que pode ser considerada a interpretação de um programa, o que é um problema NP-Completo. O terceiro resultado alcançado foi a extensão da ferramenta ToChAPTER. Ela foi evoluída para permitir a execução da extensão do método proposto nesta dissertação.

Resumindo então os resultados alcançados, a ferramenta implementa o método aqui proposto que gera parte dos testes para uma LPSSC utilizando a LNC CARNAUbA, contribuindo assim para tornar os testes gerados mais próximos de testes executáveis.

Vale ainda ressaltar que paralelamente ao desenvolvimento desta dissertação, artigos foram produzidos. Um deles está diretamente ligado a este trabalho, que foi o artigo publicado no Workshop de Teses e Dissertações do Congresso Brasileiro de Software de 2013 (ARAÚJO *et*

al., 2013). Os outros artigos foram publicados no Simpósio Brasileiro de Sistemas de Informação (SBSI) de 2013 (SANTOS *et al.*, 2013) e no Workshop on Tools and Applications (WTA) do Brazilian Symposium on Multimedia and the Web (WebMedia) de 2013 (LIMA *et al.*, 2013). O artigo publicado no SBSI faz um comparativo entre ferramentas de teste para aplicações móveis e o artigo publicado no WTA do WebMedia apresentou o GREat Tour. Esses dois últimos artigos, embora não diretamente ligados à dissertação, foram importantes para o entendimento de parte da fundamentação desta dissertação: testes e aplicações móveis.

7.2 Trabalhos Futuros

Esta dissertação não é autocontida e abre possibilidades futuras de pesquisa. Dentre elas podem ser destacados os seguintes trabalhos futuros:

- Geração completa do teste, diminuindo ainda mais o esforço do testador na geração automática dos mesmos. Isso é importante, pois o método aqui proposto ainda não gera o teste por completo. Para isso, é necessário também evoluir a linguagem CARNAUbA inserindo uma maior análise semântica com o intuito de permitir que testes mais próximos da execução sejam criados. As limitações citadas nos capítulos 4 e 5 podem ser solucionadas neste trabalho futuro.
- Simulação de contexto para executar os testes. O método pode ser evoluído de maneira a permitir a simulação do contexto antes e durante a execução dos testes para garantir que dentro das condições especificadas, os produtos da linha executem corretamente e que fora dessas condições, os produtos possam estar aptos a tratar essas condições.
- Priorização dos testes criados. Dependendo da complexidade da linha podem existir diversos testes a serem realizados, porém, os custos e o tempo podem não permitir a execução dos mesmos, necessitando assim que os testes sejam priorizados.
- Execução de uma avaliação experimental. O método proposto para estender o ChAPTER, bem como a ferramenta podem ser avaliados de maneira experimental para assegurar que há uma redução no tempo e nos custos envolvidos na geração dos testes para uma LPSSC.
- Evolução da ferramenta. A ferramenta ainda pode ser evoluída para permitir uma melhor usabilidade e facilidade de uso para os usuários da mesma.

REFERÊNCIAS

ANDROID. Android. Disponível em http://www.android.com/. Último acesso em março de 2014. 2014.

ANTHONYSAMY, P.; SOMÉ, S. S. Aspect-oriented use case modeling for software product lines. In: **Proceedings of the 2008 AOSD workshop on Early aspects**. New York, NY, USA: ACM, 2008. (EA '08), p. 5:1–5:8. ISBN 978-1-60558-143-9. Disponível em: <http://doi.acm.org/10.1145/1404946.1404951>.

ARAÚJO, I.; ANDRADE, R.; NETO, P. S. Uma proposta para geração automática de testes para linhas de produto de software sensíveis ao contexto. In: **CBSoft 2013 - WTDSoft 2013**. [S.l.: s.n.], 2013.

ARAúJO, D. O. **Elaboração de especificações de casos de uso para linhas de produto de software baseada em fragmentos**. Dissertação — Universidade Federal do Rio de Janeiro, 2010.

