PONTIFÍCIA UNIVERSIDADE CATÓLICA DE MINAS GERAIS
Programa de Pós-Graduação em Informática

Carolina Diniz Cunha

UM AMBIENTE PARA GERAÇÃO AUTOMÁTICA DE TESTES A PARTIR DE CASOS DE USO

Belo Horizonte

2015

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

Cunha, Carolina Diniz
C972a Um ambiente para geração automática de testes a partir de casos de uso / Carolina Diniz Cunha. Belo Horizonte, 2015.
169 f. : il.

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

1. Software - Testes. 2. Software de aplicação – Desenvolvimento. 3. Automação. 4. Programação para Internet. 5. Sistemas de computação. I. Song, Mark Alan Junho. II. Pontifícia Universidade Católica de Minas Gerais. Programa de Pós-Graduação em Informática. III. Título.

CDU: 681.3.03.001

Carolina Diniz Cunha

UM AMBIENTE PARA GERAÇÃO AUTOMÁTICA DE TESTES A PARTIR DE CASOS DE USO

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

Orientador: Prof. Dr. Mark Alan Junho Song

Belo Horizonte
2015

Carolina Diniz Cunha

UM AMBIENTE PARA GERAÇÃO AUTOMÁTICA DE TESTES A PARTIR DE CASOS DE USO

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

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

Prof. Dr. Ricardo Terra Nunes Bueno Villela (UFLA)
Convidado

Prof. Dr. Humberto Torres Marques Neto (PUC Minas)
Convidado

Belo Horizonte, 27 de abril de 2015

Aos meus pais.

AGRADECIMENTOS

À família, amigos e namorado, que apoiaram e compreenderam tantos momentos de ausência. Ao orientador, que me auxiliou a rever conceitos e superar minhas dificuldades. À banca, com suas sugestões que tanto aprimoraram o trabalho. À Capes, pelo auxílio tão bem-vindo, em parte do curso. Ao pessoal da secretaria do curso, que sempre ajudou tranquilizando e esclarecendo questões, nos momentos mais difíceis. Enfim, agradeço a oportunidade de ter convivido com pessoas tão admiráveis, em especial minhas colegas de classe Monica e Amanda. Gratidão.

“Tudo o que vive, quer viver !”
(Francisco de Assis)

RESUMO

Esta dissertação propõe um ambiente para geração automática de testes funcionais para aplicações *web*, a partir de especificação de casos de uso. O ambiente é baseado em uma linguagem de domínio específico própria, formalizada por meio de uma gramática livre de contexto. Buscou-se estreitar a relação entre a especificação e verificação de sistemas, visando facilitar e incentivar o uso de testes automatizados e a manutenção dos mesmos, que frequentemente são abandonados devido a limitações de custos e prazos. Todas as ferramentas utilizadas no processo, desde a especificação, geração, execução e gerenciamento de testes, são gratuitas e de domínio público. Além disso, apresenta-se os resultados de um estudo de caso em que a abordagem proposta é aplicada a sistemas reais, para análise de viabilidade da solução.

Palavras-chave: testes-funcionais; casos-de-uso; automatização.

ABSTRACT

This dissertation proposes an environment for use case specification and automated functional test generation for web-based applications. The environment is based on our own domain-specific language, formalized by means of a context-free grammar. The proposal seeks to tighten the relation between system verification and specification, aiming to facilitate and encourage the long-term use of automated tests, which are frequently abandoned due to cost and deadline limitations. All tools used in the process, including test generation, execution, management and specification, are free and in the public domain. Additionally, we present the results of a case study in which the proposed approach is applied to real systems, to gauge our solution's feasibility.

Keywords: functional-tests; use-cases; automation.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE QUADROS

LISTA DE LISTAGENS

LISTA DE ABREVIATURAS E SIGLAS

API	*Application Programming Interface*
BNF	*Backus-Naur Form*
CRUD	*Create, Read, Update, Delete*
CT	Caso de teste
DSL	Linguagem de domínio específico
EnvUCT	*Environment for Use Cases to Tests*
IU	Interface de usuário
LUCT	*Language of Use Cases to Test*
SST	Sistema sob teste
UC	Caso de uso
UC2Test	*Use Case to Test*
XMI	*Extensible Markup Language Metadata Interchange*
XML	*Extensible Markup Language*

SUMÁRIO

1 INTRODUÇÃO

Este capítulo apresenta uma visão geral do problema tratado nesta dissertação e da solução proposta. A Seção 1.1 apresenta a motivação para o desenvolvimento deste trabalho de pesquisa. A Seção 1.2 expõe os objetivos gerais e específicos. E, por fim, a Seção 1.3 descreve a estrutura desta dissertação.

1.1 Motivação

Testes são atividades fundamentais no desenvolvimento de um software, pois correspondem ao último recurso para avaliação da qualidade do produto antes da entrega ao usuário final (PRESSMAN, 2011). Dentre os vários tipos de testes destaca-se o teste funcional, em que o sistema é avaliado em relação às suas entradas e saídas, sem considerar seu funcionamento interno. Por ser baseado na especificação, testes funcionais são largamente executados – uma vez que é necessário verificar se o software se comporta de acordo com o que foi definido na análise de requisitos (PEZZÈ; YOUNG, 2008).

A demanda para que os sistemas de software sejam desenvolvidos com cada vez mais agilidade e qualidade exige a execução de testes com frequência; em especial, os testes de regressão, para garantir que novas funcionalidades não comprometam o correto comportamento de requisitos já implementados e validados (LUNA; ROSSI; GARRIGÓS, 2011). O teste de regressão é uma atividade importante, porém, uma das mais custosas nas fases de desenvolvimento e manutenção de software. Segundo Harrold (2009), estima-se que consumam até 80% do custo total gasto com atividades de teste e até 50% do custo de manutenção do software.

A automatização reduz o tempo necessário para a execução dos testes, possibilitando sua realização em maior número ou frequência, o que favorece a realização de testes de regressão (RIOS; FILHO, 2006). Além de possibilitar a redução do tempo gasto na fase de testes, a utilização de testes automatizados permite um aumento indireto da qualidade do software, uma vez que possibilita que os esforços sejam concentrados em outros tipos de teste ou em testes que não possam ser automatizados (PEDEMONTE; MAHMUD; LAU, 2012). Automatizar tarefas repetitivas não apenas reduz custos, mas melhora a precisão, pois os seres humanos não são tão ágeis como computadores, além de serem sujeitos a erros, quando lidam com tais tarefas (PEZZÈ; YOUNG, 2008). É notório que a geração de casos de testes funcionais é, ainda, uma tarefa preponderadamente manual e trabalhosa (MARIANI et al., 2011), assim como sua execução, quadro especialmente complicado em projetos grandes e de longa duração (PEDEMONTE; MAHMUD; LAU, 2012).

Para a automatização dos testes, são construídos *scripts* que reproduzem os testes manuais. A geração de *scripts* pode ser feita por codificação manual ou por meio de uma ferramenta de gravação. Com tais ferramentas, as ações do usuário sobre a interface do sistema são gravadas, e um código é gerado reproduzindo tais ações (PEDEMONTE; MAHMUD; LAU, 2012). Esse código pode ser re-executado posteriormente, simulando a navegação do usuário. Tal técnica é denominada Captura e Repetição.

À primeira vista, a técnica de Captura e Repetição mostra-se uma boa solução por não exigir conhecimentos técnicos de programação – necessários na codificação manual. Mesmo dominando-se tais conhecimentos, a facilidade na geração dos *scripts* pode levar à escolha dessa opção. Entretanto, em situações de manutenção, realizar novamente as execuções manuais para a gravação de um novo código pode tornar-se trabalhoso, dependendo do número de execuções a serem refeitas. E alterar o código gerado automaticamente nem sempre é tarefa simples, devido ao fato do mesmo ser de difícil redigibilidade e manutenção (PEDEMONTE; MAHMUD; LAU, 2012). Ademais, a ligação entre a página de teste e o código gerado é fraca, sendo difícil encontrar as razões específicas de um erro apontado na execução dos testes (WANG; XU, 2009).

Apesar das vantagens na utilização de testes automatizados, há um custo considerável para mantê-los, pois a cada alteração no sistema sob teste, os *scripts* de teste precisam ser verificados, e, se necessário, sofrer as devidas manutenções. Consequentemente, somente uma parte (por vezes menos de 20%) dos testes chegam a ser automatizados (THUMMALAPENTA et al., 2012). Ao se optar pela automação de testes, deve-se levar em conta os custos de manutenção dos *scripts* de teste, que devem ser mantidos atualizados, para gerarem os benefícios da automação.

Frequentemente, devido a limitações de custo e prazo, manutenções nos testes automatizados deixam de ser realizadas por não serem prioritárias. Sem a manutenção necessária, esses perdem seu valor, podendo ser abandonados (RIOS; FILHO, 2006).

Uma forma de se reduzir o esforço na fase de testes e garantir sua efetividade, é gerar casos de teste automaticamente a partir de artefatos usados em fases anteriores de processo de desenvolvimento (SOME; CHENG, 2008). Uma das formas mais utilizadas para capturar requisitos funcionais de um sistema é por meio da descrição de Casos de Uso, mecanismo central recomendado no Processo Unificado (PU) e em muitos métodos modernos (SOMÉ, 2006; LARMAN, 2007). Casos de teste derivados de casos de uso podem ser utilizados para verificar se os requisitos do sistema (casos de uso) foram corretamente implementados (HEUMANN, 2001; SOME; CHENG, 2008).

Entretanto, a falta de formalidade, comum na descrição de casos de uso textuais, dificulta sua utilização em processos automáticos para geração de casos de teste. Formalizar e estruturar a descrição de casos de uso textuais, além de reduzir o número de variações na sua interpretação,

é um passo essencial para a sua transformação em código de teste, de forma automática (JIANG; DING, 2011; PINTO; STAA, 2013).

1.2 Objetivo

O presente trabalho tem, por objetivo geral, propor um ambiente para especificação de casos de uso e sua derivação, de forma automática, em testes funcionais para aplicações *web*.

Como objetivos específicos, encontram-se:

- Definir uma linguagem de domínio específico (DSL) para a descrição de casos de uso, dicionário de dados e interfaces de usuário;
- Propor um ambiente com validações em tempo de edição, que auxiliem o usuário durante a especificação;
- Propor uma ferramenta que realize, a partir da descrição de um sistema de software no ambiente proposto, a geração automática de cenários, valores e oráculos de teste, resultando em casos de teste automatizados.

1.3 Estrutura da Dissertação

Este trabalho está organizado em cinco capítulos como a seguir. O Capítulo 2 apresenta a fundamentação teórica, em que são abordados temas relacionados a este trabalho, como casos de uso, testes de software e automação, visando fornecer a base conceitual para o entendimento da abordagem proposta.

O Capítulo 3 apresenta a solução proposta, em que são descritos a metodologia adotada, a DSL desenvolvida para este trabalho de pesquisa, o processo de transformação da especificação em casos de teste, sua execução e análise dos resultados.

O Capítulo 4 demonstra a viabilidade técnica da abordagem proposta, em que são descritos experimentos e a análise dos resultados obtidos.

O Capítulo 5 apresenta alguns trabalhos relacionados a esta pesquisa, considerados relevantes para o desenvolvimento da solução proposta.

O Capítulo 6 conclui este trabalho, descrevendo as vantagens, principais contribuições e limitações encontradas na abordagem proposta, além de possíveis trabalhos futuros.

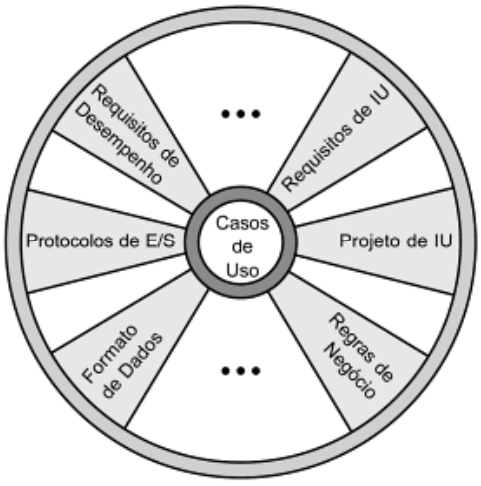
2 FUNDAMENTAÇÃO TEÓRICA

Este capítulo apresenta uma revisão de literatura de temas relacionados a este trabalho de pesquisa, visando fornecer a base conceitual para a abordagem proposta. O capítulo está organizado da seguinte forma: a Seção 2.1 apresenta os casos de uso textuais, sua estrutura, níveis, formatos e cenários. A Seção 2.2 descreve um dicionário de dados. A Seção 2.3 apresenta conceitos relacionados a testes de software, como critérios, tipos, automação e ferramentas de apoio. Também nessa seção é abordada a geração de casos de teste a partir de casos de uso. Por fim, a Seção 2.4 apresenta o *framework* Xtext, seu funcionamento e notação utilizada.

2.1 Casos de Uso

Um caso de uso é uma descrição de um conjunto de sequências de ações (fluxos de eventos) que um sistema executa para produzir um resultado observável para um ator. Esses fluxos incluem variantes, ou seja, além do fluxo principal, suas possíveis variações, como fluxos alternativos ou de exceção (BOOCH; RUMBAUGH; JACOBSON, 2005). Por conterem um levantamento de todas as condições que podem causar falhas e os respectivos tratamentos, casos de uso são excelentes descrições de testes funcionais. Além disso, proveem uma armação que conecta informações em diferentes partes dos requisitos, ajudando a ligar informações de perfis de usuário, regras de negócio ou requisitos de formato de dados (Figura 1) (COCKBURN, 2005).

Figura 1 – Modelo de requisitos "eixo-e-raios"



Fonte: Cockburn (2005), p. 32

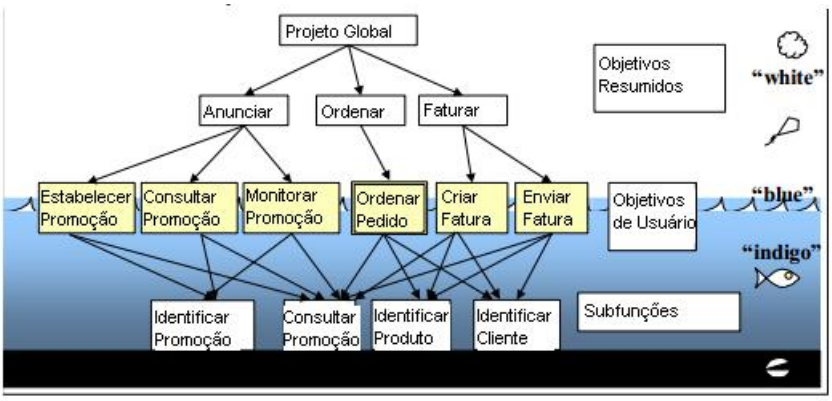
Casos de uso são bastante populares, entre outras razões, por usar uma estrutura semi-formal que facilita sua compreensão por pessoas leigas (JIANG; DING, 2011). As subseções a seguir apresentam os casos de uso textuais, sua estrutura, níveis, formatos e cenários.

2.1.1 Níveis de Objetivos de Casos de Uso

Os passos em um caso de uso descrevem as etapas que compõem um processo; o nome do caso de uso (objetivo) indica porque o processo é de interesse. Tanto os objetivos quanto as interações em um cenário de caso de uso podem ser desdobrados em objetivos e interações mais refinados e mais granulados (COCKBURN, 2005).

Em Cockburn (2005), os níveis de objetivo de casos de uso são apresentados como Objetivos Resumidos, Objetivos de Usuário e Subfunções. Objetivos de níveis inferiores constituem partes dos objetivos de níveis superiores, respondendo à pergunta "Como", enquanto os objetivos superiores respondem à pergunta "Por que" (Figura 2).

Figura 2 – Níveis de casos de uso: O conjunto de casos de uso revela uma hierarquia de objetivos.



Fonte: Cockburn (2005), p. 74

O nível Objetivo do Usuário corresponde a objetivos que o usuário tem, ao utilizar o sistema, em uma única seção. Objetivos de nível Resumo têm como propósito mostrar o contexto no qual os objetivos de usuário operam e a sequência do ciclo de vida de objetivos relacionados. Já os objetivos de nível Subfunção são aqueles necessários para a realização de objetivos de usuário. São oportunamente necessários para legibilidade ou por serem utilizados por vários outros objetivos, como por exemplo, "Encontrar um produto", "Encontrar um cliente" ou "Salvar um arquivo" (COCKBURN, 2005).

No presente trabalho utiliza-se os níveis de objetivo descritos nesta seção.

2.1.2 Formato do Caso de Uso

Em geral, os casos de uso são descritos inicialmente de forma narrativa informal. Sendo necessária maior formalidade, os mesmos são reescritos utilizando um formato estruturado (PRESSMAN, 2011). Em Larman (2007), o nível de formalidade de um caso de uso textual é classificado em:

- Resumido: Resumo sucinto de um parágrafo, geralmente o cenário principal. Utilizado durante a análise inicial de requisitos para obter uma rápida idéia do assunto e do escopo;
- Informal: Formato informal de parágrafos. Múltiplos parágrafos cobrem vários cenários;
- Completo: Todos os passos e variantes são escritos em detalhe, com seções de suporte, como pré-condições e garantias de sucesso.

Quadro 1 – Exemplo de estratégia para o desenvolvimento de casos de uso no PU

Disciplina	**Artefato**	**Comentários e Nível de Esforço em Requisitos**				
		Concep. **1 semana**	**Elab. 1** **4 semanas**	**Elab. 2** **4 semanas**	**Elab. 3** **3 semanas**	**Elab. 4** **3 semanas**
Requisitos	Modelo de Casos de Uso	Seminário de requisitos de dois dias. A maioria dos casos de uso é identificada por nome, e estes são resumidos em um parágrafo curto. Pegue 10% dos casos da lista dos mais relevantes para analisar e redigir em detalhe. Esses 10% serão os mais arquiteturalmente importantes, arriscados e de maior valor de negócio.	Perto do fim desta iteração faça um seminário de requisitos de dois dias. Obtenha percepções e realimentação a partir do trabalho de implementação, e, então complete 30% dos casos de uso em detalhe.	Perto do fim desta iteração faça um seminário de requisitos de dois dias. Obtenha percepções e realimentação a partir do trabalho de implementação, e, então complete 50% dos casos de uso em detalhe.	Repita e complete 70% de todos os casos de uso em detalhe.	Repita com a meta de ter 80-90% dos casos de uso esclarecidos e redigidos em detalhes. Somente uma pequena parte destes foi construída na elaboração; o restante é feito na construção.

Fonte: Larman (2007), p. 122

Em métodos de desenvolvimento evolutivos e iterativos, a reescrita dos casos de uso em detalhe é realizada incrementalmente ao longo das iterações. No Processo Unificado (PU), esse detalhamento tem início na fase de concepção, concentrando-se nas iterações da fase de elaboração (LARMAN, 2007). O Quadro 1 exibe um exemplo de estratégia no PU para o desenvolvimento de casos de uso. A análise de requisitos de forma escalonada beneficia-se de percepções e realimentação de informações obtidas pela implementação dos casos de uso, com qualidade de produção, desde as primeiras fases de elaboração. No presente trabalho casos de uso podem ser descritos inicialmente de forma informal, devendo estar completos e estruturados no momento de geração dos testes.

Vários formatos podem ser utilizados para a descrição estruturada de casos de uso, como o formato “Inteiramente Completo”, o formato de duas colunas e o template do Rational Unified Process (RUP). O formato “Inteiramente Completo” utiliza uma coluna de texto e passos numerados com uma convenção com números e letras para passos alternativos. O template RUP é similar, porém, a numeração de passos é opcional. Existem seções próprias com títulos, na descrição dos fluxos básico e alternativos. Já o formato de duas colunas separa as ações do ator primário das ações do sistema, auxiliando na visão da conversação, útil quando se analisa a interface de usuário (COCKBURN, 2005).

No presente trabalho utiliza-se casos de usos no formato “Inteiramente Completo”, cuja estrutura de numeração fornece informação suficiente para a geração de cenários.

2.1.3 Estrutura do Caso de Uso

Um caso de uso é composto de várias partes, descritas a seguir (COCKBURN, 2005):

- Nome: Objetivo do caso de uso, na forma de uma pequena frase de verbo ativo.
- Contexto: Detalhes do contexto de uso, se necessário.
- Escopo: Um dentre estes: Sistema <nome do sistema> ou Subsistema <nome do subsistema>. Não são utilizados os escopos “Empresa” e “Componente”, pois os mesmos se referem a casos de uso de negócio ou detalhes de implementação, não tratados por testes funcionais.
- Nível: Um dentre estes: Resumo, Objetivo do Usuário ou Subfunção (detalhes na Seção 2.1.1).
- Ator: Nome ou papel do ator primário.
- *Stakeholders* e interesses: Lista de *stakeholders* e interesses chaves no caso de uso.
- Pré-condição: É o que o sistema garantirá ser verdadeiro, antes do início do caso de uso. Geralmente uma pré-condição indica que algum outro casos de uso já foi executado, para estabelecê-la.
- Acionador (ou gatilho): especifica o evento que faz o caso de uso começar.
- Cenário principal: Conjunto de passos do cenário principal.
- Extensões: Conjunto de passos que desviam do cenário principal, gerando caminhos alternativos de sucesso e caminhos de falha. Cada extensão indica o passo do cenário principal em que o desvio pode ocorrer e a condição para o acionamento do mesmo.

- Garantias mínimas: são as menores promessas que o sistema faz aos *stakeholders*, na execução de todos os cenários, inclusive naqueles em que o objetivo não é alcançado.

- Garantias de sucesso: estabelecem promessas que o sistema faz aos *stakeholders*, para os cenários que terminam de forma bem sucedida.

- Lista de variações tecnológicas e de dados: Indica formas diferentes com que um passo pode ser feito. Por exemplo, no passo "Reembolsar cliente", pode-se ter a variação "Reembolsar por cheque, TED, ou vale-compras".

- Informação relacionada: Toda informação adicional necessária ao projeto.

2.1.4 Cenários de Casos de Uso

Casos de uso agrupam um conjunto de possíveis cenários para se alcançar um objetivo. Alguns cenários mostram o objetivo sendo alcançado, enquanto outros terminam com o objetivo sendo abandonado. Cada cenário contém uma sequência de passos, que mostram como se desdobram as ações e interações do usuário com o sistema (COCKBURN, 2005).

Figura 3 – Fluxo básico e fluxos alternativos para um caso de uso

Fonte: tradução livre de Heumann (2001)

A Figura 3 apresenta a estrutura típica de fluxos de eventos de um caso de uso. A seta reta representa o fluxo básico, e as setas curvas, os fluxos alternativos. Observe que alguns fluxos alternativos retornam ao fluxo básico de eventos, enquanto outros terminam o caso de uso. Um cenário de caso de uso é uma instância do caso de uso, ou um caminho completo através dele. O Quadro 2 lista todos os possíveis cenários do diagrama. O *cenário1* é chamado de cenário básico ou principal.

Quadro 2 – Cenários para o caso de uso da Figura 3

Cenário1	Fluxo Básico			
Cenário2	Fluxo Básico	Fluxo Alternativo 1		
Cenário3	Fluxo Básico	Fluxo Alternativo 1	Fluxo Alternativo 2	
Cenário4	Fluxo Básico	Fluxo Alternativo 3		
Cenário5	Fluxo Básico	Fluxo Alternativo 3	Fluxo Alternativo 1	
Cenário6	Fluxo Básico	Fluxo Alternativo 3	Fluxo Alternativo 1	Fluxo Alternativo 2
Cenário7	Fluxo Básico	Fluxo Alternativo 4		
Cenário8	Fluxo Básico	Fluxo Alternativo 3	Fluxo Alternativo 4	

Fonte: Heumann (2001)

2.2 Glossário e Dicionário de Dados

Um glossário, em sua forma mais simples, é uma lista de termos relevantes e suas definições, útil para reduzir problemas de comunicação e ambiguidade nos requisitos. Já um dicionário de dados é um documento que registra requisitos relativos aos dados do sistema, tais como formatos (tipo, tamanho, unidade), termos compostos, relacionamento entre elementos, intervalo de valores válidos e regras de validação. Um exemplo de dicionário de dados é apresentado no Quadro 3.