BARROS, F. A.; NEVES, L.; HORI, E.; TORRES, D. The ucscnl: A controlled natural language for use case specifications. In: **SEKE**. Knowledge Systems Institute Graduate School, 2011. p. 250–253. ISBN 1-891706-29-2. Disponível em: <http://dblp.uni-trier.de/db/conf/seke/seke2011.html#BarrosNHT11>.

BECK, K. **Test-driven development: by example**. [S.l.]: Addison-Wesley Professional, 2003.

BERTOLINO, A. Software testing research: Achievements, challenges, dreams. In: **Future of Software Engineering, 2007. FOSE '07**. [S.l.: s.n.], 2007. p. 85 –103.

BERTOLINO, A.; FANTECHI, A.; GNESI, S.; LAMI, G.; MACCARI, A. Use case description of requirements for product lines. In: **Proceedings of the International Workshop on Requirements Engineering for Product Lines 2002 - REPL'02. Technical Report: ALR2002-033, AVAYA**. [S.l.: s.n.], 2002. p. 12–18.

BERTOLINO, A.; GNESI, S. Use case-based testing of product lines. In: . [S.l.: s.n.].

BONIFáCIO, R.; BORBA, P. Modeling scenario variability as crosscutting mechanisms. In: **Proceedings of the 8th ACM international conference on Aspect-oriented software development**. New York, NY, USA: ACM, 2009. (AOSD '09), p. 125–136. ISBN 978-1-60558-442-3. Disponível em: <http://doi.acm.org/10.1145/1509239.1509258>.

BUTHPITIYA, S.; LUQMAN, F.; GRISS, M.; XING, B.; DEY, A. Hermes – a context-aware application development framework and toolkit for the mobile environment. In: **26th International Conference on Advanced Information Networking and Applications Workshops (WAINA), 2012**. [S.l.: s.n.], 2012.

CARDEY, S.; GREENFIELD, P.; ANANTALAPOCHAI, R.; BEDDAR, M.; DEVITRE, D.; JIN, G. Modelling of multiple target machine translation of controlled languages based on language norms and divergences. In: **Universal Communication, 2008. ISUC '08. Second International Symposium on**. [S.l.: s.n.], 2008. p. 322–329.

CHEN, L.; LI, Q. Automated test case generation from use case: A model based approach. In: **3rd IEEE International Conference on Computer Science and Information Technology (ICCSIT), 2010**. [S.l.: s.n.], 2010.

CHOI, W.-s.; KANG, S.; CHOI, H.; BAIK, J. Automated generation of product use case scenarios in product line development. In: **8th IEEE International Conference on Computer and Information Technology, 2008. CIT 2008.** [S.l.: s.n.], 2008. p. 760–765.

COCKBURN, A. **Writing Effective Use Cases**. [S.l.]: Addison Wesley, 2001.

DANTAS, V. L. L. **Requisitos para Testes de Aplicações Móveis**. 132 p. Dissertação — Universidade Federal do Ceará, 2009.

DEY, A. K. Understanding and using context. **Personal and ubiquitous computing**, Springer, v. 5, n. 1, p. 4–7, 2001.

DU, W.; WANG, L. Context-aware application programming for mobile devices. In: **Proceedings of the 2008 C3S2E conference**. New York, NY, USA: ACM, 2008. (C3S2E '08), p. 215–227. ISBN 978-1-60558-101-9. Disponível em: <http://doi.acm.org/10.1145/1370256.1370292>.

ENSAN, A.; BAGHERI, E.; ASADI, M.; GASEVIC, D.; BILETSKIY, Y. Goal-oriented test case selection and prioritization for product line feature models. In: **Information Technology: New Generations (ITNG), 2011 Eighth International Conference on**. [S.l.: s.n.], 2011. p. 291–298.

ERIKSSON, M.; BÖRSTLER, J.; BORG, K. Marrying features and use cases for product line requirements modeling of embedded systems. In: **Proceedings of the Fourth Conference on Software Engineering Research and Practice in Sweden**. [S.l.: s.n.], 2004. p. 73–83.

FANTECHI, A.; GNESI, S.; JOHN, I.; LAMI, G.; DÖRR, J. Elicitation of use cases for product lines. In: **Software Product-Family Engineering**. [S.l.]: Springer, 2004. p. 152–167.

FERNANDES, P.; WERNER, C.; TEIXEIRA, E. An approach for feature modeling of context-aware software product line. **Journal of Universal Computer Science**, v. 17, n. 5, p. 807–829, 2011.

FERNANDES, P. C. C. **UBIFEX: Uma Abordagem para Modelagem de Caracteristicas de Linhas de Produtos de Software Sensível ao Contexto**. 1–133 p. Dissertação — Universidade Federal do Rio de Janeiro, 2009. Pp. 1–139.

FITNESSE. **FitNesse: The fully integrated standalone wiki and acceptance testing framework**. 201e. Disponível em: http://fitnesse.org/. Último acesso em março de 2014.

GALLINA, B.; GUELFI, N. A template for requirement elicitation of dependable product lines. In: **Proceedings of the 13th international working conference on Requirements engineering: foundation for software quality**. Berlin, Heidelberg: Springer-Verlag, 2007. (REFSQ'07), p. 63–77. ISBN 978-3-540-73030-9. Disponível em: <http://dl.acm.org/citation.cfm?id=1768904.1768909>.

GOIS, F. N. B.; FARIAS, P.; OLIVEIRA, R. Test script diagram–um modelo para geração de scripts de testes. **Anais do IX Simpósio Brasileiro de Qualidade de Software (SBQS'10)**, p. 73–87, 2010.

GOMMA, H. **Designing Software Product Lines with UML - From Use Cases to Pattern-Based Software Architectures**. [S.l.]: Addison-Wesley, 2004.

HIERONS, R. M.; BOGDANOV, K.; BOWEN, J. P.; CLEAVELAND, R.; DERRICK, J.; DICK, J.; GHEORGHE, M.; HARMAN, M.; KAPOOR, K.; KRAUSE, P.; LüTTGEN, G.; SIMONS, A. J. H.; VILKOMIR, S.; WOODWARD, M. R.; ZEDAN, H. Using formal specifications to support testing. **ACM Comput. Surv.**, ACM, New York, NY, USA, v. 41, n. 2, p. 9:1–9:76, fev. 2009. ISSN 0360-0300. Disponível em: <http://doi.acm.org/10.1145/1459352.1459354>.

JACOBSON, I.; GRISS, M.; JONSSON, P. **Software Reuse - Architecture, Process and Organization for Bussiness success**. [S.l.]: Addison-Wesley, 1997.

JOHN, I.; MUTHIG, D. Product line modeling with generic use cases. In: **SPLC-2 Workshop on Techniques for Exploiting Commonality Through Variability Management, Second Software Product Line Conference**. San Diego, Estados Unidos: [s.n.], 2002.

KANG, K. C.; COHEN, S. G.; HESS, J. A.; NOVAK, W. E.; PETERSON, A. S. **Feature-oriented domain analysis (FODA) feasibility study**. [S.l.], 1990.

KONG, L.; YUAN, T. Extension features-driven use case model for requirement traceability. In: **Computer Science Education, 2009. ICCSE '09. 4th International Conference on**. [S.l.: s.n.], 2009. p. 866–870.

KOSKELA, L. **Test Driven: Practical TDD and Acceptance TdDD for Java Developers**. Manning Publications Company, 2008. (Manning Pubs Co Series). ISBN 9781932394856. Disponível em: <http://books.google.com.br/books?id=_nQMGQAACAAJ>.