Quadro 3 – Exemplo de dicionário de dados

Termo	**Definição e Informações**	**Formato**	**Regras de Validação**	**Sinônimos**
Item	Um produto ou serviço para venda.			
Autorização de Pagamento	Validação por um serviço externo de autorização de pagamento que fará ou garantirá o pagamento para o vendedor.			
Solicitação de Autorização de Pagamento	Uma composição de elementos é enviada eletronicamente para um serviço de autorização, normalmente como uma sequência de caracteres. Os elementos incluem o ID da loja, o número da conta do cliente, a quantia e uma marca de tempo.			
CUP	Código numérico que identifica um produto. Geralmente é simbolizado por um código de barras colocado nos produtos. Ver www.uc-council.org para detalhes sobre formato e validação.	Código de 12 dígitos com diversas sub-partes.	Dígito 12 é um dígito de controle.	UPC - Universal Code Product ou Código Universal de Produto.
...	...			

Fonte: Larman (2007), p.141

No PU, o glossário é normalmente criado na fase de concepção, podendo ser expandido para a função de dicionário de dados durante a fase de elaboração (Quadro 4) (LARMAN, 2007).

Quadro 4 – Amostra de artefatos do PU e sequência temporal (i=início, r=refino)

Disciplina	Artefato Iteração ->	Concep. I1	Elab. E1.En	Const. C1..Cn	Trans. T1..T2
Modelagem de Negócio	Modelo de Domínio		i		
Requisitos	***Modelo de Caso de Uso***	i	r		
	Visão	i	r		
	Especializações Suplementares	i	r		
	Glossário	i	r		
Projeto	Modelo de Projeto		i	r	
	Documento de Arquitetura de Software		i		

Fonte: Larman (2007), p. 123

2.3 Testes de Software

Teste é a atividade na qual um sistema ou componente é executado sob condições especificadas, os resultado são observados ou gravados, e é feita uma avaliação sobre algum aspecto desse sistema ou componente (IEEE, 2008). Com tal avaliação, visa-se verificar se o comportamento do sistema está de acordo com o esperado (DELAMARO; MALDONADO; JINO, 2007).

Na prática, dificilmente pode-se testar um programa completamente e assim garantir a ausência de erros. Devido às muitas possibilidades em sua utilização, seja na entrada de dados, nos caminhos internos e condições, é praticamente impossível testar todas as combinações existentes. Contudo, o propósito dos testes é descobrir e corrigir defeitos, e com isso, melhorar a qualidade do software (MYERS; SANDLER; BADGETT, 2011; RIOS; FILHO, 2006).

Segundo Delamaro, Maldonado e Jino (2007), "Defeito (do inglês, *fault*) é um passo, processo ou definição de dados incorretos. A existência de um defeito pode ocasionar a ocorrência de um erro (*error*) durante a execução de um programa, que se caracteriza por um estado inconsistente ou inesperado. Tal estado pode levar a uma falha (*failure*), ou seja, pode fazer com que o resultado produzido pela execução seja diferente do resultado esperado". Os autores ressaltam que, no geral, o termo "erro" é utilizado de maneira bastante flexível, muitas vezes significando defeito ou falha.

"Sistema sob teste" (SST) é a aplicação (ou a parte dela) que está sendo exercitada e verificada pelos testes (IEEE, 2010).

As subseções a seguir apresentam conceitos relacionados a testes de software, como critérios, tipos, automação e ferramentas de apoio. Também é apresentada uma subseção abordando a geração de casos de teste a partir de casos de uso.

2.3.1 Caso de Teste

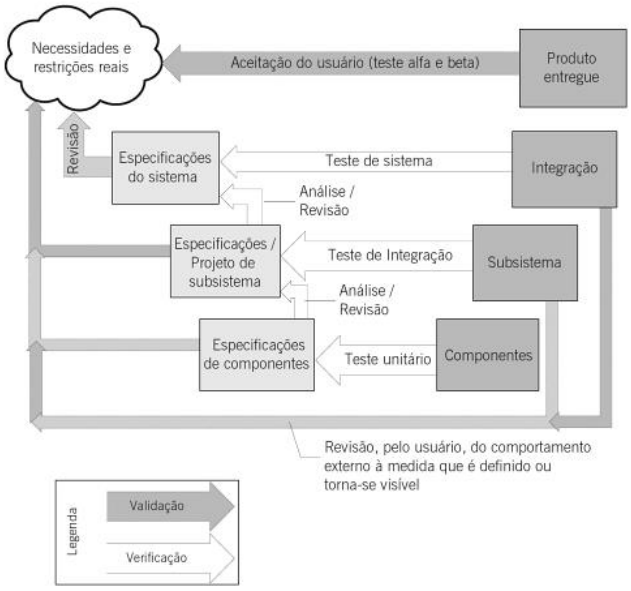
Um caso de teste especifica como deve ser testada uma parte do software. Essa especificação inclui valores de entrada, condições de execução e resultados esperados, desenvolvidos para um determinado objetivo, tais como para exercitar o caminho de um determinado programa ou verificar o atendimento a um requisito específico (IEEE, 2008).

Um procedimento de teste detalha as ações a serem executadas em um determinado caso de teste. Um caso de teste só faz sentido se for especificado seu procedimento de teste. Sem essa descrição, não é possível determinar o que deve ser feito com as entradas especificadas no caso de teste (NETO, 2006). Um oráculo é qualquer agente (humano ou não) que decide se um software funcionou corretamente em um determinado teste.

2.3.2 Estágios (ou Níveis) de Teste

Testes são executados em diferentes estágios (ou níveis) do desenvolvimento de um software, e vão desde testes de componentes até testes de sistemas completos. A Figura 4 apresenta os quatro níveis típicos de teste de software e ilustra a relação das atividades de verificação e validação com os artefatos produzidos no desenvolvimento de software. As setas brancas são atividades de verificação, enquanto as cinzas, de validação.

Figura 4 – Níveis de teste



Fonte: Pezzè e Young (2008), p. 41

Verificação é a checagem da consistência de uma implementação com uma especificação (declaração sobre uma solução particular proposta para um problema). Já a validação é avaliar o grau em que um sistema de software realmente satisfaz seus requisitos, no sentido de atender às

reais necessidades do usuário. Satisfazer requisitos não é o mesmo que estar conforme a uma especificação de requisitos. A solução proposta em uma especificação pode ou não ter conseguido atingir seus objetivos, além de poder conter erros, por ter sido desenvolvida por pessoas. As atividades de validação referem-se, principalmente, à especificação geral do sistema e a seu código final. Testes de validação entre a especificação e o produto final são, basicamente, testes de decisões deixados em aberto na especificação, como detalhes da interface ou características do produto (PEZZÈ; YOUNG, 2008).

O presente trabalho aborda testes funcionais, no nível de Teste de Sistema. Abaixo seguem descrições dos quatro níveis de teste (RIOS; FILHO, 2006):

- Teste de Unidade: Neste nível, é testada cada unidade do sistema que pode ser verificada isoladamente. Os testes de unidade são normalmente executados pelo próprio desenvolvedor do código, podendo ser considerados parte da implementação. Dessa forma, a execução de testes de unidade é realizada predominantemente utilizando-se técnicas caixa-branca.

- Teste de Integração: Neste nível, são testadas unidades integradas, que podem corresponder a unidades de código, módulos, aplicações distintas, etc. Visa-se verificar se as interfaces funcionam corretamente e se os dados são processados de forma correta. Esse tipo de teste pode ser realizado pela equipe de testes, mas normalmente é realizado por desenvolvedores, e também predomina-se o uso de técnicas caixa-branca.

- Teste de Sistema: Neste nível, deve ser simulada a operação normal do sistema ou subsistema, sendo testadas suas funções de forma mais próxima possível do que irá ocorrer no ambiente do usuário. Enquanto os testes unitários e de integração são normalmente realizados por desenvolvedores, testes de sistema são executados, preferencialmente, em um ambiente operacional separado e controlado, com configuração idêntica ou similar ao ambiente de produção. Esse tipo de teste é normalmente realizado pela equipe de testes, utilizando-se técnicas caixa-preta. É tipicamente durante este estágio que são realizados os testes funcionais, alvo do presente trabalho. Além dos testes funcionais, são típicos desse estágio os testes de carga, de performance, de segurança, entre outros.

- Teste de Aceitação: Neste nível, o sistema é validado quanto a satisfazer as necessidades reais do cliente. É usualmente realizado por um grupo de usuários finais, que verificam se a solução atende aos objetivos de negócio e a seus requisitos, quanto à funcionalidade e usabilidade. São os testes finais de execução do sistema, antes de sua utilização no ambiente de produção.

2.3.3 Critérios de Teste

Um dos objetivos do teste é revelar falhas e, para isso, muitos critérios foram desenvolvidos. O princípio básico por trás de um critério é ser sistemático, a ponto de identificar um conjunto representativo de comportamentos dos programas (NETO, 2006).

Esses critérios são freqüentemente classificados como caixa-branca ou caixa-preta. Uma abordagem não é alternativa à outra. Ao contrário, são abordagens complementares, que trazem possibilidades de se descobrir diferentes classes de erro (NETO, 2006; PEZZÈ; YOUNG, 2008).

Nesta seção, são apresentadas as classificações de critérios: baseados em caixa-branca, baseados em caixa-preta e baseados em falhas.

2.3.3.1 Testes Caixa-Branca

Nesta abordagem de teste, o executor dos testes tem acesso à estrutura interna do software. Caminhos lógicos do software e as colaborações entre componentes são testados exercitando conjuntos específicos de condições ou ciclos. Também são utilizados os termos “teste estrutural” ou “teste caixa-de-vidro” (PRESSMAN, 2011).

2.3.3.2 Testes Caixa-Preta

Nesta abordagem de teste, o executor dos testes não tem acesso ao código fonte, portanto, o programa é visto como uma caixa-preta. O teste caixa-preta focaliza os requisitos funcionais do software e faz referência a testes realizados na sua interface. As técnicas de teste caixa-preta permitem derivar séries de condições de entrada que utilizarão completamente todos os requisitos funcionais para um programa. Também são utilizados os termos “teste comportamental” ou “teste baseado em especificação” (PRESSMAN, 2011).

No presente trabalho, utiliza-se a abordagem caixa-preta, uma vez que realiza-se testes baseado em especificações de casos de uso. Os principais critérios de teste caixa-preta e relacionados a este trabalho são: o particionamento de equivalência e a análise de valor limite.

No Particionamento de equivalência o domínio de entrada é subdividido em uma coleção de subdomínios, ou classes equivalentes, e um conjunto de testes representativos é extraído a partir de cada classe identificada. O comportamento do sistema para um valor de entrada contido em uma classe é equivalente ao comportamento para todos os outros valores dessa classe. Assim, se um caso de teste de uma classe de equivalência detecta um erro, todos os outros casos de teste dessa classe devem encontrar o mesmo erro (MYERS; SANDLER; BADGETT, 2011).

Na Análise de valor-limite os casos de teste são escolhidos próximo ou nas fronteiras dos domínios de entrada, uma vez que boa parte das falhas tende a se concentrar próximo a

esses valores. Normalmente, quando se aplica essa técnica, são testados: os limites exatos, o valor imediatamente acima do limite superior e o valor imediatamente abaixo do limite inferior (NETO, 2006; MYERS; SANDLER; BADGETT, 2011).

2.3.3.3 Testes Baseados em Falhas

Os critérios de teste baseados em falhas utilizam informação sobre os erros mais comumente detectadas no processo de desenvolvimento e sobre tipos específicos de erros que se deseja revelar (NETO, 2006). O conceito básico desse critério de teste é selecionar casos de teste capazes de distinguir o programa que está sendo testado de programas alternativos que contêm falhas hipotéticas. Isso geralmente é feito por meio da modificação do programa em teste, produzindo, de fato, programas com as falhas hipotéticas (PEZZÈ; YOUNG, 2008).

A Análise de mutantes, forma mais comum de teste baseado em falhas, é um critério que utiliza um conjunto de programas intencionalmente modificados (mutantes), obtidos a partir do programa original P, para avaliar o quanto um conjunto de testes T é adequado ao programa P. O objetivo é avaliar a habilidade de um conjunto de testes T em detectar todas as diferenças de comportamento entre P e seus mutantes. O Quadro 5 apresenta um exemplo de um programa original e três mutantes. Nota-se que a mutação gera programas diferentes, que deveriam ser detectados por casos de teste que testam o programa original. Caso um mutante gerado tenha um comportamento idêntico ao programa original, diz-se que é um mutante equivalente. Se o conjunto de testes T detecta a diferença existente no mutante, diz-se que T matou o mutante (NETO, 2006).

Quadro 5 – Exemplo de mutação

Programa	**Mutante1**
1. if (x > 0) 2. doThis(); 3. if (x > 10) 4. doThat();	1. if (x < 0) 2. doThis(); 3. if (x > 10) 4. doThat();
Mutante2	**Mutante3**
1. if (x > 0) 2. doThis(); 3. if (x < 10) 4. doThat();	1. if (x >= 0) 2. doThis(); 3. if (x > 10) 4. doThat();

Fonte: Neto (2006)

Em Gutiérrez et al. (2008) é apresentada uma lista de operadores mutantes para casos de uso (Quadro 6), obtidos do modelo de falhas de caso de uso introduzido por (BINDER, 2000). Em Pinto e Staa (2013) e Gutiérrez et al. (2008), utiliza-se abordagens com emprego desses operadores, que descrevem comportamentos diferentes para o SST, a partir da especificação

original. O objetivo não é testar o código do sistema em si, mas gerar mudanças no seu comportamento que possam ser observadas por testes funcionais (PINTO; STAA, 2013).

No presente trabalho, utiliza-se o critério de análise de mutantes para avaliar a efetividade dos casos de teste gerados pela abordagem proposta.

Quadro 6 – Operadores mutantes para casos de uso. (GUTIÉRREZ et al., 2008)

#	Operador Mutante
1	Troca de operador lógico da condição de uma alternativa ou a substituição de um passo com erro.
2	Troca da condição de uma alternativa ou de um passo com erro sempre avaliado como verdadeiro ou falso.
3	Um passo terminado de repente.
4	A remoção de um passo.
5	A adição de um passo.
6	Substituição de regras de validação ou da admissão de um dado como incorreto.
7	Informação incorreta ou incompleta mostrada pelo sistema.
8	Uma operação que pode falhar, mas sempre funciona corretamente.
9	Uma operação com falha sem ter um passo incorreto.
10	A informação mostrada para o ator tem menos elementos.
11	Substituição do executor do passo.

Fonte: Pinto e Staa (2013)

2.3.4 Tipos de Teste

Nesta seção são apresentados os tipos de teste descritos em RUP (2014). O RUP (*Rational Unified Process*) categoriza testes sob as cinco dimensões de qualidade do modelo FURPS: *Functionality* (Funcionalidade), *Usability* (Usabilidade), *Reliability* (Confiabilidade), *Performance* (Desempenho) e *Supportability* (Suportabilidade). Para cada uma dessas dimensões de qualidade, existe um ou mais tipos de teste associado(s) (Quadro 7). Os tipos de teste são descritos a seguir, destacando-se, ao final, os testes funcionais e os de regressão, ambos relacionados a este trabalho.

O Teste de Função objetiva validar as funções do sistema conforme o esperado, o Teste de Segurança, garantir que os dados(ou sistemas) possam ser acessados apenas por determinados atores, enquanto o Teste de Volume, verificar o comportamento do software quando submetido a grandes volumes de dados. Testes de Usabilidade enfatizam, entre outros, estética, consistência na interface do usuário, ajuda on-line e documentação do usuário.

Testes de Integridade destinam-se a avaliar a resistência do software a falhas e a compatibilidade técnica em relação à linguagem, sintaxe e utilização de recursos, enquanto os Testes de Stress, a avaliar como o sistema responde em condições anormais, como cargas de trabalho extremas, memória insuficiente, hardware e serviços indisponíveis ou recursos compartilhados

Quadro 7 – Tipos de Teste por dimensão de qualidade (RUP, 2014)

Dimensão	Tipos de Teste
Funcionalidade	Teste de Função Teste de Segurança Teste de Volume
Usabilidade	Teste de Usabilidade
Confiabilidade	Teste de Integridade Teste de Estrutura Teste de Stress
Desempenho	Teste de Avaliação de Desempenho Teste de Contenção Teste de Carga Teste de Perfil de Desempenho
Suportabilidade	Teste de Configuração Teste de Instalação

Fonte: Grady (1992)

limitados. Já o Teste de Estrutura é realizado, em geral, em aplicativos habilitados para a Web, garantindo que todos os links estejam conectados.

O Teste de Avaliação de Desempenho visa comparar o desempenho do software mediante uma carga de trabalho conhecida, enquanto o Teste de Carga, avaliar os limites operacionais de um sistema mediante cargas de trabalho variáveis. Já o Teste de Perfil de Desempenho destina-se a identificar gargalos de desempenho e processos ineficientes, a partir do monitoramento do software, incluindo fluxo de execução, acesso a dados e chamadas de funções, enquanto o Teste de Contenção visa verificar se o software lida de forma aceitável com as demandas de vários atores ao mesmo recurso (registros de dados, memória, etc.).

O Teste de Configuração visa garantir que o software funciona de maneira aceitável quando submetido a diferentes configurações de hardware ou software, enquanto o Teste de Instalação, que o sistema seja instalado conforme o esperado, mesmo em condições anormais, como espaço insuficiente em disco e interrupção de energia.

2.3.4.1 Teste Funcional

Teste funcional é um processo que busca encontrar discrepâncias entre o programa e sua especificação externa. Uma especificação “externa” é uma descrição precisa do comportamento do programa, do ponto de vista do usuário (MYERS; SANDLER; BADGETT, 2011). Segundo RUP (2014), testes funcionais podem ser implementados e executados em diferentes níveis de teste (unidade, integração, ou sistema). O presente trabalho propõe um ambiente para a realização de testes funcionais.

2.3.4.2 Teste de Regressão

Quando se cria uma nova versão de um sistema (removendo falhas, alterando ou acrescentando funcionalidades), podem ser inseridas, de maneira não intencional, defeitos (e consequentemente falhas) em funcionalidades já homologadas (os chamados efeitos colaterais). Quando isso ocorre, diz-se que a nova versão regrediu em relação às versões anteriores. Um teste de (não) regressão corresponde à re-execução de um teste previamente realizado com sucesso, sob um conjunto de funcionalidades do sistema, a fim de assegurar que seu funcionamento permanece correto (PEZZÈ; YOUNG, 2008).

2.3.5 Automação de Testes

Nesta seção, são apresentados alguns conceitos sobre testes de software automatizados, contextualizando a solução desenvolvida. Aborda-se a relação entre testes manuais e automatizados, as técnicas existentes para a construção de *scripts* de teste, as ferramentas de Captura e Repetição e suas limitações, e finaliza-se com a apresentação da ferramenta Selenium WebDriver.

2.3.5.1 Testes Manuais x Testes Automatizados

Para automatizar a execução de testes, é necessária a criação de programas que implementem os casos de teste, denominados *scripts* de teste. Como todo software, os *scripts* de teste demandam manutenção. Sempre que o sistema sob teste é alterado, os *scripts* devem ser verificados e modificados, se necessário. Por isso tende-se a selecionar, para a automatização de testes, sistemas que não mudem muito frequentemente. O esforço para manter os *scripts* de teste atualizados leva, ao abandono, muitas iniciativas de automatização de testes (FEWSTER; GRAHAM, 1999; RIOS; FILHO, 2006).

Apesar do esforço adicional em relação aos testes manuais, a construção e manutenção de testes automatizados busca se justificar, principalmente, pela economia gerada após sucessivas execuções dos *scripts* criados. Normalmente, o tempo necessário para a execução de uma suíte de teste automatizada é insignificante em relação ao tempo de realização dos testes manuais equivalentes. Isso permite a execução de mais testes, e mais frequentemente, o que favorece a realização de testes de regressão, descritos na Subseção 2.3.4.2. Além disso, a automatização possibilita a realização de testes que seriam difíceis ou impossíveis de se realizar manualmente, como a simulação de centenas de acessos simultâneos a um sistema (FEWSTER; GRAHAM, 1999; RIOS; FILHO, 2006).

2.3.5.2 Técnicas para Criação de *Scripts* de Teste

Em Fewster e Graham (1999), são apresentadas as seguintes técnicas para a construção de *scripts*:

- ***Scripts* Lineares**: *Scripts* lineares são aqueles que implementam suas instruções seqüencialmente, não havendo, portanto, comandos de seleção ou interação. Normalmente são criados utilizando-se ferramentas de Captura e Repetição (Subseção 2.3.5.3), que implementam somente esse tipo de *script*. *Scripts* lineares não são compartilhados para compor outros testes e apresentam entradas e comparações *hard-coded*, tendo alto custo para serem mantidos ;

- ***Scripts* Estruturados**: *Scripts* estruturados implementam estruturas de controle (de seleção/*ifs* ou iteração/*loops*) ou estrutura para chamada a outros *scripts*, sendo, portanto, *scripts* programados, mesmo que parcialmente. Essa técnica pode ser usada para dividir *scripts* em *scripts* menores e mais gerenciáveis, favorecendo o seu reúso ;

- ***Scripts* Compartilhados**: Essa técnica visa identificar tarefas repetitivas que possam ser reutilizadas por mais de um caso de teste. Criando *scripts* para tais tarefas, obtém-se melhoria no seu reúso ;

- **"*Data-driven*"**: Essa técnica consiste em eliminar dados de teste do corpo dos *scripts*, inserindo-os em arquivos de dados independentes ou tabelas. Com essa independência entre o código dos *scripts* e os dados de teste, possibilita-se que o *script* seja reutilizado para vários conjuntos de dados de entrada. Além disso, novos testes podem ser adicionados sem alteração no *script* correspondente ;

- **"*Keyword-driven*"**: Essa técnica propõe separar o desenvolvimento dos casos de teste (que descrevem o que fazer) do desenvolvimento dos *scripts* de teste (que descrevem como fazer). Isso possibilita que colaboradores com diferentes perfis trabalhem nas tarefas em que têm domínio, ou seja, testadores podem concentrar-se na criação dos arquivos de teste, enquanto programadores concentram-se nos *scripts* de suporte. A ligação entre os testes e os *scripts* de suporte se dá pelas *keywords*, que representam ações, de maior ou menor complexidade, sobre a interface do sistema. O uso da técnica "*Keyword-driven*" diminui a complexidade da criação de testes automatizados, pois permite que os testes sejam especificados de maneira menos detalhada e menos técnica.

Além dessas, há a **técnica baseada em modelo**, que vai além das propostas anteriores, criando testes automatizados de um modo semi-inteligente. Testes Baseados em Modelo (TBM) (2.3.7) derivam casos de teste inteira ou parcialmente a partir de um modelo, que descreve aspectos de um sistema (GARCíA; DUEÑAS, 2011).

De acordo com o Automated Testing Institute (ATI, 2014), há três gerações de *frameworks* de testes automatizados de software. A 1ª geração compreende o uso de *scripts* lineares, tipicamente por meio de ferramentas de Captura e Repetição. A 2ª geração compreende *frameworks* que utilizam dois tipos de técnicas: a decomposição funcional e a *data-driven*. Já a 3ª geração,

inclui *frameworks* que abordam as técnicas *keyword-driven* e a baseada em modelo (GARCíA; DUEÑAS, 2011).

O presente trabalho apresenta uma abordagem que gera *scripts* de teste a partir de um modelo, constituído de descrições de casos de uso, interfaces de usuário e dicionário de dados. Os *scripts* são gerados com independência dos dados de teste, e a cada execução podem ser utilizados valores diferentes, que estejam de acordo com a especifição de valores permitidos para os dados de testes. Além disso, os *scripts* são criados como especificações de casos de uso, que descrevem como os testes devem ser feitos em alto nível, sendo mantidos, ocultos, detalhes de como os mesmos são realizados. Ademais, os *scripts* seguem a estrutura dos fluxos dos casos de uso, que podem ser compartilhados e invocados por outros. Assim, os *scripts* gerados no ambiente proposto utilizam as técnicas de TBM, *keyword-driven*, *data-driven* e de *scripts* compartilhados.

2.3.5.3 Ferramentas de Captura e Repetição

Ferramentas de Captura e Repetição (como a Selenium IDE[1]) oferecem a capacidade de geração automática de *scripts* de testes funcionais. Nesta técnica, as ações do usuário sobre a interface do sistema são gravadas pela ferramenta, que gera um código, reproduzindo essas ações. O usuário adiciona oráculos nos momentos em que devam ser realizadas verificações nos resultados apresentados pelo sistema.

A grande vantagem em se utilizar ferramentas de Captura e Repetição é o fato de se criar testes automatizados de forma simples, sem a necessidade de conhecimentos em programação. Entretanto, o uso dessa técnica limita o reúso e a manutenção dos *scripts*, uma vez que os *scripts* gravados são lineares, apresentando em sua estrutura dados *hard-coded* e comandos desnecessários (Subseção 2.3.5.2). Devido ao fato do código gerado automaticamente ser um código de difícil entendimento, quando há necessidade de manutenção nos testes, opta-se, geralmente, em realizar novamente as execuções manuais, gerando um novo código, em vez de se alterar o existente. Dependendo do número de execuções a serem refeitas, esse processo pode tornar-se oneroso.

2.3.5.4 Selenium WebDriver

Selenium WebDriver[2] é uma API gratuita e de domínio público para a automatização de testes de aplicações *web*. As funções disponibilizadas via API realizam a manipulação de elementos de interface *web*, e sua execução é feita por meio de chamadas diretas ao navegador, utilizando o suporte nativo para automação, próprio de cada navegador.

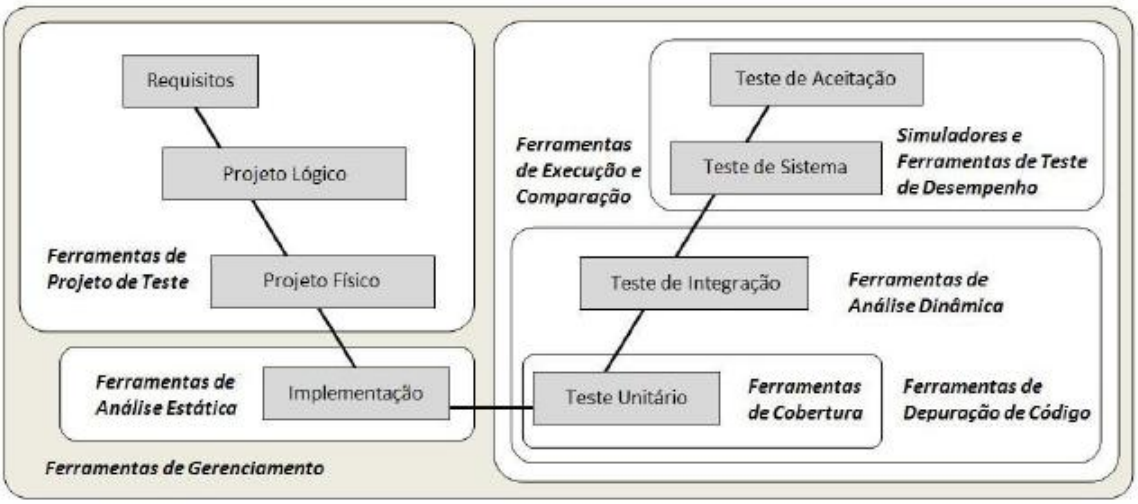
[1] Selenium IDE: <http://www.seleniumhq.org>
[2] Selenium WebDriver: <http://www.seleniumhq.org>

O presente trabalho apresenta uma abordagem que utiliza e integra-se à ferrramenta Selenium WebDriver, para a interação com os elementos da interface *web*. A abordagem gera código na linguagem reconhecida pelo Selenium e o invoca, para a realização dos testes.

2.3.6 Ferramentas de Suporte aos Testes

Ferramentas de suporte a testes estão disponíveis em cada fase do ciclo de desenvolvimento do software, como ilustrado na Figura 5. A seguir tem-se a descrição de cada tipo de ferramenta. (FEWSTER; GRAHAM, 1999).

Figura 5 – Tipos de Ferramentas de Suporte a Testes



Fonte: tradução livre de Fewster e Graham (1999), p. 7

Ferramentas de Projeto Lógico são utilizadas para gerar casos de teste a partir da especificação, enquanto Ferramentas de Projeto Físico, para manipular dados existentes ou gerar dados de teste. Ferramentas de Análise Estática são utilizadas na análise do código-fonte para detecção de defeitos, enquanto Ferramentas de Análise Dinâmica, para avaliar o comportamento do sistema enquanto o software está sendo executado, por exemplo, para detectar *leaks* de memória. Já Ferramentas de Depuração de Código são utilizadas na depuração de código, atividade comum no desenvolvimento, e também útil nos testes, especialmente quando se tenta isolar um defeito.

Simuladores possibilitam testes de partes de um sistema, de uma forma que não seria possível na vida real. Ferramentas de Execução e Comparação são utilizadas para automatizar a execução de testes, comparando as saídas do teste com as saídas esperadas (Ferramentas de Captura e Repetição são ferramentas desse tipo). Já as Ferramentas de Teste de Desempenho são utilizadas para medir o tempo de resposta do software, podendo ser usadas para testes de Volume ou de Estresse. As Ferramentas de Cobertura são utilizadas para avaliar o quanto o SST tem sido exercitado por um conjunto de testes. Por fim, Ferramentas de Gerenciamento suportam o plano de testes, possibilitando manter a relação dos testes que foram executados. Essa categoria

também inclui ferramentas para ajudar na rastreabilidade de testes e ferramentas de rastreamento de defeitos.

O uso adequado de ferramentas de testes aumentam a eficiência e eficácia do processo de testes. Nesse sentido, ferramentas de automação são especialmente importantes para testes funcionais e de regressão (RIOS; FILHO, 2006).

A ferramenta criada durante esta pesquisa enquadra-se como ferramenta de projeto lógico e de projeto físico. O ambiente proposto neste trabalho envolve e integra também ferramentas de gerenciamento e de execução e comparação.

2.3.7 Teste Baseado em Modelos

O Teste Baseado em Modelos (TBM) consiste em uma técnica para geração automática de casos de testes, com entradas e saídas esperadas, utilizando modelos extraídos a partir dos requisitos do sistema. Para que seja possível utilizar o TBM, é necessário que a especificação do software seja definida utilizando modelos em um formato apropriado para a automação das atividades de teste, como modelos representados utilizando métodos formais, máquinas de estado finito e a UML (NETO, 2006).

Com a possibilidade de se gerar casos de teste a partir da especificação do software, utilizando procedimentos automáticos que podem ser menos suscetíveis a erros, o TBM surge como uma abordagem aplicável para controlar a qualidade do software, bem como reduzir os custos associados ao processo de testes. Assim, a adoção do TBM pode gerar melhorias nas atividades de teste por meio da simplificação do seu planejamento, da semi-automação de suas atividades, e da possibilidade de re-execução automática após modificações (NETO, 2006).

Na abordagem proposta neste trabalho, casos de teste – e posteriormente testes executáveis – são gerados a partir da descrição de casos de uso, interfaces de usuário e dicionário de dados. Essas especificações são cadastradas em um ambiente específico para esse fim – um editor com sintaxe gerada pelas regras de uma DSL, desenvolvida para esse trabalho. Devido às validações nesse ambiente ocorrerem em tempo de edição, o autor pode refinar o caso de uso interativamente. A padronização na escrita dos casos de uso é auxiliada e guiada pelo ambiente, melhorando sua qualidade e facilitando a identificação de seus componentes, que podem então ser utilizados para as transformações automáticas posteriores.

2.3.8 Geração de Casos de Teste a partir de Casos de Uso

Uma vez que o objetivo de um caso de teste é exercitar um caminho particular do programa ou verificar o cumprimento de um requisito específico, cenários de casos de uso podem ser utilizados como ponto de partida para a geração dos casos de teste, pois determinam os

possíveis caminhos a percorrer. Em Heumann (2001), é sugerido um processo para derivação de casos de teste a partir dos casos de uso, composto de três passos:

1. Para cada caso de uso, gere um conjunto completo de cenários;

2. Para cada cenário, identifique pelo menos um caso de teste e as condições que o tornarão executável;

3. Para cada caso de teste, identifique os valores dos dados com os quais testá-lo.

A geração de cenários para os casos de uso, apontada no passo 1, foi descrita na Subseção 2.1.4. No passo 2, são identificadas e analisadas as entradas dos casos de teste e seus possíveis resultados. A identificação de mais de um caso de teste para um cenário ocorre, por exemplo, quando a condição de acionamento de um fluxo alternativo envolve mais de uma possibilidade (ex: "3a Aluno suspenso ou com matrícula trancada"). Além disso, podem ser gerados mais de um caso de teste para um cenário, por exemplo, quando se utiliza a técnica de análise de valor limite. Já no passo 3, são indicados valores reais para os dados de entrada. Sem dados de teste, casos de teste (ou procedimentos de teste) não podem ser implementados, executados ou verificados, pois são apenas descrições de condições e cenários. Portanto, é necessário identificar os valores reais a serem utilizados na execução dos testes e na verificação dos resultados (HEUMANN, 2001).

2.4 Xtext

Xtext[3] é um *framework open-source* baseado no openArchitectureWare[4], Eclipse Modeling Framework[5] e ANTLR[6]. Seu uso permite desenvolver linguagens de domínio específico (DSLs) e obter um editor de texto correspondente à linguagem, baseado no IDE Eclipse[7], com todo o suporte característico, incluindo suporte a validação, conclusão de código, formatação, coloração de sintaxe, visão estrutural dos elementos e referência cruzada (SAVIC et al., 2011).

O Xtext utiliza, em suas gramáticas, uma notação estendida de Backus-Naur (Extended Backus-Naur Form), baseada na notação de (BRAY et al., 2014), chamada neste trabalho simplesmente de BNF. Cada regra na gramática define um símbolo na forma **<símbolo> : expressão;**[8] e uma expressão pode ser formada pelos seguintes elementos:

[3] Xtext: <http://www.eclipse.org/Xtext>
[4] oAW: OpenArchitectureWare. <http://www.openarchitectureware.org>
[5] EMF: Eclipse Modeling Framework. <http://www.eclipse.org/modeling/emf>
[6] ANTLR: Another Tool for Language Recognition. <http://www.antlr.org/>
[7] Eclipse: <http://www.eclipse.org/>
[8] Neste ponto há uma diferença entre a notação utilizada e a apontada na referência, onde define-se a regra como <símbolo> ::= expressão;

- Outro símbolo;
- Cadeias de caracteres (strings) literais, entre apóstrofos ou aspas;
- Os símbolos terminais[9] STRING, INT e ID, representando respectivamente: uma string, um número inteiro e um identificador (iniciado por letra e seguido por letras ou dígitos).

Expressões podem ser combinadas para gerar outros padrões mais complexos, como é apresentado no Quadro 8, onde A e B representam expressões simples.

Quadro 8 – Significado de algumas combinações de elementos nas expressões

Expressão	Significado
A?	Gera A ou nada; A é opcional
A+	Gera uma ou mais ocorrências de A
A*	Gera zero ou mais ocorrências de A
A B	Gera A seguido de B
A | B	Gera A ou B

Fonte: Bray et al. (2014)

A gramática BNF é utilizada para a descrição concreta da linguagem. Baseado na gramática, o *framework* gera um meta-modelo (implementado em EMF) que descreve a sintaxe abstrata da linguagem, além de um parser e um editor de texto específico para a linguagem.

Quadro 9 – Exemplo de gramática Xtext

```
Modelo:
  (tipos+=Tipo)*;
Tipo:
  TipoDeDado | Entidade;
TipoDeDado:
 "tipoDeDado"  name=ID;
Entidade:
 "entidade"  name=ID "{"
    (caracteristicas+=Caracteristica)*
  "}";
Caracteristica:
  tipoDaCaracteristica=[Tipo]  name=ID
;
```

Fonte: Xtext (2014)

Os *parsers* Xtext criam objetos do modelo em memória, enquanto consomem o texto informado no editor. Cada símbolo da gramática resulta em um objeto, cujas propriedades são geradas em pontos onde há atribuições (SAVIC et al., 2011). O Quadro 9 exibe um exemplo de gramática Xtext. Durante a leitura de um texto baseado na gramática apresentada, objetos

[9] Símbolos terminais disponibilizados para gramáticas Xtext

TipoDeDado, **Entidade** e **Caracteristica** serão instanciados. Note que a última propriedade (tipoDaCaracteristica) é atribuída de forma diferente das demais: ela recebe uma referência para um objeto. No caso, a um objeto **Tipo**. Isso significa que, nesse ponto, deve ser indicado um objeto **Tipo**, já declarado, configurando um ponto de referência cruzada.

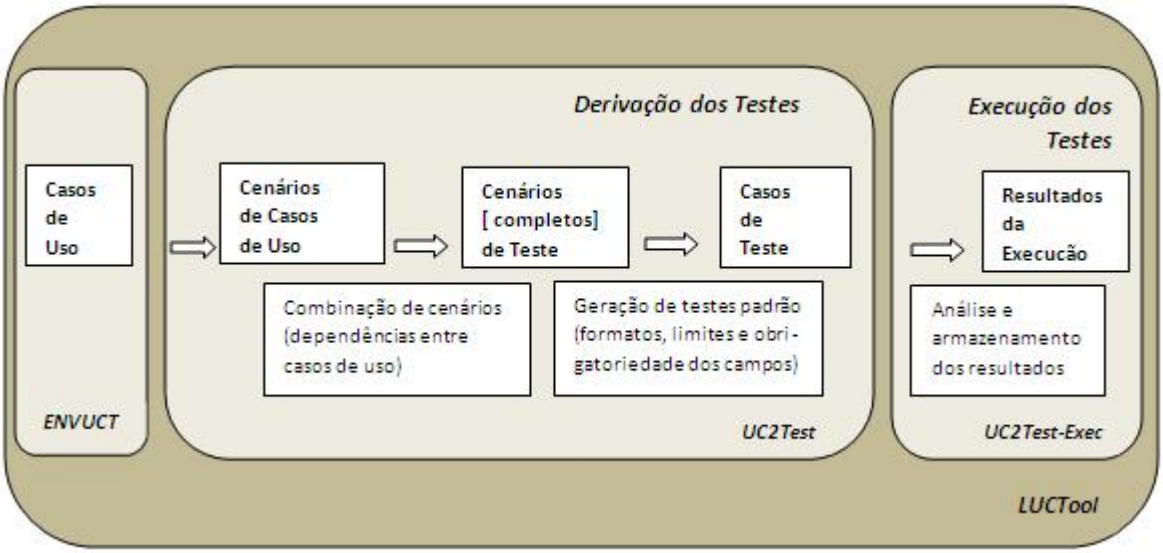
Assim, além dos elementos citados quando se apresentou a notação BNF, podem ser utilizados, nas gramáticas Xtext, atribuições – que permitem designar um resultado a uma propriedade do objeto que está sendo gerado pela expressão – e atribuições de referência cruzada – que atribuem uma referência a uma propriedade do objeto gerado pela expressão.

A utilização de Xtext, em que as gramáticas são expressas como meta-modelos, permite tratar os elementos da linguagem como partes de um modelo, que podem ser usados para transformações automáticas, posteriormente (SAVIC et al., 2011).

3 ABORDAGEM PROPOSTA

Este trabalho propõe um ambiente para especificação de casos de uso, sua derivação em cenários de teste e posterior transformação em casos de teste executáveis, que, por fim, são convertidos em *scripts* de testes funcionais automatizados. A geração de cenários combina cenários de casos de uso por meio de suas relações de dependência, enquanto a geração de casos de teste envolve estratégias que possibilitam explorar várias situações, como o uso de valores limítrofes, valores aleatórios válidos, inválidos e a ausência de valores obrigatórios (Figura 6).

Figura 6 – Etapas do processo - Visão geral

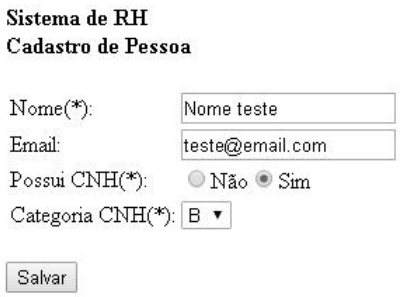


Exercitar todos esses cenários de forma automática simplifica o processo de geração e execução de testes, cobrindo um grande número de casos práticos, que seriam trabalhosos se feitos manualmente. E uma vez que testes são obtidos a partir da documentação do sistema, proporciona-se maior utilidade a esses artefatos, incentivando sua criação e manutenção.

Para efeito de demonstração, será utilizado, no decorrer deste capítulo, um caso de uso para cadastro de pessoas, cuja interface é exibida na Figura 7. Esse exemplo simples será utilizado para descrever as etapas de especificação, geração e execução de testes.

Este capítulo está organizado da seguinte forma: a Seção 3.1 apresenta a metodologia utilizada. A Seção 3.2 apresenta a etapa de especificação do sistema, de acordo com as regras da DSL LUCT. A Seção 3.3 apresenta as etapas para a derivação de testes a partir da especificação. Por fim, a Seção 3.4 apresenta as etapas de execução dos testes gerados, visualização e gerenciamento dos resultados de execução.

Figura 7 – Tela de Cadastro de pessoa



3.1 Metodologia

Para o presente trabalho, desenvolveu-se a DSL LUCT (*Language of Use Cases to Test*), com a finalidade de descrever casos de uso, interfaces de usuário e dicionário de dados.

Casos de uso podem ser descritos em nível de detalhamento variado, de acordo com a etapa do projeto. Em Larman (2007), são apresentados três níveis de detalhamento para a escrita de casos de uso e é sugerido que se escreva em estilo essencial principalmente no início da especificação – quando deve-se focar na intenção, e não na interface – e no estilo concreto/detalhado, nas etapas seguintes, em que se tem o projeto da IU (interface do usuário).

No presente trabalho, propõe-se que cada passo do caso de uso no nível abstrato seja seguido por seu detalhamento, onde são inseridos detalhes da interface. Detalhando os passos dos casos de uso com informações da interface, *scripts* de teste podem ser gerados. A utilização de um ambiente em que os casos de usos são mantidos interligados aos testes, auxilia para que alterações no primeiro possam ser refletidas no segundo. Mesmo que essas alterações não sejam realizadas automaticamente em sua totalidade, passam a ser motivadas pela ligação dos passos dos casos de uso com seu detalhamento e avisos decorrentes das validações. Além disso, os casos de uso geram informações úteis para *debbuging*, quando um teste falha.

Assim como em Cockburn (2005), os casos de uso foram classificados de acordo com o nível de seu objetivo: de Resumo, de Usuário ou Subfunção. No presente trabalho, que visa a realização de testes funcionais em sistemas de software, são tratados principalmente casos de uso do nível de Objetivo de Usuário. Entretanto, é incentivada a especificação dos casos de uso de nível Subfunção – para favorecer a reutilização e manutenção – e os de nível Resumo – para orquestrar e otimizar as chamadas aos de nível Objetivo do Usuário. O sequenciamento e a reutilização de casos de uso é possibilitada pela invocação de um caso de uso por outro, recurso disponível tanto nos passos dos fluxos, quanto nas pré-condições.

Além dos casos de uso, devem ser descritos as interfaces de usuário – com seus elementos e menus – e um dicionário de dados – com formatos e valores aceitáveis para os dados do sistema.

A geração dos *scripts* de teste deve ser realizada sempre que for feita uma alteração na especificação. Para a geração dos testes, é necessário, primeiramente, gerar os cenários de teste, percorrendo as possibilidades de caminhos, nos fluxos principal e alternativos.

Dentre as várias opções para se gerar os caminhos de um caso de uso, como transformação em um grafo ou em uma FSM (máquina de estados finitos), optou-se por tratar o caso de uso como um diagrama de atividades, por ser um diagrama de fácil entendimento para o usuário, auxiliando-o a visualizar e verificar a completeza e a correção dos caminhos do caso de uso.

Os *scripts* gerados refletem a estrutura dos casos de uso, o que, em situações de erros, enriquece as informações das evidências. Os valores de entrada dos testes são gerados em tempo de execução, o que permite sua variação em execuções diferentes. Para a interação com a interface de usuário, os *scripts* utilizam o Selenium WebDriver, API que, além de gratuita e de domínio público, é madura, robusta e conceituada na automatização de testes para aplicações *web*.

Os resultados dos testes são armazenados no ambiente TestLink[1], por ser uma ferramenta gratuita e completa para o gerenciamento de testes, que permite centralizar tanto testes automatizados quanto manuais, manter o histórico de execuções (com suas evidências) e obter relatórios gerenciais customizáveis.

3.2 Especificação do Sistema

A especificação do sistema envolve a descrição dos casos de uso, interfaces de usuário e dicionário de dados. As Subseções 3.2.1, 3.2.2 e 3.2.3 descrevem o formato da especificação, que segue as regras definidas na DSL proposta (LUCT). A descrição completa da DSL encontra-se no Apêndice A. Conforme mencionado, os trechos de especificação aqui apresentados referem-se ao caso de uso `CadastrarPessoa`, cuja interface foi apresentada na Figura 7.

3.2.1 Descrição dos Casos de Uso

A Listagem 1 exibe trecho da DSL LUCT que trata da definição de casos de uso e a Listagem 2, um exemplo de especificação. O significado de cada elemento que compõe um caso de uso pode ser obtido na Seção 2.1.3. A seguir descreve-se como os elementos devem ser informados e como são utilizados na geração dos testes.

Listagem 1 – Trecho da DSL LUCT para definição de casos de uso

```
1 CasoDeUso =
2   "CasoDeUso"  ID  [NivelObjetivo]  "{"
3   ["contexto"  STRING ]
4   ["ator"  ID ]
5   ........
6   ["escopo"  STRING ]
```

[1] TestLink: <http://testlink.org/>

```
7    ["stakeholders"  STRING ]
8    ........
9    ["listadevariacoes"  STRING ]
10   ["informacaorelacionada"  STRING ]
11   "}"
12 ;
13 NivelObjetivo = ( "(+)" | "(!)" | "(-)" ) ;
```

Listagem 2 – Exemplo de especificação de caso de uso

```
1 CasoDeUso CadastrarPessoa (!){
2   contexto "Cadastro de pessoa"
3   .......
4   ator Secretaria
5   .......
```

A especificação indica um `ID`, identificação pela qual o caso de uso poderá ser referenciado. Informa-se seu `NivelObjetivo` (Resumo "+", Objetivo do Usuário "!" ou Subfunção "-"), que indica possibilidades de invocação, uma vez que casos de uso podem ser invocados por outros de mesmo nível de objetivo ou de níveis de objetivo superiores. `Contexto` é uma descrição do caso de uso e `Ator` indica o perfil que deve realizar o teste. Os demais elementos (`Escopo`, `Stakeholders`, `ListaDeVariacoes` e `InformacaoRelacionada`), assim como `Contexto`, são de caráter informativo, não sendo utilizados na geração de testes.