KUHN, T. How to evaluate controlled natural languages. **arXiv preprint arXiv:0907.1251**, 2009.

KUHN, T. An evaluation framework for controlled natural languages. In: **Controlled Natural Language**. [S.l.]: Springer, 2010. p. 1–20.

LIMA, E. R. R.; ARAúJO, I. L.; SANTOS, I. S.; OLIVEIRA, T. A.; MONTEIRO, G. S.; COSTA, C. E. B.; SEGUNDO, Z. F. S.; ANDRADE, R. Great tour: Um guia de visitas móvel e sensível ao contexto. In: **Webmedia 2013 - Workshop on Tools and Applications 2013**. [S.l.: s.n.], 2013. Artigo aceito para publicação.

LIU, Z.; GU, N.; YANG, G. An automate test case generation approach: using match technique. In: **Computer and Information Technology, 2005. CIT 2005. The Fifth International Conference on**. [S.l.: s.n.], 2005. p. 922–926.

LOKE, S. **Context-aware pervasive systems: architectures for a new breed of applications**. [S.l.]: Auerbach Publications, 2006.

LU, H. A context-oriented framework for software testing in pervasive environment. In: **Companion to the Proceedings of the 29th International Conference on Software Engineering**. Washington, DC, USA: IEEE Computer Society, 2007. (ICSE COMPANION '07), p. 77–78. ISBN 0-7695-2892-9. Disponível em: <http://dx.doi.org/10.1109/ICSECOMPANION.2007.10>.

LU, H. **A software testing framework for context-aware applications in pervasive computing**. Tese (Tese de Doutorado em Ciência da Computação) — Universidade de Hong Kong, Hong Kong, 2009.

LU, R. Pseudo natural language vs. controlled natural language. In: **2010 4th International Universal Communication Symposium**. [S.l.: s.n.], 2010. p. K–1–K–3.

MAIA, M. E.; ROCHA, L. S.; ANDRADE, R. M. Requirements and challenges for building service-oriented pervasive middleware. In: **Proceedings of the 2009 international conference on Pervasive services**. New York, NY, USA: ACM, 2009. (ICPS '09), p. 93–102. ISBN 978-1-60558-644-1. Disponível em: <http://doi.acm.org/10.1145/1568199.1568214>.

MAKER, A. G. Arcade game maker: Pedagogical product line. Disponível em http://www.sei.cmu.edu/productlines/ppl/. Último acesso em março de 2014. 2014.

MARINHO, F.; ANDRADE, R.; WERNER, C. A verification mechanism of feature models for mobile and context-aware software product lines. In: **Software Components, Architectures and Reuse (SBCARS), 2011 Fifth Brazilian Symposium on**. [S.l.: s.n.], 2011. p. 1–10.

MARINHO, F. G.; LIMA, F.; FILHO, J. B. F.; ROCHA, L.; MAIA, M. E. F.; AGUIAR, S. B. de; DANTAS, V. L. L.; VIANA, W.; ANDRADE, R. M. C.; TEIXEIRA, E.; WERNER, C. A software product line for the mobile and context-aware applications domain. In: **Proceedings of the 14th International Conference on Software Product Lines: Going Beyond**. [S.l.: s.n.], 2010.

MILUZZO, E.; LANE, N. D.; FODOR, K.; PETERSON, R.; LU, H.; MUSOLESI, M.; EISENMAN, S. B.; ZHENG, X.; CAMPBELL, A. T. Sensing meets mobile social networks: the design, implementation and evaluation of the cenceme application. In: **Proceedings of the 6th ACM conference on Embedded network sensor systems**. New York, NY, USA: ACM, 2008. (SenSys '08), p. 337–350. ISBN 978-1-59593-990-6. Disponível em: <http://doi.acm.org/10.1145/1460412.1460445>.

MOBILINE. Uma linha de produto de software móvel e sensível ao contexto. Disponível em http://mobiline.great.ufc.br/. Último acesso em setembro de 2013. 2013.