A Listagem 3 apresenta a definição de quatro elementos utilizados no controle da geração e execução dos testes: `PalavrasChave` e `Prioridade`, que classificam os casos de uso, servindo de filtro na execução dos testes; `ExecuçãoPrópria`, que indica se devem ser gerados casos de teste para o caso de uso, ou se ele é executado somente como parte de outros casos de uso e o `Tipo: Login`, que caracteriza o caso de uso como tal, resultando na sua execução antes dos demais, autenticando o `Ator` indicado. A Listagem 4 exibe um exemplo de especificação.

Listagem 3 – Trecho da DSL LUCT para definição de elementos de controle

```
1   ["palavraschave"  STRING ]
2   ["prioridade"  INT ]
3   ["execucaopropria"  ("S"|"N") ]
4   ["tipo"  TipoCasoUsoLogin ]
```

Listagem 4 – Exemplo de especificação – Elementos de controle

```
1   palavraschave "pessoa"
2   prioridade 3
3   execucaopropria S
```

O `CenárioPrincipal` e `Extensões` apresentam, nos seus fluxos de evento, a base para a geração dos testes. Sua especificação segue o formato "Inteiramente Completo", que utiliza passos numerados com uma convenção de números e letras para passos alternativos.

A Listagem 5 exibe trechos da DSL LUCT que tratam da definição do cenário principal e a Listagem 6, um exemplo de especificação. O elemento `PENDENTE`[2] indica a possibilidade de ser descrito um caso de uso informal – como um texto, em vez do formato estruturado. Casos de uso informais (assim como todos os pontos indicados como pendentes) devem ser reescritos – necessitando estar no formato completo – no momento da geração dos testes.

Listagem 5 – Trechos da DSL LUCT para definição do cenário principal

```
1 (("cenarioprincipal"  CenarioPrincipal) | (PENDENTE  STRING))
2 CenarioPrincipal = PassoUC  { ","  PassoUC }* ;
3 PassoUC = INT  STRING  "{"  Passo  { ","  Passo }*  "}" ;
4 Passo = Acao | Verificacao | Controle | PENDENTE ;
```

Listagem 6 – Exemplo de especificação – Cenário principal

```
1   cenarioprincipal
2     1 "Usuário informa dados da pessoa" {
3       Informo[nome, email]
4     },
5     2 "Usuário informa que pessoa não tem CNH" {
6       Informo possuiCNH com Pessoa.possuiCNH.nao
7     },
8     .....
```

Os fluxos são compostos por passos de casos de uso (`PassoUC`) que por sua vez são compostos por passos de detalhamento (`Passo`) de um dos super-tipos:

- Ação: Interações do usuário com a IU;
- Verificação: Asserções que validam as respostas do sistema;
- Controle: Funções de apoio para o sequenciamento de passos.

A lista completa dos tipos de passo disponíveis por cada super-tipo é apresentada no Quadro 14 (Apêndice A). A Listagem 7 exibe alguns trechos da DSL que tratam dessa definição e a Listagem 8, exemplos de especificação. Vale destacar que o tipo de passo `InformarCampo` tem a indicação de um `Valor` para preenchimento do campo como opcional, devido ao fato da abordagem trabalhar com geração de valores dinâmicos para os campos, a partir de regras de valores válidos atreladas aos mesmos. Normalmente indica-se um `Valor` quando estão sendo exercitadas situações alternativas específicas, ou em fluxos relacionados a classes de equivalência, em que cria-se um fluxo alternativo para cada situação a ser testada. A Listagem 6 exibe um exemplo dessa situação (linha 6).

[2] O elemento PENDENTE pode ser utilizado em vários pontos em que ainda não se tem todas as informações detalhadas, permitindo que a especificação possa ser feita de forma incremental.

Listagem 7 – Trechos da DSL LUCT para definição de passos

```
Acao = ( Acessar | Informar | Clicar | Aguardar | Executar | ResponderAlerta ) ;
Acessar = "Acesso"  ("menu"  ID | "link"  ID) ;
InformarCampo = "Informo"  ID  [ "com"  Valor ]  ;
Executar = "Executo"  ID  [ PreExecucaoParms ]  [ "com"  "extensao"  "="  STRING ] ;

Controle = ArmazenarValor | AcesseiTela;
ArmazenarValor = "Armazeno"  ( Valor | ValorTela | "campo" ID )  "como"  ID ;

Verificacao = ( VerificacaoNaTela | VerificacaoDeMsgErroTestesPadrao ) ;
VerificacaoNaTelaPrincipal = "Visualizo "  ( "mensagem"  ID | "texto = "  STRING  |
    "valor armazenado" ID )
  [ ValorTela ] ;
```

Listagem 8 – Exemplos de especificação – Passos

```
Acesso link LinkTelaCadastroPessoa
Clico botaoOK
Visualizo texto = "Cadastro realizado com sucesso" no campo AreaMensagem
```

Os fluxos de extensão, assim como os fluxos de cenário principal, são compostos por um conjunto de passos. Além desses, fazem parte da sua definição: o passo do cenário principal em que o fluxo é iniciado, o passo em que o fluxo retorna ao cenário principal (quando for o caso), a informação do fluxo ser ou não de erro, a informação do fluxo ser ou não de erro em testes padrão (onde verifica-se automaticamente valores inválidos e campos obrigatórios não preenchidos) e a informação do fluxo estar relacionado a uma extensão de tela. A Listagem 9 exibe trechos da DSL LUCT relacionados à definição desses fluxos e a Listagem 10, um exemplo de especificação.

Listagem 9 – Trechos da DSL LUCT para definição de extensões

```
["extensoes"  Extensoes ]
Extensoes = CondicaoEPassosFluxoAlt  { ","  CondicaoEPassosFluxoAlt }* ;
CondicaoEPassosFluxoAlt = FluxoAlternativoCondicao
  { PassoUCAlt }+  [ PassoUCAltFinal ] ;
FluxoAlternativoCondicao = INT  LETRA
  [ ("erro" | "erro-teste-padrao") ]  STRING
  [ "extensão de tela"  ID ] ;
PassoUCAlt = INT  LETRA   INT  STRING
  [ "{"  Passo  { ","  Passo }*  "}"  ] ;
PassoUCAltExecutaFinal = Executa passo  INT ;
```

Listagem 10 – Exemplo de especificação – Extensões

```
  extensoes
    2a "Pessoa tem CNH" extensão de tela CamposCNH
    2a 1 "Usuário informa dados da CNH"{
      Informo possuiCNH com Pessoa.possuiCNH.sim,
      Informo categoriaCNH
    }
    2a 2 Executa passo 3,

```

```
9       4a erro-teste-padrao "Erros testes padrão"
10      4a 1 "Sistema exibe mensagem de erro de um dos testes padrão" {
11        Visualizo uma mensagem de erro de testes padrão no campo AreaMensagem
12      }
```

Por fim, a Listagem 11 exibe trechos da DSL LUCT que tratam da definição dos elementos `PréCondicao`, `Acionador`, e pós-condições (`GarantiasMínimas` e `GarantiasDe Sucesso`). Cada um desses elementos permite a indicação de um subconjunto de tipos de `Passo`: O elemento `PréCondicao` é composto por passos do tipo `Executar`, `Armazenar Valor`, `Acessar` ou o elemento de controle `AcesseiTela`. O elemento `Acionador`, por passos do tipo `Acessar` ou `Clicar`. E os elementos `GarantiasMínimas` e `Garantias DeSucesso`, por passos do tipo `ArmazenarValor` ou `Verificacao`. A Listagem 12 apresenta um exemplo de especificação.

Listagem 11 – Trechos da DSL LUCT para definição do acionador do caso de uso, pré e pós-condições

```
1 PassoPreCondicao = ( Executar | ArmazenarValor | Acessar | AcesseiTela | PENDENTE );
2 PassoAcionador = Acessar | Clicar | PENDENTE ;
3 PassoPos = ( ArmazenarValor | Verificacao| PENDENTE ) ;
```

Listagem 12 – Exemplo de especificação – Acionador

```
1   acionador {
2     Acesso link LinkTelaCadastroPessoa
3   }
```

A Listagem 13 exibe, agora de forma completa, o caso de uso `CadastrarPessoa`, que será utilizado como exemplo no decorrer deste capítulo.

Listagem 13 – Exemplo de especificação – Caso de uso `CadastrarPessoa`

```
1  CasoDeUso CadastrarPessoa (!){
2    contexto "Cadastro de pessoa"
3    palavraschave "pessoa"
4    prioridade 3
5    execucaopropria S
6    ator Secretaria
7    acionador {
8      Acesso link LinkTelaCadastroPessoa
9    }
10   cenarioprincipal
11     1 "Usuário informa dados da pessoa" {
12       Informo[nome, email]
13     },
14     2 "Usuário informa que pessoa não tem CNH" {
15       Informo possuiCNH com Pessoa.possuiCNH.nao
16     },
17     3 "Usuário submete solicitação de cadastro" {
18       Clico botaoOK ,
19     } ,
```

```
20      4 "Sistema exibe mensagem de cadastramento OK" {
21        Visualizo texto = "Cadastro realizado com sucesso" no campo AreaMensagem
22      }
23    extensoes
24      2a "Pessoa tem CNH" extensão de tela CamposCNH
25      2a 1 "Usuário informa dados da CNH"{
26        Informo possuiCNH com Pessoa.possuiCNH.sim,
27        Informo categoriaCNH
28      }
29      2a 2 Executa passo 3,
30
31      4a erro-teste-padrao "Erros testes padrão"
32      4a 1 "Sistema exibe mensagem de erro de um dos testes padrão" {
33        Visualizo uma mensagem de erro de testes padrão no campo AreaMensagem
34      }
35  }
```

3.2.2 Descrição das Interfaces de Usuário

3.2.2.1 Menus, Áreas e Mensagens Globais

A especificação de opções de menu ou *links* (com o papel de menu) é composta por um identificador interno (que será utilizado para referenciá-lo nos casos de uso), a identificação do menu na tela e a indicação da IU que é acessada através do mesmo.

Quando o SST apresenta um padrão na definição de áreas da tela para determinados fins - como para a exibição do usuário autenticado, para o título da IU atualmente em execução, ou para mensagens geradas pelo SST para confirmações, erros ou alertas - é útil a especificação dessas áreas (ou campos) globais, para utilização nos casos de uso. A forma de definição segue o formato de especificação de campos, explicado a seguir.

Já para a especificação de mensagens globais (mensagens padrão do sistema), deve-se informar um identificador interno (para indicação nos casos de uso), além do texto da mensagem. Já na especificação de mensagens de erro para testes padrão indica-se, além desses, a situação de erro (campo fora de formato, fora do limite especificado, etc.) em que a mensagem deve ser exibida.

Um detalhamento maior sobre o preenchimento dessas informações pode ser obtido no Apêndice A.

3.2.2.2 Interfaces de Usuário

A especificação de uma IU é composta pela lista de campos e mensagens relacionadas à interface (Listagem 14). Os campos podem ser declarados no nível da IU, ou em uma de suas extensões, que devem ser declaradas quando parte da IU é exibida (e deve ser preenchida) somente em situações específicas.

Listagem 14 – Trechos da DSL LUCT para definição de IUs

```
Tela = "Tela"  ID  "{"
  Campo  { ","  Campo }*
  {ExtensaoTela}*  [MensagensTela]  "}" ;
ExtensaoTela = "Extensao"  ID  "{"
  Campo  { ","  Campo }*  "}" ;
Campo = ["*"]  ID  ":"  TipoElementoHtml  IdentificacaoNaTela
 [ "descricao ="  STRING ]
 [ "valores ="  ValorDinamico ]
 [ "(carrega valores após"  ID  ")" ]
 [ "aciona tela"  ID ]
 [ "preenchimento forçado"]
 { "["  MensagemErroCampo  "]" }* ;
```

Para cada campo deve ser descrito: se o mesmo é obrigatório (indicado por '*'), seu identificador interno (que será utilizado para referenciá-lo nos casos de uso), seu tipo HTML e sua identificação na tela (por meio do rótulo a ele associado, sua propriedade *id* ou expressão XPath relacionada[3]). Entre as informações opcionais, encontram-se: uma descrição, a regra do dicionário de dados que gera os valores permitidos para o campo, a indicação de que esse campo depende ou não de outro para carregar seus valores válidos, a indicação de que esse campo funciona como um acionador de uma nova tela, a indicação de que esse campo tem sempre um valor preenchido e mensagens de erro específicas relacionadas ao campo (que não sejam cobertas pelas mensagens globais). Um detalhamento maior sobre o preenchimento dessas informações pode ser obtido no Apêndice A.

Vale destacar que os campos, textos dos menus e das mensagens são descritos somente na especificação das telas, sendo referenciados nos casos de uso por um identificador interno. Dessa forma, segue-se o padrão *Page Object* (PAGEOBJECT, 2014), e alterações que venham a ocorrer nos nomes e textos das interfaces do sistema precisam ser refletidas somente na especificação das IUs, o que facilita a atualização da especificação e dos testes.

A Listagem 15 exibe a especificação da IU `CadastrarPessoa`.

Listagem 15 – Exemplo de especificação – Interface de usuário

```
Tela TelaCadastrarPessoa {
  * nome: CAMPO_TEXTO label = "Nome" valores = Pessoa.nome ,
  email: CAMPO_TEXTO label = "Email" valores = Pessoa.email,
  * possuiCNH: CAMPO_RADIO label = "Possui CNH"  valores = Pessoa.possuiCNH,
  botaoOK: CAMPO_BOTAO label = "Salvar"

  Extensao CamposCNH  {
    * categoriaCNH: CAMPO_TEXTO label="Categoria CNH" valores=Pessoa.categoriaCNH,
  }
}
```

[3] XPath(XML Path Language): Linguagem para identificar partes de um documento XML. Uma das formas de identificação de elementos para automação de testes de software em plataforma *web*. http://www.w3.org/TR/xpath/

3.2.3 Descrição do Dicionário de Dados (Valores Válidos)

O dicionário de dados contém as regras para a geração de valores válidos para os dados de entrada. A Listagem 16 exibe trechos da DSL LUCT que tratam dessa definição. Optou-se por uma notação que assemelha-se à descrição de um diagrama de classes, embora o mapeamento de classes aqui possa ser elaborado pelo usuário de forma mais próxima a um modelo de domínio – com classes conceituais do mundo real – e não necessariamente com classes de software.

Listagem 16 – Trechos da DSL LUCT para definição do dicionário de dados

```
1 ClasseDicionario = "Classe"  ID   [ "ClasseBase"  ID ]  "{"
    PropriedadeDicionario  { ","  PropriedadeDicionario }*  "}" ;
2 PropriedadeDicionario = ID  ":"  ( ValoresValidos |
3          "[" ClassesEquivalenciaDicionario  "]"  | PENDENTE ) ;
4 ValoresValidos = ( TipoComLimiteInfSup | ListaDeValores | Regex | ExpressaoJS | SQL
    ) ;
5 TipoComLimiteInfSup = TipoDeDado
6     "min"  (STRING | ExpressaoJS | "valor armazenado" ID | "campo" ID )
7     "max"  (STRING | ExpressaoJS | "valor armazenado" ID | "campo" ID ) ;
8 ListaDeValoresEnumerados = "valores ("  STRING  { ","  STRING }*  ")" ;
```

O dicionário é composto por classes, que podem ter superclasses. Dessa forma facilita-se a especificação de classes que tenham um conjunto de informações em comum. Uma `Classe` é composta por `Propriedades`, onde são especificados os valores válidos. `Propriedades` podem ser compostas por `Classes de Equivalência`, quando necessita-se testar tais conjuntos de valores em separado.

Os campos das interfaces de usuário devem estar relacionados às regras do dicionário de dados que gerem os valores permitidos para cada um (Listagem 15). Passos do tipo `InformarCampo` também podem referenciar uma determinada regra do dicionário, normalmente para exercitar classes de equivalência em partes diferentes do fluxo (Listagem 6, linha 6).

O conjunto de valores válidos é gerado em tempo de execução, podendo ser originados a partir de informações do tipo do campo e seus limites mínimo e máximo, de uma lista de valores enumerados, da lista de opções carregadas para um campo de seleção, de uma expressão regular, de uma expressão JavaScript ou de um SQL. A descrição detalhada de cada um dos tipos encontra-se no Apêndice A. Quando o valor válido para um campo depende de valores informados para outros campos, pode-se indicar essa relação na especificação.

A Listagem 17 exibe a especificação do dicionário de dados relacionado ao caso de uso `CadastrarPessoa`.

Listagem 17 – Exemplo de especificação – Dicionário de dados

```
1 Classe Pessoa {
2   nome:  Texto min "10" max "60" ,
3   email:  expressao "[a-z0-9]{1,30}[@][a-z0-9]{1,10}[.][a-z0-9]{2,3}",
```

```
4   possuiCNH:  [sim: valores ("S"), nao: valores ("N")],
5   categoriaCNH:  valores ("A", "B", "C", "D")
6 }
```

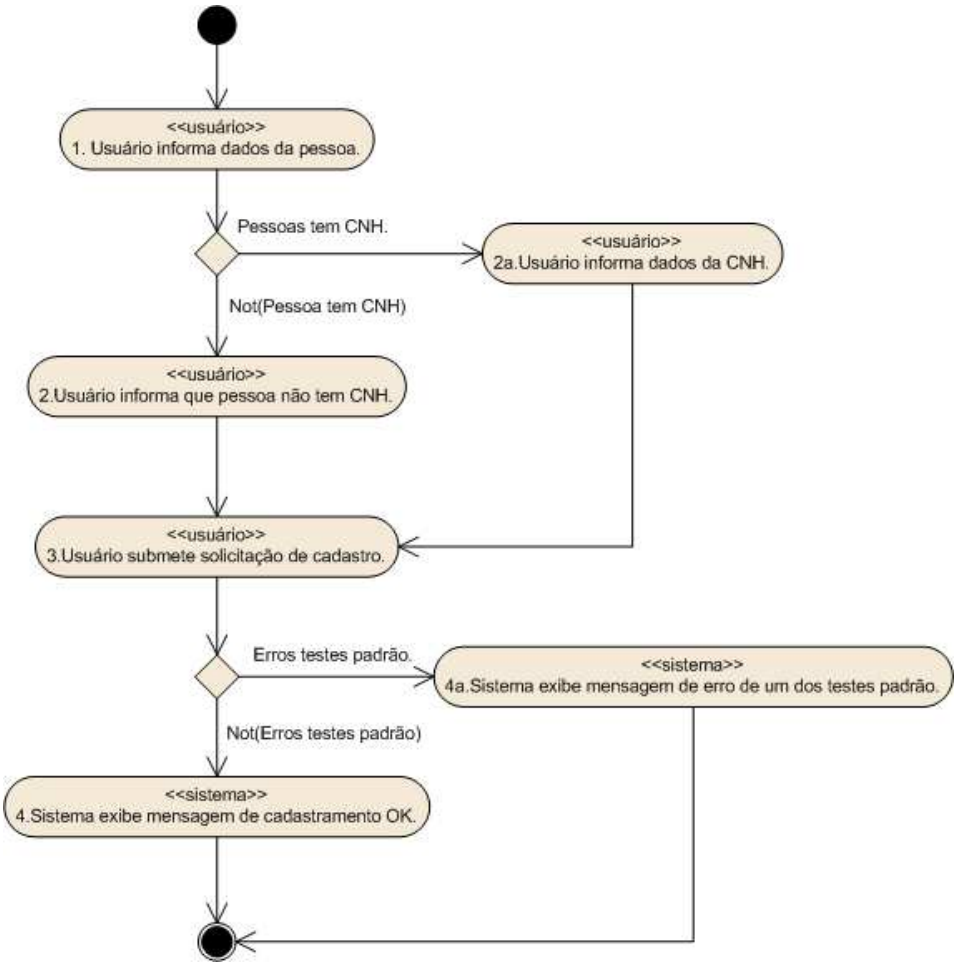
3.3 Derivação dos Testes

Esta seção apresenta as etapas realizadas para a derivação dos testes a partir da especificação. A Subseção 3.3.1 descreve o processo de geração de cenários para cada caso de uso. E a Subseção 3.3.2, o processo de geração de casos de teste.

3.3.1 Geração de Cenários de Caso de Uso

A geração de cenários recebe, como entrada, o caso de uso com seus passos em alto nível e gera os cenários possíveis, por meio de um diagrama de atividades. A Figura 8 exibe o diagrama de atividades para o caso de uso `CadastrarPessoa`. Como pode ser visto, as extensões são transformadas em pontos de decisão.

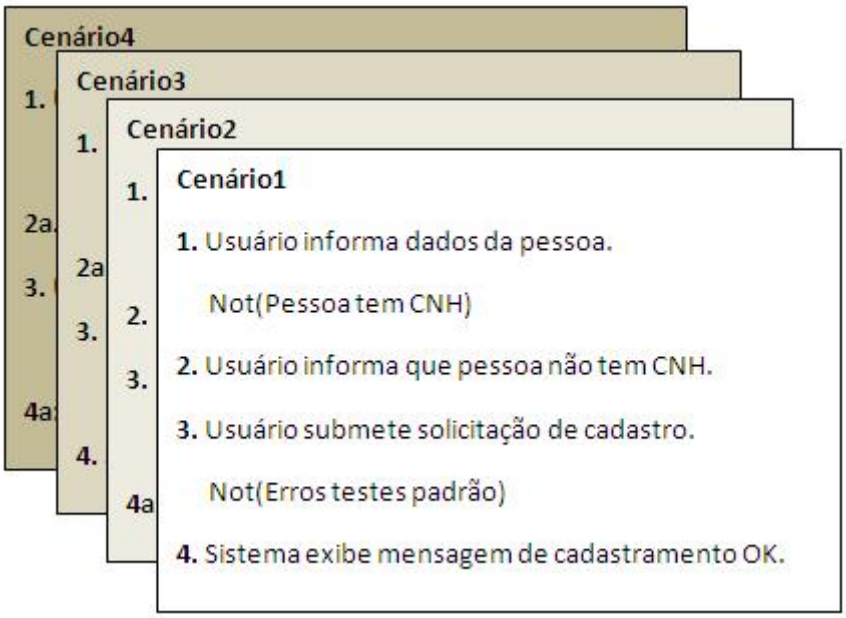
Figura 8 – Diagrama de atividades gerado para o caso de uso `CadastrarPessoa`



É gerada então uma lista, com todas as opções de caminhos que percorrem o diagrama de atividades do início ao fim. As combinações tomam como base as arestas que saem dos nós de

decisão, resultando em todas as possibilidades de execução concreta do caso de uso – o algoritmo encontra-se em Gutiérrez et al. (2007). Havendo ciclos no diagrama, esses são percorridos uma única vez em cada caminho de que façam parte. Os cenários gerados para o caso de uso do diagrama são exibidos na Figura 9.

Figura 9 – Cenários gerados para o caso de uso `CadastrarPessoa`



Cenário1: 1_2_3_4 /Cenário2: 1_2_3_4a /Cenário3: 1_2a_3_4 /Cenário4: 1_2a_3_4a

3.3.2 Geração de Casos de Teste

A etapa de geração de casos de teste tem como entrada o modelo – que reflete a especificação – e a lista de cenários de caso de uso, gerada na etapa anterior. A etapa ocorre em duas sub-etapas, que são descritas a seguir: (i) sequenciamento entre casos de uso gerando cenários completos de teste e (ii) geração de testes padrão de campos.

3.3.2.1 Sequenciamento de Casos de Uso

A invocação de um caso de uso por outro – dentro dos passos do cenário ou nas pré-condições – favorece a reutilização e a manutenção dos casos de uso, além de permitir o sequenciamento entre os mesmos.