MOBILINE. Uma linha de produto de software móvel e sensível ao contexto. Disponível em http://mobiline.great.ufc.br/. Último acesso em março de 2014. 2014.

MYERS, G.; SANDLER, C.; BADGETT, T. **The Art of Software Testing**. Wiley, 2011. (ITPro collection). ISBN 9781118133132. Disponível em: <http://books.google.com.br/books?id=CmhLOaSJ7esC>.

NEBUT, C.; FLEUREY, F.; TRAON, Y. L.; JEZEQUEL, J.-M. Automatic test generation: a use case driven approach. **Software Engineering, IEEE Transactions on**, march 2006.

NETO, C. R. L. **SPLMT-TE: A Software Product Lines System Test Case Tool**. 1–135 p. Dissertação — Universidade Federal de Pernambuco, 2011. Pp. 1–135.

NORTHROP, L. Sei's software product line tenets. **Software, IEEE**, v. 19, n. 4, p. 32 – 40, jul/aug 2002. ISSN 0740-7459.

PENG. Peng: Processable english. Disponível em http://web.science.mq.edu.au/ rolfs/peng/example-specification.html. Último acesso em março de 2014. 2014.

POHL, K. **Requirements Engineering: Fundamentals, Principles, and Techniques**. 1st. ed. [S.l.]: Springer Publishing Company, Incorporated, 2010. ISBN 3642125778, 9783642125775.

POHL, K.; BÖCKLE, G.; LINDEN, F. van der. **Software Product Line Engineering: Foundations, Principles and Techniques**. Springer, 2010. ISBN 9783642063640. Disponível em: <http://books.google.com.br/books?id=N8cJkgAACAAJ>.

ROBOTIUM. Robotium: The world's leading android test automation framework. Disponível em https://code.google.com/p/robotium/. Último acesso em março de 2014. 2014.

RUBINOV, K. Generating integration test cases automatically. In: **Proceedings of the Eighteenth ACM SIGSOFT International Symposium on Foundations of Software Engineering**. New York, NY, USA: ACM, 2010. (FSE '10), p. 357–360. ISBN 978-1-60558-791-2. Disponível em: <http://doi.acm.org/10.1145/1882291.1882346>.

SALEHIE, M.; TAHVILDARI, L. Self-adaptive software: Landscape and research challenges. **ACM Trans. Auton. Adapt. Syst.**, ACM, New York, NY, USA, v. 4, n. 2, p. 14:1–14:42, maio 2009. ISSN 1556-4665. Disponível em: <http://doi.acm.org/10.1145/1516533.1516538>.

SANTOS, I. S. **Um ambiente para geração de cenários de testes para linhas de produto de software sensíveis ao contexto**. 1–133 p. Dissertação — Universidade Federal do Ceará, 2013. Pp. 1–133.

SANTOS, I. S.; BEZERRA, C. I. M.; MONTEIRO, G. S.; ARAUJO, I. L.; OLIVEIRA, T. A.; SANTOS, R. M.; DANTAS, V. L. L.; ANDRADE, R. M. C. Uma avaliação de ferramentas para testes em sistemas de informação móveis baseada no método dmadv. **Simpósio Brasileiro de Sistemas de Informação**, p. 1–13, 2013.

SANTOS, I. S.; SANTOS, A. R.; NETO, P. A. S. Reusing functional testing in order to decrease performance and stress testing costs. In: **International Conference on Software Engineering and Knowledge Engineering**. Miami Beach: [s.n.], 2011. p. 1–5. Proceedings of the 23rd International Conference on Software Engineering and Knowledge Engineering.

SCHNELTE, M. Generating test cases for timed systems from controlled natural language specifications. In: **Secure Software Integration and Reliability Improvement, 2009. SSIRI 2009. Third IEEE International Conference on**. [S.l.: s.n.], 2009. p. 348–353.