Casos de uso invocados são executados em todos os seus cenários de sucesso, a menos que sejam indicadas extensões que ele deva percorrer. Nesse caso, são identificados somente os cenários de sucesso que percorram todas as extensões indicadas. Caso alguma das extensões indicadas seja de fluxo de erro, então são identificados todos os cenários que percorram todas as extensões, sendo de erro ou não. Nesse caso, entende-se que a invocação de cenários de erro é intencional, em casos de uso que queiram sequenciar e verificar situações específicas. A opção

por selecionar somente cenários que terminam em sucesso – a não ser que seja especificado o contrário – visa garantir que o caso de uso chamador possa executar seu fluxo completo. Por exemplo, se um caso de uso "Contratar Pessoa" invoca um caso de uso "Cadastrar Candidato" e esse falhar, o primeiro poderá não ser completado.

Para cada cenário de caso de uso são selecionados os cenários possíveis para cada ponto de invocação, e então, por meio da combinação desses cenários, são geradas todas as possibilidades de sequenciamento, ou cenários completos de teste. O pseudocódigo é apresentado no Algoritmo 1. O algoritmo completo pode ser visualizado no Apêndice B.

Algoritmo 1: Possibilidades de sequenciamento de casos de uso

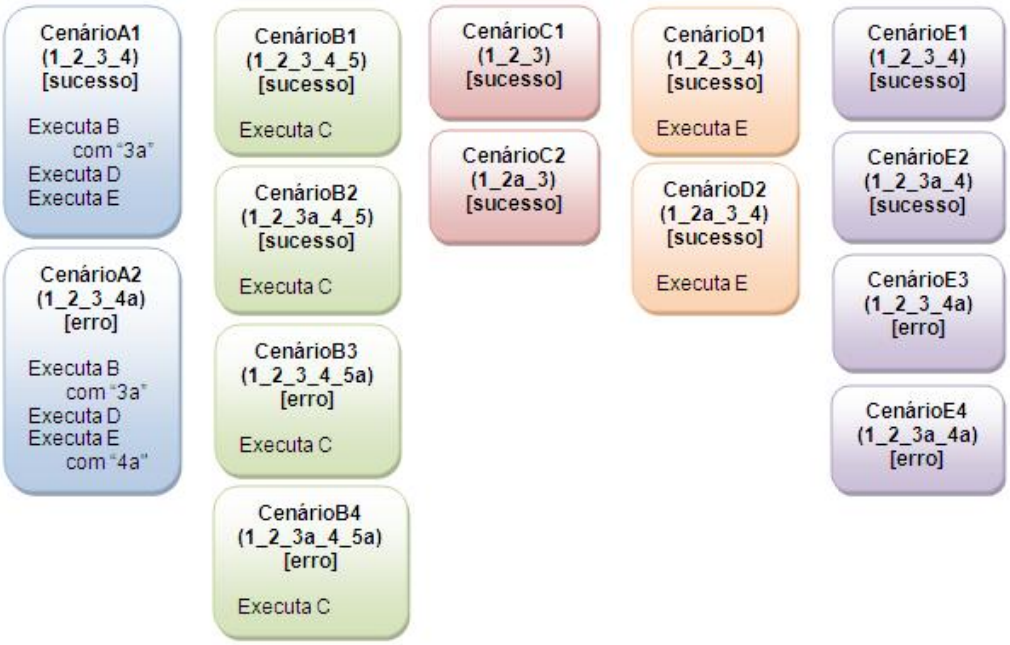
```
1  for Cada cenário de caso de uso do
2      I <- Invocações a outros casos de uso;
3      for Cada invocação de caso de uso do
4          if (invocação sem indicação de extensão) then
5              C_i <- Cenários de sucesso do caso de uso invocado;
6          end
7          else
8              C_i <- Cenários do caso de uso invocado que passem pela extensão indicada;
9          end
10         S <- Conjunto de sequenciamentos por meio das combinações de C;
11     end
12 end
```

A Figura 10 exibe um exemplo hipotético de casos de uso com invocações e o Quadro 10, as opções de combinação de cenários em cada ponto de invocação. Os sequenciamentos resultantes são:

Para o cenário A1: {B2 D1 E1}, {B2 D1 E2}, {B2 D2 E1}, {B2 D2 E2}.
Para o cenário A2: {B2 D1 E3}, {B2 D1 E4}, {B2 D2 E3}, {B2 D2 E4}.
Para todos os cenários de B: {C1}, {C2}.
Para todos os cenários de C: {}.
Para todos os cenários de D: {E1}, {E2}.
Para todos os cenários de E: {}.

A combinação de cenários é realizada considerando os cenários dos casos de uso invocados somente em um primeiro nível. Se um cenário do caso de uso *A* invocar o caso de uso *B*, e o caso de uso *B* invocar o caso de uso *C*, os cenários de *C* não serão considerados para as combinações de sequenciamentos de *A*. Assim, executando-se casos de teste gerados para o caso de uso *A*, chegando-se ao ponto em que *B* invoca *C*, será escolhido um cenário aleatoriamente, entre os cenários permitidos na invocação de *C* em *B*. Entretanto, na geração de sequenciamentos para o caso de uso *B*, as combinações de *C* serão consideradas.

Figura 10 – Exemplo hipotético de casos de uso com invocações



Quadro 10 – Opções de combinação de cenários de casos de uso invocados

Cenário	**Invocações**		
A1	**B(com 3a)***	**D**	**E**
	B2	D1	E1
		D2	E2
A2	**B(com 3a)***	**D**	**E(com 4a)****
	B2	D1	E3
		D2	E4
B(todos)	**C**		
	C1		
	C2		
C(todos)	**–**		
D(todos)	**E**		
	E1		
	E2		
E(todos)	**–**		

* B (com 3a) = Todos os cenários de sucesso de *B* que percorram a extensão *3a*
** E (com 4a) = Todos os cenários de *E* que percorram a extensão *4a*, mesmo sendo cenários de erro
Demais invocações: Todos os cenários de sucesso

A cobertura de casos de teste gerados para um cenário de um caso de uso *A* que invoque dois casos de uso (*B* e *C*) pode ser calculada como:

$$\text{COB Cenario de A} = \frac{\text{Cenários Selecionados de B}}{\text{Cenários de B}} \text{ X } \frac{\text{Cenários Selecionados de C}}{\text{Cenários de C}}$$

Se o caso de uso *B* tiver quatro cenários sendo dois selecionados, e o caso de uso *C* tiver seis cenários sendo quatro selecionados, a cobertura alcançada seria de oito combinações, ou seja, 30% da cobertura total (24 combinações). Embora a cobertura não seja total, entende-se que seja eficaz, devido aos motivos expostos anteriormente quanto à seleção de cenários para a combinação.

Nota-se que a combinação entre cenários para a geração de sequenciamentos é multiplicativa, o que pode vir a gerar muitos casos de teste em determinadas situações. A Seção B.3 explora possibilidades que permitem diminuir o número de testes a executar, em momentos em que não se deseje realizar uma execução completa.

3.3.2.2 Geração de Testes Padrão de Campos

Esta sub-etapa tem por finalidade derivar testes que exercitem um mesmo cenário variando a forma de geração de valores para o conjunto de dados de entrada. Uma das formas de testes padrão (TP) é a de geração de valores aleatórios (dentro das regras especificadas no dicionário de dados).

As outras formas de geração verificam se o SST comporta-se adequadamente recebendo dados de entrada exatamente nos limites especificados, fora desses limites ou fora do formato esperado, recebendo somente os dados indicados como obrigatórios, ou não recebendo algum dado obrigatório. Comportar-se adequadamente quando os dados de entrada são inválidos significa que o SST acusa tal situação, exibindo mensagem apropriada. Os Quadros 11, 12 e 13 apresentam os testes gerados para as situações descritas, de acordo com o tipo de cenário.

Quadro 11 – Testes padrão de campos – Cenários de Sucesso

Nro de testes	**Teste Padrão(TP)**	**Descrição**
1	ValoresVálidosQuaisquer	Valores válidos quaisquer
1	SóObrigatórios	Somente campos obrigatórios (com valores válidos quaisquer)
2 (Se Limite)	ValoresVálidosMin ValoresVálidosMax	Todos os campos com valor/tamanho mínimo Todos com valor/tamanho máximo

Quadro 12 – Testes padrão de campos – Cenários de Erro

Nro de testes	**Teste Padrão(TP)**	**Descrição**
1	ValoresVálidosQuaisquer	Valores válidos quaisquer

Quadro 13 – Testes padrão de campos – Cenários de Erro-Testes-Padrão

Nro de Testes	Teste Padrão(TP)	Descrição
1 para cada campo Obrigatório (O)	CampoFaltante	Todos os campos com valores válidos quaisquer exceto um, que não é informado
1 para cada campo com Formato (F)	ValorForaFormato	Todos os campos com valores válidos quaisquer exceto um, com valor não permitido pelo formato
1 para cada campo com Limite (L)	ValorInválidoMin -1	Todos os campos com valores válidos quaisquer exceto um, com valor imediatamente anterior ao mínimo
1 para cada campo com Limite (L)	ValorInválidoMax +1	Todos os campos com valores válidos quaisquer exceto um, com valor imediatamente posterior ao máximo

`Cenários de sucesso` são todos os que finalizam no fluxo do cenário principal ou em fluxos de extensão que não são indicados como sendo fluxos de erro. `Cenários de Erro` são cenários que finalizam em fluxos indicados como tal, assim como os `Cenários de Erro-Testes-Padrão`. A indicação de que um fluxo de extensão é de erro ou de erro-testes-padrão é feita ao lado da descrição da extensão, como pode ser visto na Subseção 3.2.1.

Os testes gerados para um `Cenário de Erro` são do tipo `Valores Válidos Quaisquer` uma vez que, diferentemente de um `Cenário de Erro-Testes-Padrão`, a atribuição de valores incorretos não é realizada automaticamente; deve ser indicada pelo usuário no fluxo em questão, sendo o restante dos pontos de atribuição tratados como valores aleatórios, dentro da faixa permitida.

3.3.2.3 Lista de Casos de Teste Resultante

O resultado desta etapa é uma lista de casos de teste, com identificadores (códigos de execução) que seguem a seguinte estrutura:

Id (caso de uso) . Id (cenário) . Id (sequenciamento) . Id (teste padrão)

Os identificadores (*ids*) são sequenciais em cada nível. A geração da lista é obtida percorrendo todos os casos de uso que tenham o atributo *execução própria* = "S". Cada caso de uso recebe um *id* sequencial, e então percorre-se todos os seus cenários, que também recebem *ids* sequenciais, iniciados dentro de cada caso de uso. Em cada cenário percorre-se a lista de sequenciamentos e, por último, percorre-se a lista de testes padrão, sempre atribuindo *ids* sequenciais dentro de cada nível. O Algoritmo 2 apresenta o pseudocódigo para essa geração.

A lista de casos de teste gerada para o caso de uso `CadastrarPessoa` é exibida na Listagem 18. Como pode-se observar, os identificadores dos casos de teste:

Algoritmo 2: Algoritmo que gera a lista de casos de teste disponíveis para execução

```
1  for Cada caso de uso do
2      for Cada cenário de caso de uso do
3          for Cada sequenciamento/cenário completo de teste do
4              for Cada teste padrão do
5                  idCasoTeste <- idCasoUso . idCenário . idSequenciamento .
                   idTestePadrão;
6                  CasoDeTeste_i <- geraCasoDeTeste(idCasoTeste);
7              end
8          end
9      end
10 end
```

. no segundo nível - variam de 1 a 4 (devido aos 4 cenários de caso de uso);

. no terceiro nível - não variam (por haver somente 1 caminho de sequenciamento[4]);

. no quarto nível - variam de forma diferenciada: para os cenários 1.1 e 1.3 variam de 1 a 4 (testes padrão de final com sucesso), e para os cenários 1.2 e 1.4 variam de acordo com as características de cada campo (para os quais são gerados os testes padrão de final com erro).

Listagem 18 – Lista de testes gerados para o caso de uso `CadastrarPessoa`

```
1  1 : CasodeUso CadastrarPessoa [prioridade:3]  [tags:pessoa] [TelaCadastrarPessoa]
2  1.1 : CadastrarPessoa/Cenario_1_2_3_4
3  1.1.1 : sequenciamento: [FazerLogin/Cenario_1_2_3]
4  1.1.1.1 : TP_VALIDOS_QUAISQUER[]
5  1.1.1.2 : TP_SO_OBRIGATORIOS[]
6  1.1.1.3 : TP_VALORES_VALIDOS_MIN[]
7  1.1.1.4 : TP_VALORES_VALIDOS_MAX[]
8  1.2 : CadastrarPessoa/Cenario_1_2_3_4a
9  1.2.1 : sequenciamento: [FazerLogin/Cenario_1_2_3]
10 1.2.1.1 : TP_VALOR_INVALIDO_MIN_MENOS_UM[nome]
11 1.2.1.2 : TP_VALOR_INVALIDO_MAX_MAIS_UM[nome]
12 1.2.1.3 : TP_VALOR_INVALIDO_FORA_DO_FORMATO[email]
13 1.2.1.4 : TP_VALOR_FALTANTE[nome]
14 1.2.1.5 : TP_VALOR_FALTANTE[possuiCNH]
15 1.3 : CadastrarPessoa/Cenario_1_2a_3_4
16 1.3.1 : sequenciamento: [FazerLogin/Cenario_1_2_3]
17 1.3.1.1 : TP_VALIDOS_QUAISQUER[]
18 1.3.1.2 : TP_SO_OBRIGATORIOS[]
19 1.3.1.3 : TP_VALORES_VALIDOS_MIN[]
20 1.3.1.4 : TP_VALORES_VALIDOS_MAX[]
21 1.4 : CadastrarPessoa/Cenario_1_2a_3_4a
22 1.4.1 : sequenciamento: [FazerLogin/Cenario_1_2_3]
23 1.4.1.1 : TP_VALOR_INVALIDO_MIN_MENOS_UM[nome]
24 1.4.1.2 : TP_VALOR_INVALIDO_MAX_MAIS_UM[nome]
25 1.4.1.3 : TP_VALOR_INVALIDO_FORA_DO_FORMATO[email]
26 1.4.1.4 : TP_VALOR_FALTANTE[nome]
```

[4] Apesar de o caso de uso `CadastrarPessoa` não ter invocação a outros casos de uso, a descrição do sequenciamento aponta para a execução do caso de uso `FazerLogin`. Esse apontamento é feito automaticamente para todos os casos de uso que indicam um ator para sua execução. O caso de uso utilizado para a autenticação é o indicado como tal, na especificação.

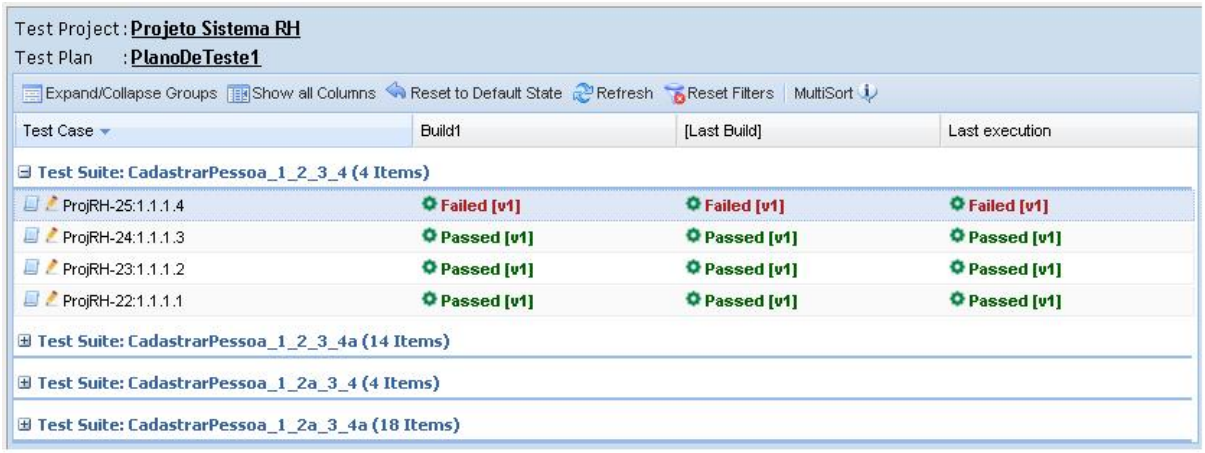
```
1.4.1.5 : TP_VALOR_FALTANTE[possuiCNH]
1.4.1.6 : TP_VALOR_FALTANTE[categoriaCNH]
```

3.4 Execução e Análise dos Resultados

Para cada execução é registrado um *log* – composto pelos passos percorridos, os dados de entrada e o resultado (sucesso/falha) – para posterior depuração. A ocorrência de falhas resulta no armazenamento das seguintes evidências: um *printscreen* da tela e uma pilha de erros Java. A pilha de erros contém informações sobre o caso de uso, o cenário e o passo em que ocorreu o erro, além da falha apontada na execução. Tais evidências visam auxiliar na descoberta do motivo do erro.

A Figura 11 exibe a lista de casos de testes executados para o caso de uso `Cadastrar Pessoa`, agrupados em suítes de teste (que nesta proposta são os cenários dos casos de uso). Os resultados apresentados (sucesso/falha) são referentes à última execução.

Figura 11 – Resultado das execuções do caso de uso `CadastrarPessoa`



Já a Figura 12 mostra o resultado de duas execuções do caso de teste 1.1.1.4: a primeira com finalização normal e a segunda, com falha.

3.5 LUCTool

Esta Seção apresenta a ferramenta LUCTool, desenvolvida para esta pesquisa. A ferramenta é composta pelo ambiente EnvUCT (*Environment for Use Cases to Tests*) – onde é realizada a especificação dos casos de uso, pelo módulo UC2Test (*Use Case to Test*) – que realiza a derivação dos casos de uso em testes, e o módulo UC2Test-Exec – que executa os *scripts* gerados e armazena seus resultados. A Figura 13 exibe a estrutura do ambiente proposto e os artefatos utilizados durante o processo. Detalhes podem ser obtidos no Apêndice B.

Figura 12 – Resultado das execuções do caso de teste 1.1.1.4

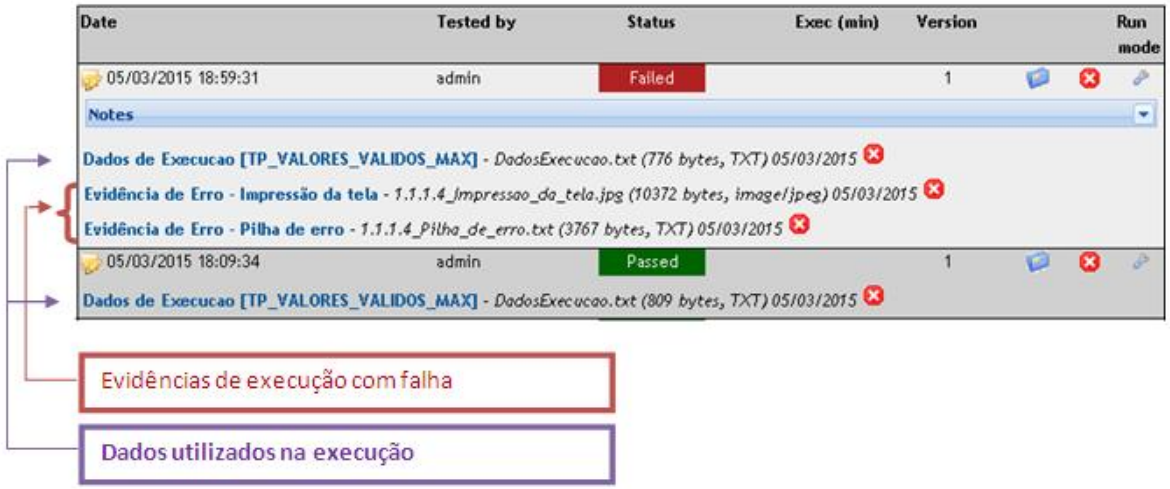
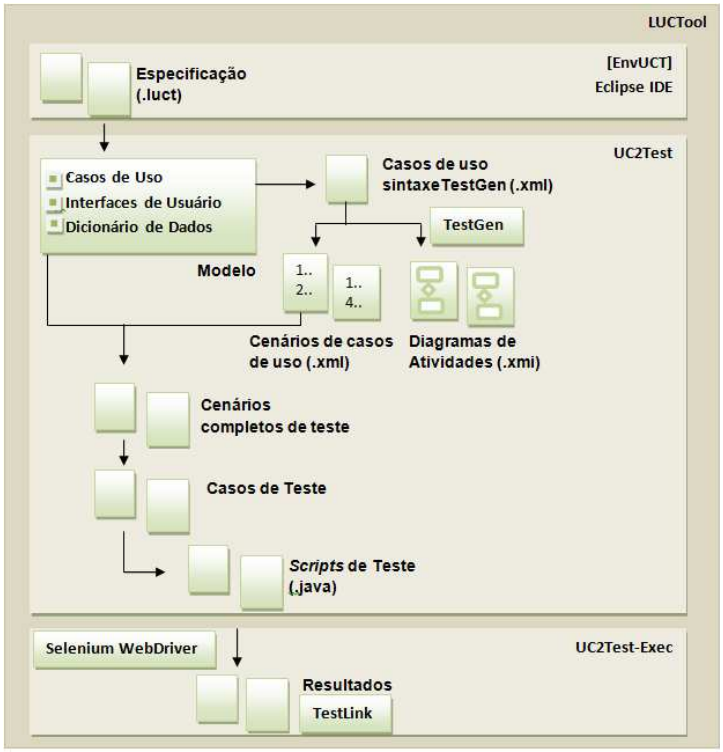


Figura 13 – Estrutura e artefatos do ambiente proposto.



4 EXPERIMENTOS

Este capítulo apresenta uma avaliação da abordagem proposta, por meio de estudos de caso com sistemas de software reais. A Seção 4.1 apresenta os sistemas de software selecionados, a Seção 4.2 analisa a expressividade da DSL LUCT, e por fim, as Seções 4.3 e 4.4 apresentam os experimentos realizados. Ao longo do capítulo são apresentados exemplos de casos de uso, sendo exibidos somente os trechos necessários para o entendimento de cada assunto explorado. Os casos de uso completos são apresentados nos Apêndices C e D.

4.1 Seleção de Sistemas de Software

Para a avaliação proposta selecionou-se, em uma universidade, dois sistemas de software em uso que tivessem seus requisitos especificados como casos de uso textuais, documentação das interfaces de usuário e dicionário de dados, apresentação em interface *web*, além de código-fonte disponível.

O primeiro software selecionado é utilizado para gerenciar ações de extensão de uma universidade, como programas, projetos, cursos, eventos e prestações de serviços. É implementado em Java, com uso do *framework* Struts. O software possui 71 funcionalidades, sendo 26 do tipo "Gerir Entidade" (que agrupam subfuncionalidades do tipo de ações CRUD), sete do tipo de "Etapa de *workflow*" e 38 do tipo "Execução de Relatórios" (Tabela 1).

O segundo software selecionado é utilizado para gerenciar informações e atividades de ex-alunos. É implementado em Java, com uso de JavaServerFaces e JBossSeam. O software possui 25 funcionalidades, sendo 17 do tipo "Gerir Entidade" (que agrupam subfuncionalidades do tipo de ações CRUD), duas do tipo "Gerir Entidade Composta" (que agrupam, cada uma, de cinco a nove subfuncionalidades "Gerir Entidade", não computadas no primeiro grupo) e seis do tipo "Execução de Relatórios" (Tabela 1).

Tabela 1 – Número de funcionalidades dos sistemas de software selecionados

Sistema de Software	Gerir Entidade simples	Gerir Entidade composta	Etapa de *workflow*	Execução de Relatórios	Total
Sistema de Informações de Extensão	26	0	7	38	71
Sistema de Ex-Alunos	17	2	0	6	25

4.2 Expressividade da DSL LUCT

Esta seção tem por objetivo discorrer sobre a expressividade da DSL LUCT, por meio da análise de sua aplicação prática, utilizando casos de uso de sistemas de software reais, apresenta-

dos na Seção 4.1. Foram exercitados casos de uso pertencentes aos tipos de funcionalidade “Gerir Entidade simples” e “Gerir Entidade composta”. Considera-se que para as demais classificações – etapas de *workflow* e execução de relatórios – são utilizados recursos comuns aos demais tipos de casos de uso, conforme explanado a seguir.

Os casos de uso que configuram etapas de *workflow* – identificados no software que trata de informações de extensão – são compostos por fluxos que submetem, aprovam, reprovam, concluem ou invalidam uma ação de extensão, assim como fluxos que fazem a sua auditoria, aprovando ou reprovando suas atualizações. Esses fluxos passam pela seleção da ação de extensão, sua exibição e a solicitação de mudança de seu *status*, perante a indicação de uma justificativa. Assim, tais fluxos podem ser considerados como ações do tipo CRUD, uma vez que executa-se uma pesquisa e uma alteração de dados. Para a execução do *workflow*, pode-se definir um caso de uso com invocações a todas as etapas (casos de uso) em sequência, sendo os dados necessários às etapas posteriores, disponibilizados por meio de armazenamento de valores, recurso oferecido pela DSL.

Considerando que relatórios podem envolver regras de negócio na seleção de seus dados, e que a exibição de um elemento em um relatório pode ser considerada a validação de um processo executado por um ou mais casos de uso, pode ser desejável a verificação do resultado de sua execução. Para casos de uso que envolvem a execução de relatórios, a DSL permite expressar a solicitação da execução, uma vez que essa envolve o preenchimento de um formulário com campos com a função de filtros e a sua submissão. Para especificar a verificação do resultado, desde que esse seja apresentado em HTML, a DSL apresenta as possibilidades de buscar textos em uma tabela ou em um campo – caso mapeie-se o resultado do relatório como uma tela. Os textos procurados são, possivelmente, valores armazenados em etapas anteriores, durante a execução do processo que deverá resultar nos dados esperados no relatório.

Assim, considera-se que, em ambos os casos, não se faz necessária a realização de uma análise à parte, uma vez que são utilizados mecanismos comuns aos demais tipos de casos de uso, exercitados nesta seção.

4.2.1 Análise da Expressividade a partir de Exemplos de Casos de Uso

Nesta subseção são apresentados cinco casos de uso, para a realização da análise proposta. Esses exemplos foram selecionados por apresentarem características variadas e certa complexidade, sendo alguns pontos não cobertos, ou cobertos parcialmente, pela DSL.

Para cada caso de uso, descreve-se a sua finalidade, suas características e como as mesmas foram especificadas utilizando a DSL, sendo exibidos, para cada uma, o trecho da especificação e o trecho da DSL correspondente.

O caso de uso `InserirCurso` realiza a inclusão, na base de dados, de informações a respeito de um curso de extensão a ser ofertado à comunidade. Essas informações incluem seu período de realização, carga horária, público estimado, entre outros[1]. A Listagem 19 apresenta a estrutura básica do caso de uso `InserirCurso`, que contém um acionador – que acessa a funcionalidade a partir de um *link* do menu – e o cenário principal – que contém passos que informam os campos da IU, solicitam a inclusão do curso e, por fim, visualizam o resultado da solicitação. A Listagem 20 apresenta o trecho da DSL em que define-se a estrutura básica para os casos de uso, e a Listagem 21, os trechos que definem os passos básicos de interação com a IU, citados acima e comuns à maioria dos casos de uso (acessar uma tela por meio de um menu, informar campos, submeter um formulário – normalmente acionando um botão – e visualizar os resultados exibidos).

Listagem 19 – Caso de uso `InserirCurso`-Estrutura básica

```
1  CasoDeUso InserirCurso (!){
2    ...
3    acionador {
4      Acesso link LinkInserirCurso
5    }
6    cenarioprincipal
7      1 "Usuário informa campos do novo curso (informações gerais de ação de extensão)
           " {
8        Informo[titulo,dataInicio,previsaoTermino, areaTematicaPrincipal],
9        ...
10     },
11     2 "Usuário informa que novo curso é de caracterização geral (não aperfeiçoamento
           ) e a carga horária" {
12       Informo caracterizacao com Curso.caracterizacao.geral,
13       Informo cargaHoraria com Curso.cargaHoraria.geral
14     },
15     3 "Usuário informa demais campos do novo curso (informações específicas de
           cursos)" {
16       ...
17       Informo[gratuito,publicoEstimado]
18     },
19     4 "Usuário solicita inclusão do novo curso" {
20       Clico Salvar
21     },
22     5 "Sistema exibe mensagem de sucesso" {
23       Visualizo mensagem DadosDoCursoSalvos
24     }
25   ...
26 }
```

Listagem 20 – DSL LUCT-Estrutura básica de casos de uso

```
1  CasoDeUso =
2    "CasoDeUso"  ID  [NivelObjetivo]  "{"
3      ...
4      ["acionador"  Acionador ]
5      ...
```

[1] A estrutura completa pode ser vizualizada no Apêndice C

```
6       (("cenarioprincipal"  CenarioPrincipal) | (PENDENTE  STRING))
7       ...
8     "}"
9   ;
10  NivelObjetivo = ( "(+)" | "(!)" | "(-)" ) ;
11
12  CenarioPrincipal = PassoUC  { ","  PassoUC }* ;
13  PassoUC = INT  STRING  "{"  Passo  { ","  Passo }*  "}" ;
```

Listagem 21 – DSL LUCT-Passos básicos de interação com a IU

```
1   Acessar = "Acesso"  ("menu"  ID | "link"  ID) ;
2
3   InformarCampo = "Informo"  ID  [ "com"  Valor ]  ;
4   InformarCampos = "Informo["  ID  { ","  ID }*  "]" ;
5   Valor = ( ValorFixo | ValorDinamico | "valor armazenado" ID ) ;
6   ValorFixo = STRING ;
7   ValorDinamico = ID "." ID [ "." ID ] ;
8
9   Clicar = "Clico"  ID ;
10
11  VerificacaoNaTelaPrincipal = "Visualizo "  ( "mensagem"  ID | "texto = "  STRING | "
        valor armazenado" ID )  [ ValorTela ] ;
12  VerificacaoDeMsgErroTestesPadraoNaTelaPrincipal = "Visualizo uma mensagem de erro de
         testes padrão"  [ ValorTela ] ;
13  ValorTela = ( "no campo"  ID | "na tabela"  TabelaLinhaColuna ) ;
```

A definição completa de um caso de uso, conforme descrito na metodologia utilizada, é composta pela descrição de casos de uso, interfaces de usuário e dicionário de dados. De acordo com as possibilidades de definição em um dicionário dados, esse caso de uso apresenta as seguintes características: regras de negócio que utilizam valores mínimos e máximos, lista de valores enumerados, um valor qualquer da lista de opções disponíveis e dependência de valores entre campos, conforme mostrado na Listagem 22. As regras da DSL que permitem realizar tais definições no dicionário de dados são exibidas na Listagem 23.

Listagem 22 – Caso de uso `InserirCurso`-Dicionário de dados

```
1   Classe AcaoExtensao {
2     titulo: Texto min "1" max "400",
3     dataInicio:  Data min "01/01/1950" max "",
4     previsaoTermino:  Data min campo dataInicio max "",
5     areaTematicaPrincipal: valores da lista (1..n),
6     ...
7   }
8
9   Classe Curso ClasseBase AcaoExtensao {
10    gratuito:  valores ("true","false"),
11    publicoEstimado:  Numerico min "0" max "999999999"
12    ...
13  }
```

Listagem 23 – DSL LUCT-Dicionário de dados

```
1  ClasseDicionario = "Classe"  ID   [ "ClasseBase"  ID ]  "{"
       PropriedadeDicionario  { ","  PropriedadeDicionario }*  "}" ;
2
3  PropriedadeDicionario = ID  ":"  ( ValoresValidos |
4       "[" ClassesEquivalenciaDicionario  "]"  | PENDENTE ) ;
5
6  ValoresValidos = (TipoComLimiteInfSup | ListaDeValores | Regex | ExpressaoJS | SQL);
7
8  ListaDeValores = ( ListaDeValoresEnumerados | ListaDeValoresValorQualquerDaLista ) ;
9  ListaDeValoresEnumerados = "valores ("  STRING  { ","  STRING }*  ")" ;
10 ListaDeValoresValorQualquerDaLista = "valores da lista (1..n)" ;
11 TipoComLimiteInfSup = TipoDeDado
12     "min"  (STRING | ExpressaoJS | "valor armazenado" ID | "campo" ID )
13     "max"  (STRING | ExpressaoJS | "valor armazenado" ID | "campo" ID ) ;
14 TipoDeDado =  ( "Data" | "Texto" | "Numerico" ) ;
```

Note que um dicionário de dados é representado por meio de classes que têm um nome, nesse exemplo, `Curso`, que apresenta uma classe base, `AcaoExtensao`. A indicação de classes com uma relação de herança permite que em casos de uso relacionados aos outros tipos de ação de extensão (eventos, prestações de serviço, programas e projetos), não se faça necessário informar e replicar os itens comuns do dicionário.

Ainda de acordo com as regras de negócio do caso de uso, faz-se necessária a definição de classes de equivalência, uma vez que a carga horária do curso apresenta variação de valores permitidos, seguindo um limite mínimo para cursos de aperfeiçoamento e outro para cursos com outras caracterizações, conforme exibido na Listagem 24. O trecho da DSL que permite definir classes de equivalência é apresentado na Listagem 25.

Listagem 24 – Caso de uso `InserirCurso`-Classes de equivalência

```
1  Classe Curso ClasseBase AcaoExtensao {
2    caracterizacao:  [aperfeicoamento: valores ("1"), geral: valores ("2","3","4")],
3    cargaHoraria: [aperfeicoamento: Numerico min "180" max "99999", geral: Numerico
         min "8" max "99999"],
4    ...
5  }
```

Listagem 25 – DSL LUCT-Classes de equivalência

```
1  PropriedadeDicionario = ID  ":"  ( ValoresValidos |
2           "[" ClassesEquivalenciaDicionario  "]"  | PENDENTE ) ;
3
4  ClassesEquivalenciaDicionario = ClasseEquivalenciaDicionario  { ","
       ClasseEquivalenciaDicionario }* ;
5  ClasseEquivalenciaDicionario = ID  ":"  ValoresValidos ;
```

Assim, devido à variação de comportamento relatada, é definido um fluxo de extensão decorrente do uso de classes de equivalência. Nesse fluxo, é necessário preencher as informações

específicas de cursos de aperfeiçoamento: sua caracterização como tal e sua carga horária. Além disso, para verificar o correto preenchimento dos campos da IU, faz-se necessário definir um segundo fluxo de extensão, que realiza os testes de erro padrão. Nesse fluxo, indica-se um passo para a visualização da mensagem de erro e o local onde a mensagem será visualizada. Enquanto o primeiro fluxo retorna ao cenário principal, o segundo finalizará no próprio fluxo de extensão. A Listagem 26 apresenta os dois fluxos de extensão, e a Listagem 27, o trecho da DSL que possibilita a definição de extensões. Já os trechos da DSL que definem os passos básicos de interação com a IU foram anteriormente exibidos, na Listagem 21.

Listagem 26 – Caso de uso `InserirCurso`-Extensões

```
1  extensoes
2    2a "Novo curso é de aperfeiçoamento"
3    2a 1 "Usuário informa que novo curso é de aperfeiçoamento e a carga horária"   {
4      Informo caracterizacao com Curso.caracterizacao.aperfeicoamento,
5      Informo cargaHoraria com Curso.cargaHoraria.aperfeicoamento
6    }
7    2a 2 Executa passo 3,
8
9
10   5a erro-teste-padrao "Usuário informa dados incorretos"
11   5a 1 "Sistema exibe mensagem de erro" {
12     Visualizo uma mensagem de erro de testes padrão no campo AreaMensagemErro
13   }
```

Listagem 27 – DSL LUCT-Extensões

```
1  ["extensoes"  Extensoes ]
2  Extensoes = CondicaoEPassosFluxoAlt  { ","  CondicaoEPassosFluxoAlt }* ;
3  CondicaoEPassosFluxoAlt = FluxoAlternativoCondicao
4          { PassoUCAlt }+  [ PassoUCAltFinal ] ;
5  FluxoAlternativoCondicao = INT  LETRA
6          [ ("erro" | "erro-teste-padrao") ]  STRING
7          [ "extensão de tela"  ID ] ;
8  PassoUCAlt = INT  LETRA   INT  STRING
9          [ "{"  Passo  { ,  Passo }*   "}"       ] ;
10 PassoUCAltFinal = INT  LETRA  INT  PassoUCAltExecutaFinal ;
11 PassoUCAltExecutaFinal = Executa passo  INT ;
12
13 Passo = Acao | Verificacao | Controle | PENDENTE ;
```

Por fim, o caso de uso necessita especificar, como pós-condição para os cenários de sucesso, a garantia de que o curso foi inserido na base de dados, com sua data de início e fim armazenados adequadamente (Listagem 28). Tal especificação se faz necessária, uma vez que que casos de uso dependentes de `InserirCurso` necessitam da garantia de tal informação em sua regra de negócio. A Listagem 29 apresenta o trecho da DSL relacionado à definição de pós-condições.

Listagem 28 – Caso de uso `InserirCurso`-Pós-condições

```
garantiasdesucesso "Valores data início e fim (do curso inserido) armazenados" {
  Armazeno campo dataInicio como dataInicioCurso,
  Armazeno campo previsaoTermino como dataTerminoCurso
}
```

Listagem 29 – DSL LUCT-Pós-condições

```
["garantiasdesucesso"  PassoGarantias]
PassoGarantias = STRING  "{"  PassoPos  { ","  PassoPos }*  "}" ;
PassoPos = ( ArmazenarValor | Verificacao| PENDENTE ) ;
ArmazenarValor = "Armazeno"  ( Valor | ValorTela | "campo" ID )  "como"  ID ;
```

A DSL LUCT permitiu expressar todas as características de forma satisfatória. A Listagem 30 apresenta a especificação, com os trechos relacionados à analise do caso de uso.

Listagem 30 – Caso de uso `InserirCurso`

```
CasoDeUso InserirCurso (!){
  ...
  cenarioprincipal
    1 "Usuário informa campos do novo curso (informações gerais de ação de extensão)
       " {
      Informo[titulo,dataInicio,previsaoTermino, areaTematicaPrincipal],
      ...
    },
    2 "Usuário informa que novo curso é de caracterização geral (não aperfeiçoamento
       ) e a carga horária" {
      Informo caracterizacao com Curso.caracterizacao.geral,
      Informo cargaHoraria com Curso.cargaHoraria.geral
    },
    3 "Usuário informa demais campos do novo curso (informações específicas de
       cursos)" {
      ...
      Informo[gratuito,publicoEstimado]
    },
    4 "Usuário solicita inclusão do novo curso" {
      Clico Salvar
    },
    5 "Sistema exibe mensagem de sucesso" {
      Visualizo mensagem DadosDoCursoSalvos
    }

  extensoes
    2a "Novo curso é de aperfeiçoamento"
    2a 1 "Usuário informa que novo curso é de aperfeiçoamento e a carga horária"   {
      Informo caracterizacao com Curso.caracterizacao.aperfeicoamento,
      Informo cargaHoraria com Curso.cargaHoraria.aperfeicoamento
    }
    2a 2 Executa passo 3,

    5a erro-teste-padrao "Usuário informa dados incorretos"
    5a 1 "Sistema exibe mensagem de erro" {
      Visualizo uma mensagem de erro de testes padrão no campo AreaMensagemErro
    }
```

```
35
36    garantiasdesucesso "Curso inserido - Data início e fim armazenados" {
37      Armazeno campo dataInicio como dataInicioCurso,
38      Armazeno campo previsaoTermino como dataTerminoCurso
39    }
40  }
41
42  Classe AcaoExtensao {
43    titulo: Texto min "1" max "400",
44    dataInicio:  Data min "01/01/1950" max "",
45    previsaoTermino:  Data min campo dataInicio  max "",
46    areaTematicaPrincipal: valores da lista (1..n),
47    ...
48  }
49
50  Classe Curso ClasseBase AcaoExtensao {
51    caracterizacao:  [aperfeicoamento: valores ("1"), geral: valores ("2","3","4")],
52    cargaHoraria: [aperfeicoamento: Numerico min "180" max "99999", geral: Numerico
          min "8" max "99999"],
53    gratuito:  valores ("true","false"),
54    publicoEstimado:  Numerico min "0" max "999999999"
55    ...
56  }
```

O caso de uso `InserirParticipacao` permite o cadastro de participantes do curso de extensão, utilizado quando há bolsistas ou voluntários trabalhando na sua organização. O caso de uso envolve a indicação de uma pessoa relacionada à universidade, a informação de qual o tipo dessa relação, a forma de participação dessa pessoa naquele curso e as datas de início e fim de participação. A seguir são apresentadas as características do caso de uso, com destaque para os elementos de especificação ainda não abordados nesta seção.

A primeira característica apresentada é a dependência entre casos de uso definida por pré-condições, uma vez que o caso de uso depende da inserção de um curso, antes de sua execução. A Listagem 31 apresenta a forma como essa pré-condição foi especificada por meio da DSL (invocação do caso de uso InserirCurso) e a Listagem 32, o trecho da DSL que possibilita a definição de pré-condições.

Listagem 31 – Caso de uso `InserirParticipacao`-Dependência por pré-condição

```
1   precondicao "Usuário insere novo curso" {
2     Executo InserirCurso
3   }
```

Listagem 32 – DSL LUCT-Dependência por pré-condição

```
1  ["precondicao"  PreCondicao ]
2  PreCondicao = PassosPreCondicao  { ","  PassosPreCondicao }* ;
3  PassosPreCondicao = STRING [ "{" PassoPreCondicao  { ","  PassoPreCondicao}* "}" ];
4  PassoPreCondicao = ( Executar | ArmazenarValor | Acessar | AcesseiTela | PENDENTE );
5  Executar = "Executo"  ID  [ PreExecucaoParms ]
6          [ "com"  "extensao"  "="  STRING ] ;
```

A segunda característica apresentada é a dependência entre campos envolvendo diferentes casos de uso, uma vez que as datas de início e fim de participação devem, além de obedecer à relação básica de precendência, estar compreendidas dentro do período do curso, indicado no caso de uso `InserirCurso`. Tal definição pôde ser expressada por meio de itens do dicionário com limites mínimo e máximo, relacionados aos valores armazenados *dataInicioCurso* e *dataTerminoCurso*, conforme exibido na Listagem 33. O trecho da DSL que define a especificação do tipo do item de dicionário em questão é apresentado na Listagem 34.

Listagem 33 – Caso de uso `InserirParticipacao`-Dependência entre campos

```
1 Classe Participacao {
2   ...
3   datainicio: Data min valor armazenado dataInicioCurso  max valor armazenado
        dataTerminoCurso,
4   dataTermino: Data min campo dataInicio max valor armazenado dataTerminoCurso
5 }
```

Listagem 34 – DSL LUCT-Item de dicionário com definição de limites mínimo e máximo

```
1 TipoComLimiteInfSup = TipoDeDado
2     "min"  (STRING | ExpressaoJS | "valor armazenado" ID | "campo" ID )
3     "max"  (STRING | ExpressaoJS | "valor armazenado" ID | "campo" ID ) ;
4 TipoDeDado =  ( "Data" | "Texto" | "Numerico" ) ;
```

Além disso, apresenta-se nesse caso de uso a necessidade de indicação, como valores válidos para um campo, de um conjunto de valores fixos que caracterizam um pequeno subconjunto do domínio real, sendo essa indicação necessária para a correta execução (dos testes) do caso de uso. O contexto citado encontra-se na indicação da pessoa participante, que se dá pela sua seleção em uma lista. Entretanto, a lista de pessoas é gerada somente a partir da digitação de parte de um nome válido. Como somente com a seleção de um nome na lista finaliza-se o caso de uso com sucesso, necessitou-se especificar partes de nome consideradas válidas, que foram dadas, nesse caso, por um conjunto de valores fixos. Uma análise sobre a necessidade de informação de domínios com essa característica, além da utilização de valores fixos nesse exemplo é realizada posteriormente nesta seção.

A Listagem 35 exibe o trecho da especificação em que se informa a pessoa participante, assim como a definição dos valores válidos utilizados para o preenchimento dos campos. As regras na DSL que definem um passo do tipo Informar, assim como as regras utilizadas para a definição de elementos do dicionário de dados, foram anteriormente apresentadas nas Listagens 21 e 23, respectivamente.

Listagem 35 – Caso de uso `InserirParticipacao` - Preenchimento de campo com um subconjunto de valores fixos

```
1 CasoDeUso InserirParticipacao (!) {
2   ...
3   cenarioprincipal
```

```
4       1 "Usuário informa dados de participação" {
5         ...
6         Informo nome com Participacao.parteValidaDeNome,
7         Informo listaDeNomes,
8         ...
9       }
10      ....
11  }
12
13  Classe Participacao {
14    parteValidaDeNome: valores ("silva","rocha","miranda"),
15    listaDeNomes: valores da lista (1..n),
16    ...
17  }
```

Por fim apresentou-se, nesse caso de uso, a necessidade de se utilizar uma expressão XPath para a identificação de um elemento da IU. A Listagem 36 exibe o trecho da especificação onde o elemento da IU que apresenta a lista de nomes é identificado dessa forma. Já a Listagem 37 exibe o trecho da DSL que define as possibilidades de identificação de um elemento pertencente a uma tela. Apesar de a DSL oferecer a identificação de elementos da interface por meio da indicação do *label* relacionado, em alguns casos faz-se necessária a indicação de XPath, como nesse exemplo, em que se lida com uma implementação de lista que não a lista HTML convencional. Uma análise a respeito do uso de expressões XPath é realizada posteriormente, nesta seção.

Listagem 36 – Caso de uso `InserirParticipacao` - Identificação de elemento da IU por meio de expressão XPath

```
1  Tela TelaEquipe {
2    ...
3    listaDenomes: CAMPO_LISTA xpath = "//*[@id=\"listaNomeMembro\"]/tr[*]/td/a"
         valores = Participacao.listaDeNomes,
4    ...
5  }
```

Listagem 37 – DSL LUCT-Definição de um elemento da tela

```
1  IdentificacaoNaTela = ( "label ="  STRING | "id ="  STRING |
2    "xpath ="  STRING | "identificacao ="  PENDENTE ) ;
```

Por meio da DSL foi possível expressar todas as características do caso de uso. A Listagem 38 exibe a especificação – trechos relacionados à análise – do caso de uso.

Listagem 38 – Caso de uso `InserirParticipacao`

```
1  CasoDeUso InserirParticipacao (!) {
2    ...
3    precondicao "Usuário insere novo curso" {
4      Executo InserirCurso
5    }
6    cenarioprincipal
```

```
7       1 "Usuário informa dados de participação" {
8         ...
9         Informo nome com Participacao.parteValidaDeNome,
10        Informo listaDeNomes,
11        Informo[datainicio, dataTermino]
12      }
13      ....
14  }
15
16  Tela TelaEquipe {
17    ...
18    * nome: CAMPO_AREA_DE_TEXTO label = "Nome",
19    * listaDenomes: CAMPO_LISTA xpath = "//*[@id=\"listaNomeMembro\"]/tr[*]/td/a"
          valores = Participacao.listaDeNomes,
20    * datainicio: CAMPO_AREA_DE_TEXTO label ="Data início" valores =Participacao.
          datainicio,
21    * dataTermino:CAMPO_AREA_DE_TEXTO label ="Data término" valores =Participacao.
          dataTermino
22  }
23
24  Classe Participacao{
25    ...
26    parteValidaDeNome: valores ("silva","rocha","miranda"),
27    listaDeNomes: valores da lista (1..n),
28    datainicio: Data min valor armazenado dataInicioCurso  max valor armazenado
          dataTerminoCurso,
29    dataTermino: Data min campo dataInicio max valor armazenado dataTerminoCurso
30  }
```

O caso de uso `CriarOrganizacao` é utilizado para incluir informações, na base de dados, de organizações que tenham relação com a universidade, no contexto que envolve os seus ex-alunos. O caso de uso apresenta-se como parte de uma funcionalidade Gerir Entidade composta, formada pelos seguintes casos de uso: inclusão de dados básicos da organização (seu nome, sigla, natureza e setor de atuação), inclusão de endereços eletrônicos, inclusão de endereços, inclusão de telefones, inclusão de datas comemorativas e vinculação de pessoas à organização. A estrutura é do tipo mestre-detalhe, sendo que o caso de uso principal invoca os casos de uso de detalhe, em seu fluxo. Assim, esse exemplo apresenta como características a dependência de casos de uso a partir do fluxo de eventos e o acionamento de telas a partir de áreas de uma tela principal.