SCHWITTER, R.; TILBROOK, M. Annotating websites with machine-processable information in controlled natural language. In: **Proceedings of the second Australasian workshop on Advances in ontologies - Volume 72**. Darlinghurst, Australia, Australia: Australian Computer Society, Inc., 2006. (AOW '06), p. 75–84. ISBN 1-920-68253-8. Disponível em: <http://dl.acm.org/citation.cfm?id=1273659.1273669>.

SELBY, R. Enabling reuse-based software development of large-scale systems. **Software Engineering, IEEE Transactions on**, v. 31, n. 6, p. 495–510, June 2005. ISSN 0098-5589.

SHAMSODDIN-MOTLAGH, E. A review of automatic test cases generation. **International Journal of Computer Applications**, v. 57, n. 13, p. 25–29, November 2012. Published by Foundation of Computer Science, New York, USA.

SIQUEIRA, H. L. F. **TaRGeT Scripts Generation: Um plug-in de geração automática de scripts de teste**. 68 p. Monografia (Trabalho de Conclusão de Curso) — Universidade Federal de Pernambuco, Recife, 2010. Trabalho de Conclusão de Curso. Universidade Federal de Pernambuco.

SOMÉ, S. S. Supporting use case based requirements engineering. **Information and Software Technology**, v. 48, n. 1, p. 43 – 58, 2006. ISSN 0950-5849. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0950584905000285>.

SOMÉ, S. S.; CHENG, X. An approach for supporting system-level test scenarios generation from textual use cases. In: **Proceedings of the 2008 ACM symposium on Applied computing**. New York, NY, USA: ACM, 2008. (SAC '08), p. 724–729. ISBN 978-1-59593-753-7. Disponível em: <http://doi.acm.org/10.1145/1363686.1363857>.

UML. Unified modeling language. Disponível em http://uml.org/. Último acesso em março de 2014. 2014.

WANG, Z.; ELBAUM, S.; ROSENBLUM, D. Automated generation of context-aware tests. In: **Software Engineering, 2007. ICSE 2007. 29th International Conference on**. [S.l.: s.n.], 2007. p. 406 –415. ISSN 0270-5257.

APÊNDICE A – LISTA DE VERBOS ACEITOS NA CARNAUBA

Verbo	Significado	Comandos na fixture	
		Para o ator	Para o sistema
Seleciona	Indica que uma opção foi escolhida	Solo.click	--
Exibe	Indica que o sistema informou algo para o ator	--	Solo.search
Retorna	Indica que uma informação foi retornada	--	assertEquals
Digita	Indica que o ator inseriu informações no sistema	Solo.enterText	
Recebe	Indica que o sistema recebe uma informação do ator	--	--
Clica	Indica que o ator clicou em algum ponto da tela	Solo.click	--
Recupera	Indica que informações são recuperadas	--	--
Cadastra	Indica que dados devem ser cadastrados no sistema	--	--
Insere	Indica que a aplicação inseriu algo no banco de dados	--	assertEquals
Deleta	Indica que informações foram deletadas do banco de dados	--	assertEquals

Altera	Indica que algum dado foi alterado no banco de dados	--	assertEquals
Valida	Indica que dados foram validados pela aplicação	--	assertEquals
Aguarda	Indica que há uma espera pela ação do ator ou do sistema	Solo.wait	Solo.wait
Executa	Indica que o sistema deve executar alguma ação	--	--
Autentica	Indica que o sistema autentica os dados do ator para prover serviços específicos	--	--
Solicita	Indica que o sistema fez alguma solicitação	--	--
Move	Indica que houve um clique e um movimento na tela do sistema	Solo.drag	--
Contabiliza	Informa que o sistema realizou uma operação para contabilizar informações	--	--
Inicia	Indica que o sistema iniciou uma operação	--	--
Reproduz	Indica que o sistema reproduz algum arquivo	--	--