A Listagem 39 apresenta o caso de uso principal (`CriarOrganizacao`), com a invocação de dois dos casos de uso de detalhe (`InserirEnderecoEletronico` e `VincularPessoaAOrganizacao`). A listagem exibe também a definição das abas que dão acesso aos dois casos de uso, sendo indicada, para cada uma, a tela acessada por meio daquele elemento da IU. Já a Listagem 41 apresenta os casos de uso invocados, que têm como pré-condição a indicação de ter sido acessada a IU correspondente ao caso de uso. Assim, os acionadores desses casos de uso são externos, podendo ser invocados de diferentes pontos do sistema, como ocorre com o caso de uso `InserirEnderecoEletronico`, que é acionado também pela IU que edita os

contatos de um ex-aluno. As Listagens 40 e 42 apresentam os trechos da DSL que possibilitam a definição desse tipo de dependência entre casos de uso (o passo do tipo “Executar”, que é utilizado dentro do fluxo do caso de uso chamador; a propriedade “aciona tela”, que caracteriza um campo como acionador de uma nova IU; e a pré-condição “AcesseiTela”, que obriga que os casos de uso chamadores tenham acessado a IU relacionada ao caso de uso invocado).

Listagem 39 – Caso de uso `CriarOrganizacao`-Dependência de casos de uso a partir do fluxo de eventos

```
1  CasoDeUso CriarOrganizacao (!){
2    ...
3    cenarioprincipal
4      ...
5      5 "Usuário cadastra endereços eletrônicos" {
6        Clico AbaEmails,
7        Executo InserirEnderecoEletronico
8      },
9      6 "Usuário vincula pessoas à organização" {
10       Clico AbaPessoas,
11       Executo VincularPessoaAOrganizacao
12     }
13     ...
14 }
15 Tela DadosCadastraisOrganizacao {
16   AbaEmails:CAMPO_LINK id="formCadastro:abadetalhesOrganizacaoEnderecoEletronico_lbl
         "      aciona tela EnderecosEletronicos,
17   AbaPessoas:CAMPO_LINK id="formCadastro:abadetalhesOrganizacaoPessoa_lbl"
18              aciona tela VinculacaoPesssoa,
19   ...
20 }
```

Listagem 40 – DSL LUCT-Dependência de casos de uso a partir do fluxo de eventos (Caso de uso principal)

```
1  Executar = "Executo"  ID  [ PreExecucaoParms ]
2          [ "com"  "extensao"  "="  STRING ] ;
3
4
5  Campo = ["*"]  ID  ":"  TipoElementoHtml  IdentificacaoNaTela
6    ...
7    [ "aciona tela"  ID ]
8    ...
9  ;
```

Listagem 41 – Dependência de casos de uso a partir do fluxo de eventos (Pré-condições dos casos de uso invocados)

```
1  CasoDeUso InserirEnderecoEletronico (!) {
2    ...
3    precondicao "Acesso à tela de endereços eletrônicos" {
4      AcesseiTela EnderecosEletronicos
5    }
6    ...
7  }
```

```
8  CasoDeUso VincularPessoaAOrganizacao(!) {
9    ...
10   precondicao "Acesso à tela de vinculação" {
11     AcesseiTela VinculacaoPesssoa
12   }
13   ...
14 }
```

Listagem 42 – DSL LUCT-Dependência de casos de uso a partir do fluxo de eventos (Pré-condições para casos de uso invocados)

```
1 ["precondicao"  PreCondicao ]
2 PreCondicao = PassosPreCondicao  { ","  PassosPreCondicao }* ;
3 PassosPreCondicao = STRING [ "{" PassoPreCondicao  { ","  PassoPreCondicao}* "}" ];
4 PassoPreCondicao = ( Executar | ArmazenarValor | Acessar | AcesseiTela | PENDENTE );
5 AcesseiTela = "AcesseiTela" ID;
```

O caso de uso `InserirEnderecoEletronico` permite a inclusão, na base de dados, de endereços eletrônicos utilizados como contato por organizações ou por ex-alunos da universidade. O caso de uso envolve a informação do endereço eletrônico e a indicação de sua categoria (se e-mail, site, rede social, etc.). Já o caso de uso `VincularPessoaAOrganizacao` permite relacionar um ex-aluno a uma organização, podendo ser indicado seu cargo e o período dessa vinculação. Para a indicação do período de vinculação, faz-se necessária a realização de um cálculo com datas, uma vez que a data término da vinculação deve ser posterior à data de início. Assim, esses casos de uso apresentam, em suas regras de negócio, a necessidade de especificação de duas características ainda não abordadas nos casos de uso anteriores, que a DSL permite expressar da seguinte forma: itens de dicionário com uso de expressão regular – para e-mails – e de expressão JavaScript – para realizar o cálculo com datas no período de vinculação. A Listagem 43 apresenta a especificação dos dois itens e a Listagem 44, os trechos da DSL que permitem a sua definição no dicionário de dados.

Listagem 43 – Casos de uso `InserirEnderecoEletronico` e `VincularPessoaAOrganizacao`-Itens do dicionário

```
1  Classe EnderecoEletronico {
2    enderecoEletronico: [geral: Texto min "1" max "200" ,
3        email: expressao "[a-z0-9]{1,30}[@][a-z0-9]{1,10}[.][a-z0-9]{2,3}"],
4    ...
5  }
6  Classe VinculacaoPesssoa {
7    ...
8    dataInicio: Data min "01/01/1900" max "",
9    dataTermino: Data min expressaoJS("somaDiasAData('{:CAMPO_dataInicio}',1)") max ""
10 }
```

Listagem 44 – DSL LUCT-Itens do dicionário expressão regular e expressão JavaScript

```
1 ClasseDicionario = "Classe"  ID   [ "ClasseBase"  ID ]  "{"
     PropriedadeDicionario  { ","  PropriedadeDicionario }*  "}" ;
2 PropriedadeDicionario = ID  ":"  ( ValoresValidos |
```

```
3          "[" ClassesEquivalenciaDicionario  "]"  | PENDENTE ) ;
4 ValoresValidos = (TipoComLimiteInfSup | ListaDeValores | Regex | ExpressaoJS | SQL);
5 Regex = "expressao"  STRING ;
6 ExpressaoJS = "expressaoJS ("  STRING  ")" ;
```

Além disso, os dois casos de uso apresentam, como verificação de sua finalização com sucesso, a exibição, em uma tabela, dos dados a serem incluídos na base de dados. Mensagens apontando erro no preenchimento do cadastro são apresentadas caso necessário, mas a mensagem de dados salvos ocorre somente no caso de uso principal, em que salva-se os dados de todas as abas relacionadas à organização. Assim, para verificar a finalização correta dos subcasos de uso, foi utilizado o passo que verifica a exibição de dados em tabela, conforme exibido na Listagem 45. O trecho da DSL que permite tal definição é exibido na Listagem 46.

Listagem 45 – Caso de uso `InserirEnderecoEletronico`-Verificação de dados em tabela

```
1  CasoDeUso InserirEnderecoEletronico(!) {
2    ...
3    cenarioprincipal
4      ...
5      2 "Usuário submete solicitação de inclusão" ...
6      3 "Sistema exibe endereco eletrônico cadastrado em lista" {
7        Armazeno campo enderecoEletronico como enderecoEletronicoCadastrado,
8        Visualizo valor armazenado enderecoEletronicoCadastrado na tabela Linha num 1
             Coluna num 1
9      }
10     ...
11 }
```

Listagem 46 – DSL LUCT–Verificação de dados em tabela

```
1 VerificacaoNaTelaPrincipal = "Visualizo "  ( "mensagem"  ID | "texto = "  STRING | "
    valor armazenado" ID )  [ ValorTela ] ;
2 ValorTela = ( "no campo"  ID | "na tabela"  TabelaLinhaColuna ) ;
3 TabelaLinhaColuna = [ ("num"  INT | "id"  ID) ]
4         [ "Linha"  ("num"  INT | "id"  ID) ]
5         [ "Coluna"  ("num"  INT | "id"  ID) ] ;
```

Por fim, serão analisadas duas outras características do caso de uso `InserirEndereco Eletronico`. O caso de uso permite a inclusão de endereços eletrônicos de categorias diversas, como e-mails, sites e rede sociais. Entretanto, seu comportamento durante o cadastro de e-mails é distinto de quando são cadastrados endereços eletrônicos das demais categorias. Uma das diferenças está nos valores permitidos para o preenchimento do campo endereço eletrônico, como pode ser visto na Listagem 43, em que o domínio foi dividido em classes de equivalência. Além disso, para o cadastro de e-mails, é necessário indicar se aquele é o e-mail preferencial – informação necessária somente para esse cadastro. Tal situação configura que a IU possui uma extensão, ou seja, um trecho exibido somente em determinadas condições. Assim, a característica

foi especificada conforme exibido na Listagem 47. O trecho da DSL que permite a definição de extensões de tela é apresentado na Listagem 48.

Listagem 47 – Caso de uso `InserirEnderecoEletronico`-Extensão de tela

```
1  Tela EnderecosEletronicos {
2    ...
3    Extensao CategoriaEmail{
4      * preferencial:CAMPO_RADIO label ="Preferencial" valores = EnderecoEletronico.
           preferencial
5    }
6  }
```

Listagem 48 – DSL LUCT-Extensão de tela

```
1  Tela = "Tela"  ID  "{"
2     Campo  { ","  Campo }*
3    {ExtensaoTela}*  [MensagensTela]  "}" ;
4
5  ExtensaoTela = "Extensao"  ID  "{"  Campo  { ","  Campo }*  "}" ;
```

Tal comportamento variado por categoria gera a necessidade de indicar, no caso de uso, uma extensão para lidar com o cadastro de e-mails. A Listagem 49 exibe o cenário principal e a extensão criada, que preenche os dados relacionados ao e-mail e retorna ao fluxo principal. O fato de a extensão estar tratando somente e-mails cadastrados como preferenciais (linha 18) será explicado a seguir. As regras da DSL que definem a criação de extensões são apresentadas na Listagem 50.

Listagem 49 – Caso de uso `InserirEnderecoEletronico`-Cenário principal e extensão para cadastro de e-mails

```
1  CasoDeUso InserirEnderecoEletronico(!) {
2    ...
3    cenarioprincipal
4      1 "Usuário informa endereço eletrônico de categoria (geral)" {
5        Informo categoria com EnderecoEletronico.categoria.geral,
6        Informo enderecoEletronico com EnderecoEletronico.enderecoEletronico.geral
7      },
8      2 "Usuário submete solicitação de inclusão" ...
9      3 "Sistema exibe endereco eletrônico cadastrado em lista" ...
10
11   extensoes
12     1 a "Categoria é de email / Cadastro de primeiro email como preferencial"
           extensão de tela CategoriaEmail
13     1a 1 "Usuário informa endereço eletrônico de categoria e-mail" {
14       Informo categoria com EnderecoEletronico.categoria.email,
15       Informo enderecoEletronico com EnderecoEletronico.enderecoEletronico.email
16     }
17     1a 2 "Usuário informa e-mail como preferencial" {
18       Informo preferencial com EnderecoEletronico.preferencial.sim
19     }
20     1a 3 Executa passo 2
21 }
```

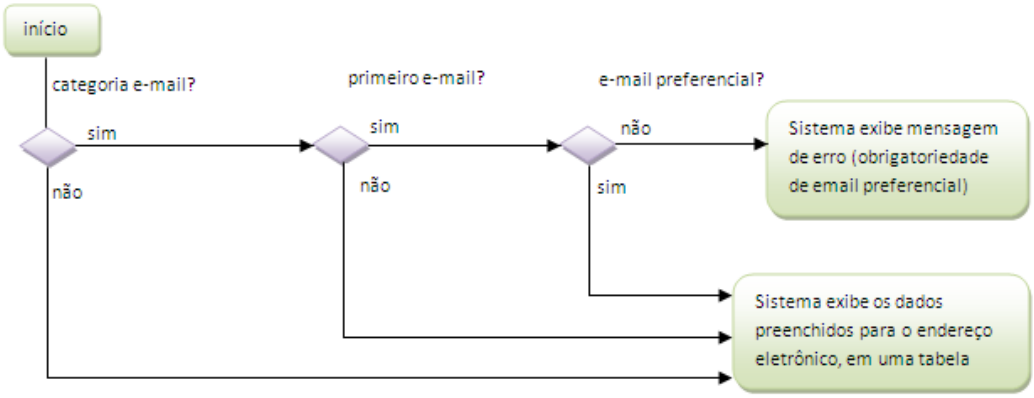
Listagem 50 – DSL LUCT-Extensões

```
["extensoes"  Extensoes ]
Extensoes = CondicaoEPassosFluxoAlt  { ","  CondicaoEPassosFluxoAlt }* ;
CondicaoEPassosFluxoAlt = FluxoAlternativoCondicao
        { PassoUCAlt }+  [ PassoUCAltFinal ] ;
FluxoAlternativoCondicao = INT  LETRA
        [ ("erro" | "erro-teste-padrao") ]  STRING
        [ "extensão de tela"  ID ] ;
PassoUCAlt = INT  LETRA   INT  STRING
        [ "{"  Passo  { ,  Passo }*   "}"       ] ;
PassoUCAltFinal = INT  LETRA  INT  PassoUCAltExecutaFinal ;
PassoUCAltExecutaFinal = Executa passo  INT ;

Passo = Acao | Verificacao | Controle | PENDENTE ;
```

A segunda característica a ser analisada é uma regra de negócio que exige a indicação de um e-mail como preferencial. Isso implica que o primeiro e-mail cadastrado seja sempre indicado como preferencial, sendo os não preferencias permitidos somente a partir do segundo cadastro (Figura 14). Essa variação gera a necessidade de indicar, na extensão que trata o cadastro de e-mails, um segundo nível de extensão, para diferenciar tal comportamento. Porém, devido à DSL permitir expressar as extensões somente no primeiro nível, as extensões precisaram ter suas estruturas transformadas para tal. Assim, foram criadas duas extensões de primeiro nível, sendo a primeira, a apresentada na Listagem 49 – para quando se indica o e-mail como preferencial – e a segunda, a apresentada na Listagem 51 – para quando se indica o e-mail como não preferencial. Na segunda extensão, o fluxo é de erro, uma vez que o SST exibe mensagem indicando a obrigatoriedade de um e-mail preferencial.

Figura 14 – Regra de negócio para o cadastro de endereços eletrônicos



Listagem 51 – Caso de uso `InserirEnderecoEletronico`-Extensão para cadastro de primeiro e-mail como não preferencial

```
CasoDeUso InserirEnderecoEletronico(!) {
  ...
  cenarioprincipal
    ...
  extensoes
```

```
6       ...
7       1 b erro "Categoria é de email/ Cadastro de primeiro email como NÃO preferencial
            " extensão de tela CategoriaEmail
8       1b 1 "Usuário informa endereço eletrônico de categoria e-mail" {
9         Informo categoria com EnderecoEletronico.categoria.email,
10        Informo enderecoEletronico com EnderecoEletronico.enderecoEletronico.email
11      }
12      1b 2 "Usuário informa e-mail como NÃO preferencial" {
13        Informo preferencial com EnderecoEletronico.preferencial.nao
14      }
15      1b 3 "Usuário submete solicitação de inclusão" {
16        Clico Inserir
17      }
18      1b 4 "Sistema mensagem de erro pois deve haver um e-mail preferencial" {
19        Visualizo mensagem msgDeveHaverUmEmailPreferencial
20      }
21
22      ...
23  }
```

Já o cenário de o e-mail ser o segundo a ser cadastrado, não é contemplado nas extensões apresentadas. Para tal, é necessária uma nova extensão que tem, como pré-condição, a obrigatoriedade de já ter sido realizado o cadastro de um primeiro e-mail. Porém, não é possível, por meio da DSL, expressar uma pré-condição somente para um fluxo. Assim, passos que realizam o cadastro do primeiro e-mail foram especificados no início do fluxo da nova extensão, conforme apresentado na Listagem 52. Essa situação é discutida posteriormente nesta seção.

Listagem 52 – Caso de uso `InserirEnderecoEletronico`-Extensão para cadastro de segundo e-mail

```
1   CasoDeUso InserirEnderecoEletronico(!) {
2     ...
3     cenarioprincipal
4       ...
5     extensoes
6       ...
7       1 c erro "Categoria é de email/ Cadastro de segundo email" extensão de tela
            CategoriaEmail
8       1c 1 "Usuário cadastra um primeiro email" {
9         Executo InserirEnderecoEletronico com extensao = "1a"
10        ...
11      }
12      1c 2 "Usuário informa endereço eletrônico de categoria e-mail" {
13        Informo categoria com EnderecoEletronico.categoria.email,
14        Informo enderecoEletronico com EnderecoEletronico.enderecoEletronico.email
15      }
16      1c 3 "Usuário informa se e-mail é preferencial" {
17        Informo preferencial
18      }
19      1c 4 Executa passo 2
20
21      ...
22  }
```

4.2.2 Considerações sobre as Características Atendidas

As características atendidas pela DSL, analisadas nesta seção por meio dos exemplos apresentados, são listadas a seguir:

- Casos de uso com pré e pós-condições, acionadores, cenário principal e extensões;
- Fluxos de extensão que retornam ou não ao fluxo principal, que finalizam com sucesso ou em erro, decorrentes do uso de classes de equivalência, da utilização de extensões de tela e que visam a verificação dos testes de erro padrão;
- Dependência entre casos de uso definida por pré-condições e por fluxos de casos de uso chamadores;
- Acionamento de menus, preenchimento de campos de formulários, acionamento de elementos para submissão de formulário, verificação de exibição de mensagens, verificação de texto em tabela;
- Regras de negócio que utilizam valores mínimos e máximos, lista de valores enumerados, um valor qualquer da lista de opções disponíveis, expressão regular e expressão JavaScript, dependência de valores entre campos – em um ou mais casos de uso – e uso de classes de equivalência;
- Identificação dos elementos da IU por meio de *labels* ou *XPath*, e IUs que apresentam extensão de tela.

Uma questão a se destacar, entre as características atendidas por meio dos recursos oferecidos, trata do uso de expressões JavaScript e da identificação de elementos da IU por meio de expressões XPath. Embora ambos sejam recursos poderosos, que permitem especificar uma grande gama de elementos e situações, seu uso eleva o nível técnico necessário para entendimento e manutenção da especificação.

Como alternativa ao uso do JavaScript em situações simples como a apresentada na Listagem 43 – em que necessitou-se somar um dia a uma data –, considera-se incorporar, à DSL, expressões matemáticas com os operadores básicos, permitindo, como operandos, os tipos básicos tratados na especificação – numérico, texto e data. Além disso, considera-se a possibilidade de tratar expressões JavaScript como parte integrante da DSL – por um mecanismo de extensão – para facilitar sua especificação e realizar validações no conteúdo especificado, em tempo de edição.

Quanto ao XPath, além de sua representação apresentar difícil leitura, seu uso acrescenta detalhes de implementação à especificação, diferentemente da identificação por meio de *labels*.

Entretanto, a possibilidade de identificação de elementos por XPath, além de permitir um grande poder de expressão, possibilita que sistemas de softwares legados – mesmo os que não utilizem *labels* – sejam especificados e testados, sem a necessidade de se alterar ou ajustar o software.

Outra característica que precisou ser exercitada foi a de extensões que necessitam de pré-condições individuais. Como visto na Listagem 52, a extensão exibida (cadastro de um segundo e-mail) necessita que um cenário específico (cadastro de um primeiro e-mail) tenha sido executado anteriormente para que a mesma seja válida naquele momento. Apesar de a solução adotada ter permitido especificar o caso de uso e obter testes corretos, considera-se necessária uma análise detalhada sobre possíveis soluções de especificação para esse tipo de pré-condição (particular a um fluxo de extensão). Tornar esse mecanismo claro e passível de validação – além de, possivelmente, facilitar sua especificação – é considerado um possível trabalho futuro desta pesquisa.

Por fim, outra característica atendida a ser analisada refere-se ao uso de um conjunto de valores fixos – subconjunto do domínio real de um campo – para que um caso de uso pudesse ser executado, situação apresentada na Listagem 35. Uma vez que se objetiva a execução de testes, em algumas situações faz-se necessária a identificação de valores válidos para a correta execução dos testes, que não configuram exatamente, ou totalmente, o domínio de um campo.

No exemplo em questão, utilizou-se um conjunto de valores fixos devido às informações que compõem a lista com os nomes de pessoas ser externa ao SST, não havendo como acessar seus dados para buscar partes válidas de nomes. Além disso, o fato de a lista apresentar um grande volume de informações, por ser a lista de todas as pessoas que possuem relacionamento com a universidade, faz com que a obtenção de um subconjunto de nomes a partir dos valores indicados seja estável e confiável. Assim, a solução de indicação de algumas opções de valores fixos foi satisfatória. Entretanto, em outros casos, considera-se que um conjunto de valores válidos deve ser obtido em um passo – ou caso de uso – anterior, por meio de navegação pelo SST – e armazenamento de um valor apresentado na IU – ou por meio da utilização de um item de dicionário do tipo SQL, recursos oferecidos pela DSL.

Uma questão a se destacar é que, uma vez que o SST é legado, conhecia-se as características dessa lista, o que permitiu definir como se realizaria a obtenção dos dados válidos necessários. Numa situação de o sistema ainda não estar disponível, considera-se que o usuário deixaria essa definição como “PENDENTE”, para ser complementada no futuro.

4.2.3 Considerações sobre Características não Atendidas

Quanto à estrutura dos fluxos, é importante citar que a DSL permite expressar variação nos fluxos de extensão em apenas um nível, o que possibilita atender a maioria dos casos. Entretanto, para casos de uso que possuam um segundo nível de variação (fluxo de extensão

de um fluxo de extensão), é possível transformar a sua estrutura , expressando-a em apenas um nível de extensão, como foi feito para o caso de uso `InserirEnderecoEletronico`, apresentado nas Listagens 49, 51 e 52. Contudo, essa nova estrutura envolve trechos replicados e possivelmente de difícil leitura, quando resulta em um número alto de extensões no primeiro nível, derivadas da transformação. Assim, a identificação dos demais níveis de extensão é proposta como um possível trabalho futuro desta pequisa.

Outra questão não tratada, que não se apresenta nos sistemas de software selecionados para essa análise, mas é presente em outros sistemas de sotware da universidade, é a de fluxos que envolvam espera não ocupada. Nesses fluxos, o caso de uso é realizado em duas etapas, com intervalo de tempo (entre as etapas) superior ao tempo de espera comum da *web*. Esses fluxos envolvem solicitação de tarefas de processamento demorado, muitas vezes envolvendo grandes massas de dados. Assim, solicita-se a tarefa e posteriormente acessa-se os resultados dessas execuções em uma área própria, ficando o usuário livre nesse intervalo.

Apesar de ser oferecido pela DSL o passo `Aguardar` – que possibilita que o teste aguarde por algum tempo ou até que algum elemento seja exibido na tela – essa é uma funcionalidade voltada para esperas curtas, devido, inclusive, ao mecanismo de *timeout* existente nos navegadores. Assim, considera-se como um possível trabalho futuro o tratamento desses fluxos, que passaria pela indicação do trecho que inicia a segunda etapa como um trecho de espera não ocupada. Esse trecho seria então executado periodicamente, por meio de um mecanismo de agendamento de tarefas, até que ocorresse com sucesso, para que só então o restante do caso de uso fosse retomado.

Por fim, uma vez que a DSL interage somente com elementos HTML, não são abrangidos conteúdos em outros meios, como por exemplo, em arquivos PDF. Também não são tratados passos que realizem *downloads* ou *uploads* de arquivos, assim como a manipulação de elementos por meio do mecanismo *drag and drop*. Além disso, não são tratados passos que verifiquem o envio de e-mails pelo SST, ou que verifiquem características de elementos da interface – como visibilidade, estado habilitado/desabilitado, etc. – verificações úteis para interfaces que se alteram dinamicamente de acordo com o preenchimento de campos. Exceto pelo mecanismo *drag and drop*, todas as características foram encontradas em sistemas internos à universidade. Todos os mecanismos são considerados como possíveis trabalhos futuros, uma vez que esta pesquisa buscou tratar as características e necessidades essenciais que envolvem a interação com interfaces *web*.

Assim, os pontos não atendidos ou atendidos parcialmente pela DSL – identificados durante a análise de sua expressividade – são classificados conforme a seguir:

- Estrutura dos casos de uso (formato, fluxos principal e de extensão):

- Fluxos de extensão com mais de um nível;
- Fluxos de espera não ocupada;

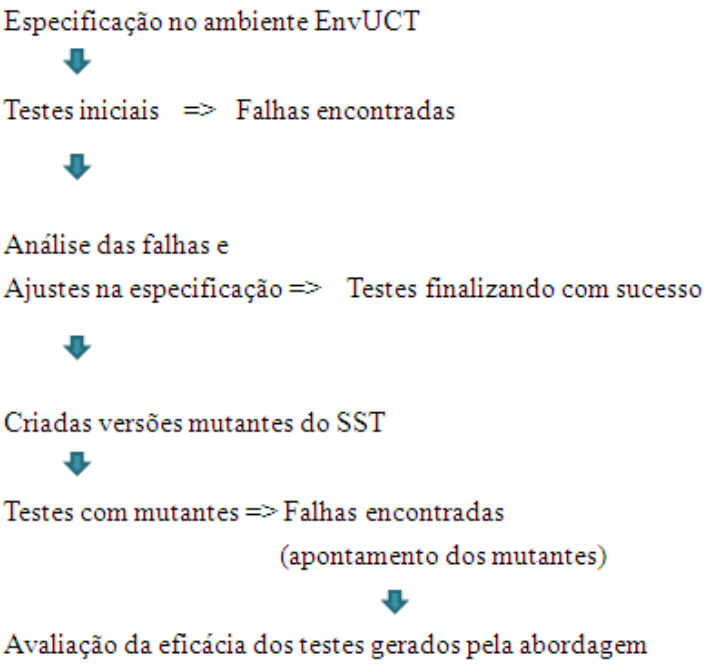
- Passos de detalhamento (interação com o sistema):
  - Manipulação de elementos por meio de mecanismo *drag and drop*;
  - Realização de *downloads* ou *uploads* de arquivos;
  - Verificação de e-mails enviados pelo SST;

- Identificação de elementos da interface:
  - Reconhecimento de elementos não HTML (exemplo: arquivos PDF);
  - Verificação de características de elementos da interface (visibilidade, estado habilitado/ desabilitado, etc.);

4.3 Sistema de Informações de Extensão

Os experimentos com o primeiro software envolveram especificá-lo no ambiente EnvUCT a partir da documentação existente – adaptando a especificação quando necessário – até a obtenção de um conjunto de testes que funcionasse adequadamente. Em seguida, trabalhou-se na verificação da eficácia dos testes gerados, por meio de alterações na especificação e no SST (uso de mutantes). A Figura 15 apresenta o fluxo realizado.

Figura 15 – Fluxo - Experimentos com o Sistema de Informações de Extensão



4.3.1 Seleção dos Casos de Uso

Para este experimento, buscou-se exercitar características comuns aos casos de uso, como fluxos alternativos, regras de negócio para o preenchimento de campos e dependência entre

casos de uso. Assim, selecionou-se casos de uso que permitissem, de forma mais ampla possível, exercitar essas características. Os casos de uso selecionados foram:

- Inserir Curso,
- Inserir Parceria,
- Inserir Local de Abrangência,
- Alterar Curso e Submeter à Aprovação.

4.3.2 Considerações Sobre os Casos de Uso

Esta subseção traz algumas considerações sobre os casos de uso, visando facilitar sua compreensão no decorrer da descrição dos experimentos.

Os casos de uso `InserirCurso` e `AlterarCursoESubmeterAAprovacao` apresentam em seus fluxos o preenchimento dos mesmos campos, com diferença na obrigatoriedade de preenchimento: quando se está inserindo um curso ou alterando seus dados sem submetê-los à aprovação, somente o campo título é obrigatório, enquanto para a submissão, são 23 os campos obrigatórios. Isso permite que o usuário edite os dados várias vezes, acrescentando aos poucos as informações, antes da submissão.

A submissão de um curso exige também a inserção de dados por meio de outras telas. Assim, o caso de uso `AlterarCursoESubmeterAAprovacao` tem como suas pré condições, além do caso de uso `InserirCurso`, os casos de uso `InserirAbrangencia` e `InserirParceria`. As dependências existentes entre os casos de uso participantes desse experimento são:

- `InserirCurso`: sem dependências;
- `InserirParceria`: depende de `InserirCurso`;
- `InserirAbrangencia`: depende de `InserirCurso`;
- `AlterarCursoESubmeterAAprovacao`: depende de `InserirCurso`, `InserirParceria` e `InserirLocalAbrangencia`.

4.3.3 Especificação dos Requisitos no Ambiente EnvUCT

A especificação dos requisitos do Sistema de Informações de Extensão é composta por documentos de texto contendo: a descrição dos casos de uso textuais, das interfaces de usuário

Figura 16 – Documentação do sistema de gerenciamento de atividades de extensão (trechos dos casos de uso, das regras de negócio e das mensagens exibidas pelo sistema)

1. Sistema apresenta a interface *Inserir Curso Aba Descrição* (**RN24, RN35**).
2. Ator
   2.1 Preenche os campos (**RN18, RN19, RN21, RN22, RN23, RN27, RN28, RN32, RN33, RN68, RN77, RN78**).
   2.2 Clica em *Salvar e Avançar*.
3. Sistema:
   3.1 Valida os dados (**RN01**).
   3.2 Insere o Registro.
   3.3 Abre a próxima aba, interface *Ação de Extensão Aba Equipe* (**RN24, RN34**).
   3.4 Apresenta a mensagem **MSG04**.

RN	Número	Regras de Negócio
RN	01	Campos obrigatórios devem ser preenchidos.
RN	02	Todos os campos cujo o conteúdo é uma data, esta data deve corresponder a uma data válida. Formato: DD/MM//AAAA. DD de 1 a 31, MM de 1 a 12.
RN	03	Todas as interfaces que possuem campos como *Data de Início* e *Data de Término* deve-se verificar se a *Data de Término* não é anterior a *Data de Início*.
RN	04	Todos os e-mails cadastrados devem corresponder a um a um formato de e-mail válido. Formato: (caracteres alfanuméricos especiais) + @ + (caracteres alfanuméricos).(caracteres alfanuméricos)

MSG	Número	Tipo	Texto
MSG	01	Erro	Não é possível excluir a Categoria selecionada pois existem Ações de Extensão vinculados a ele.
MSG	02	Alerta	Carga horária não pode ser inferior a 180 horas
MSG	03	Alerta	Carga horária não pode ser inferior a 8 horas
MSG	04	Sucesso	Dados da Descrição inseridos com sucesso, prossiga agora informando os dados de sua Equipe de Trabalho.

Figura 17 – Documentação do sistema de gerenciamento de atividades de extensão (trecho da descrição das interfaces de usuário)

Nome	Valores Válidos	Formato	Tipo	Retrições	Máscara
Registro	Caracteres alfanuméricos e especiais	Até 12 caracteres	Leitura	-	-
Título	Caracteres alfanuméricos e especiais	Até 400 caracteres	input	Obrigatório/ Alterável	-
Data de início	Caracteres numéricos	Até 8 dígitos	input	Obrigatório/ Alterável	DD/MM/YYYY
Previsão de Término	Caracteres numéricos	Até 8 dígitos	input	Obrigatório/ Alterável	DD/MM/YYYY
Órgão competente	Câmara Departamental, Câmara de Extensão, Cenex, Congregação, Órgão Equivalente	-	Select	Obrigatório/ Alterável	-
Linha de extensão	Lista das linhas de extensão	-	Select	Obrigatório/ Alterável	-

(que trazem também informações do dicionário de dados), das regras de negócio, e a lista de mensagens do sistema. Trechos desses documentos são exibidos nas Figuras 16 e 17.

A informação contida nos documentos foi transposta para o ambiente (e formato) propostos nesse trabalho. Especificou-se, como elementos do dicionário de dados, `Parceria`, `Abrangencia`, `AcaoExtensao` e `Curso`, sendo `Curso` o conjunto de informações espe-

cíficas de um curso, enquanto `AcaoExtensao`, as informações comuns a todos os tipos de ações de extensão (cursos, eventos, prestações de serviço, programas e projetos). Quanto às telas, foram cadastradas: `TelaAbrangencia`, `TelaParceria`, `TelaInserirCurso` e `TelaAlterarCurso`, além da tela de *login*.

Os fluxos alternativos para os casos de uso foram identificados a partir das regras de negócio. Identificou-se a situação de variação de valores por meio de classes de equivalência para o campo *carga horária* – que apresenta um limite mínimo para cursos de *aperfeiçoamento* diferente de cursos com outras caracterizações. Assim, especificou-se o domínio com as classes de equivalência e fluxos alternativos para tratar essa variação, nos casos de uso "Inserir Curso" e "Alterar Curso e Submeter à Aprovação". Além desses, para todos os casos de uso foram criados fluxos alternativos (um por caso de uso) para testar as situações de erro padrão no preenchimento dos campos.

4.3.4 Execução dos Testes / Ajustes na Especificação

Os testes foram gerados e executados, apresentando, como resultado, falhas em todos os casos de teste. A análise dos *logs* mostrou um erro em comum, no caso de uso `InserirCurso`, devido a incoerências na especificação (campos que existiam na documentação, mas não na interface do sistema). Como os demais casos de uso dependem desse caso de uso (por tê-lo como pré-condição), todos apresentaram o mesmo erro e tiveram sua execução interrompida. Optou-se então por dar sequência aos testes executando somente os casos de teste gerados para o caso de uso `InserirCurso`, sendo os demais casos de teste incorporados após a constatação de que esse grupo executava com sucesso. A seguir são descritas e analisadas todas as falhas encontradas pelos testes, assim como sua correção.

As falhas apontadas na execução dos primeiros testes indicavam que campos que se tentava preencher não existiam na IU de Inserção de Curso. Verificou-se, a cada erro, o campo em questão, por meio da conferência dos *labels* apresentados na tela do sistema, uma vez que foi essa a forma escolhida para a identificação dos campos na especificação.

Foram verificadas e acertadas 13 ocorrências de falha, sendo sete identificadas como *labels* que aparentemente foram alterados no sistema, mas não na especificação; três identificados como *labels* que apesar de estarem corretos visualmente, por conterem quebra de linha em seu texto, somente parte dele foi indicada como o elemento *label* (constatado verificando o código-fonte HTML); e três identificadas como campos que realmente não existiam na tela, provavelmente por terem sido excluídos do sistema, mas não da documentação.

Além desses pontos, verificou-se divergência do sistema com a documentação na definição do limite máximo, em cinco campos. Em todos os casos, o limite máximo permitido pelo sistema (visível por meio de contadores de caracteres restantes, ao lado dos campos) era

maior que o indicado na documentação. Os limites foram acertados na especificação. Após esses acertos, o caso de teste do tipo `TP_VALORES_VALIDOS_MAX` (que preenche todos os campos com os valores máximos) apresentou falha em sua execução. O *printscreen* da tela gravado na execução mostrou uma mensagem de erro indicando que uma situação inesperada ocorreu. Testando-se manualmente o preenchimento desse campo com tamanhos variados, concluiu-se que o limite máximo permitido estava entre o valor da documentação e o valor exibido no contador da tela. Esse comportamento foi considerado um *bug* encontrado no sistema.

Outro ponto em que verificou-se divergência do sistema com a documentação foi na definição de cinco mensagens apresentadas pelo SST, resultando em falha nos casos de teste que verificavam sua exibição. Por meio do *log* e *printscreen* da tela gravados no momento em que os testes falharam, pôde-se recuperar os textos corretos, corrigindo-os na especificação. Em três casos o texto da mensagem exibida era divergente do contido na especificação, sendo nos demais, o mesmo texto, com exceção do nome do campo, que divergia do *label* exibido na tela. Para o segundo caso adicionou-se mensagens de erro individuais para esses campos na especificação, deixando de utilizar para os mesmos a mensagem de erro padrão.

Outro ponto de falha ocorreu em casos de teste dos tipos de teste padrão `MAX+1` e `MIN-1` (que testam o comportamento do sistema quando é informado valor maior/menor que o limite máximo permitido para um campo). Em vez de o SST apresentar mensagem de erro para a situação, o cadastro foi finalizado com sucesso. Quanto ao limite máximo, concluiu-se que a situação se deveu ao fato de todos esses campos terem seu limite relacionado ao número de caracteres digitados, e o SST não permitir a digitação de mais caracteres, quando se chega ao limite máximo. Quanto ao limite mínimo, concluiu-se que a situação se deveu ao fato de os campos (exceto o campo carga horária) terem seu limite mínimo igual a zero, e o SST não permitir a digitação de caracteres não numéricos, o que fazia com que fosse ignorado o sinal negativo, restando somente o caractere "1", que é um valor permitido. Foram então criados parâmetros para desabilitar esses tipos de testes, de acordo com o comportamento do SST.

Além dessas ocorrências, ao contabilizar manualmente o número de campos apresentados nas IUs e os presentes na especificação, verificou-se que havia dois campos a mais em uma IU, que não existiam na documentação. Apesar de esses campos serem obrigatórios, sua ausência não resultou em erro nos testes, uma vez que a IU os exibia com valores *default*. Ou seja, eles passaram – e continuariam passando – despercebidos pelos testes. Os campos foram adicionados à especificação, com seu tipo e limites inferidos pela especificação de outros campos similares (todos campos de preenchimento de número de bolsas, sendo um por tipo de patrocínio). Essa situação será discutida posteriormente, buscando apontar possíveis soluções.

Após essas modificações e regerados os testes, uma nova execução resultou em sucesso para todos os casos de teste. Neste ponto, optou-se por indicar, em todos os pontos de invocação

do caso de uso `InserirCurso` como pré condição, uma execução de cenário que fosse a menor possível. Buscou-se, com isso, evitar que falhas no primeiro resultassem em falhas no segundo, devido à dependência. Uma vez que a sequência dos experimentos se dará com inserções de mudanças (na especificação e no SST) e análise dos resultados da execução de testes, esses efeitos colaterais poderiam restringir o número de situações distintas observadas.

Além disso, a indicação de um cenário nesses pontos de invocação permitirá simplificar a apresentação dos casos de teste, resultados e análise dos experimentos. Como descrito na Seção 3.3.2.1, a geração de cenários completos de execução envolve a combinação de todos os cenários (de sucesso) dos casos de uso invocados. Assim, sem a indicação de um cenário na invocação de `InserirCurso`, são gerados dois sequenciamentos, uma vez que há dois cenários de sucesso para `InserirCurso` (devido ao fluxo alternativo para tratar a variação de carga horária por caracterização), o que faz dobrar o número de casos de teste de `InserirParceria`, `InserirAbrangencia` e `AlterarCursoESubmeterAAprovacao`. Como verificou-se que essa variação na execução não apresentou resultados diferenciados, conclui-se pela invocação de um único cenário de `InserirCurso` na continuidade dos testes, sendo esse um cenário curto, pelos motivos expostos acima.

Assim, criou-se para o caso de uso `InserirCurso` a extensão "1a" (Figura 18), em que, ao invés de se preencher todos os campos da tela, preenche-se somente o campo *título*, que é obrigatório. E indicou-se essa extensão nos pontos de invocação (Figura 19), fazendo com que, em todos os casos, seja executado o cenário que passa pela extensão. Nota-se que o cenário criado é equivalente ao teste padrão `TP_SO_OBRIGATORIOS`. Como não é possível indicar um tipo de teste nas invocações, a extensão (e o cenário) precisaram ser criados. A possibilidade de tratar essa situação – sem a necessidade de criar tal extensão – é considerada um possível trabalho futuro dessa pesquisa.

Figura 18 – Extensão criada para o caso de uso `InserirCurso`

```
cenarioprincipal
    1 "Usuário informa campos do novo curso (informações gerais de ação de extensão)" {
    2 "Usuário informa que novo curso é de caracterização geral e a carga horária" {
    3 "Usuário informa demais campos do novo curso (informações específicas de cursos)" {
    4 "Usuário solicita inclusão do novo curso" {
    5 "Sistema exibe mensagem de sucesso" {

extensoes
    1a "Usuário informa somente o título do curso"
    1a 1 "Usuário informa o título do curso" {
        Informo titulo ,
        Informo[dataInicio,previsaoTermino]
    }
    1a 2 Executa passo 4,
```

A especificação final para os casos de uso utilizados nesse experimento é apresentada no Apêndice C e a lista de casos de teste, no Apêndice E.

Figura 19 – Invocação do caso de uso `InserirCurso` indicando extensão

```
CasoDeUso InserirParceria (!) {
    execucaopropria S
    ator Docente
    acionador {
        Acesso link LinkTelaParcerias
    }
    precondicao "Usuário insere novo curso" {
        Executo InserirCurso  com extensao = "1a"
    }
```

4.3.5 Versão Levemente Divergente da Especificação

Uma versão da especificação levemente divergente da versão final obtida na seção anterior foi construída, para verificar se os pontos divergentes seriam observados pelos testes gerados pela abordagem proposta. Os seguintes pontos foram modificados:

- Carga horária: Alterada de 180 para 120 horas a carga horária mínima para cursos de aperfeiçoamento, e de 8 para 6 horas a carga horária mínima para os demais cursos;
- Data início: Alterada de 01/01/1950 para 01/01/1960 a data de início mínima para os cursos.

Os testes resultaram em falhas em 8 casos de teste, derivados dos casos de uso `InserirCurso` e `AlterarCursoESubmeterAAprovacao`:

- um total de quatro casos de teste do tipo de teste padrão `VALORES_VALIDOS_MIN` (que preenche todos os campos da tela com os valores mínimos). Os *logs* mostraram o preenchimento do campo carga horária com os novos valores mínimos, e os *printscreens* das telas, mensagens de erro indicando valores incorretos para o campo. Como os testes deveriam finalizar com a mensagem de sucesso de cadastro realizado, os mesmos falharam.
- um total de quatro casos de teste do tipo de teste padrão `VALOR_INVALIDO_MIN_MENOS_UM` [dataInicio] (que preenche o campo data início com um valor inferior ao mínimo). Nesses testes, o cadastro finalizou com sucesso, em vez de ser exibida mensagem que apontava o valor abaixo do limite, para o campo.

Analisando a lista de casos de testes, conclui-se que os que apresentaram falha na execução foram exatamente o subconjunto esperado para tal: Todos os casos de teste do tipo `VALORES_VALIDOS_MIN` e `VALOR_INVALIDO_MIN_MENOS_UM`[dataInicio], derivados dos casos de uso `InserirCurso` e `AlterarCursoESubmeterAAprovacao`.

Observa-se que os casos de teste do tipo de teste padrão `VALIDOS_QUAISQUER` e os casos de teste do tipo de teste padrão `SO_OBRIGATORIOS` poderiam ter apontado o mesmo

erro do 1^o grupo, caso em alguma execução o valor aleatório gerado para o campo carga horária houvesse caído na faixa proibida para o campo, o que não foi o caso nessa execução. De qualquer forma, esses testes têm por finalidade principal exercitar outros pontos, diferentemente dos que indicaram a falha.

Conclui-se, portanto, que os testes gerados pela abordagem proposta apontaram corretamente os pontos divergentes na especificação.

4.3.6 Versões do SST com Uso de Mutantes

Prosseguindo na avaliação da eficácia dos testes gerados pela abordagem proposta, esta seção trata de experimentos que utilizaram versões do SST com emprego de mutantes, objetivando verificar se os mesmos seriam detectados pelos testes.

A geração das versões com o uso de mutantes baseou-se no modelo de falhas de caso de uso descrito na Subseção 2.3.3.3, que define operadores mutantes para casos de uso. Utilizou-se o operador mutante de caso de uso: "Substituição de regras de validação ou da admissão de um dado como incorreto", por meio da inversão de operadores condicionais que realizam a validação dos dados de entrada. Foram inseridas três mutações para o caso de uso `InserirParceria` (`IP`), três mutações para o caso de uso `InserirLocaldeAbrangencia` (`ILA`) e três mutações compartilhadas entre os casos de uso `InserirCurso` (`IC`) e `AlterarCursoESubmeterA Aprovacao` (`ACSA`).

Para a realização dos testes, foram geradas versões mutantes do SST utilizando uma mutação por vez, visando possibilitar a execução e análise isolada dos efeitos de cada mutação nos resultados, sem possíveis efeitos colaterais entre os trechos com mutações.

Executados os testes com as versões mutantes, os resultados esperados (novos comportamentos) foram observados na maior parte dos casos. A Tabela 2 apresenta um comparativo entre as situações esperadas (`A, B, C, D, E, F, G, - e P`) e as situações ocorridas, que em alguns casos se mostraram diferente do esperado (`A1, A2, B1 e B2`). Os pontos indicados com traço são pontos em que a mutação não gerou mudança de comportamento. Os pontos indicados por "P" são pontos em que que o caso de uso deixa de ser executado, devido a ocorrência de falha em momento anterior à sua execução.

Analisando as novas situações apresentadas, constatou-se que as mesmas decorreram de desdobramentos da execução do código, não identificados no levantamento inicial, caracterizando efeitos colaterais a partir dos defeitos inseridos.

Assim, conclui-se que os testes gerados apresentaram boa eficácia no apontamento das mutações inseridas no SST, uma vez que cada mutação – e seus desdobramentos – foi identificada

Tabela 2 – Execução das versões mutantes – Situações esperadas x ocorridas

	Caso de Uso							
Versão Mutante	**IP**		**ILA**		**IC**		**ACSA**	
IP-Mut1	A B	A B1	–	–	–	–	P	P
IP-Mut2	A B	A –	–	–	–	–	P	P
IP-Mut3	A B	A B	–	–	–	–	P	P
ILA-Mut1	–	–	C	C	-	–	P	P
ILA-Mut2	–	–	A B	A B2	-	–	P	P
ILA-Mut3	–	–	C	C	-	–	P	P
IC/ACSA-Mut1	–	P	-	P	– –	A1 A2	A B	P P
IC/ACSA-Mut2	–	–	–	–	–	–	D E	D E
IC/ACSA-Mut3	-	-	-	-	-	-	F G	F G

por ao menos um caso de teste. Além disso, constatou-se que as informações providas como resultado das execuções foram úteis na compreensão dos motivos das falhas, auxiliando na identificação de possíveis defeitos no software.

4.3.7 Observações sobre o Experimento

Os experimentos realizados permitiram exercitar as seguintes características da abordagem proposta:

- Dependência entre casos de uso definida por pré-condições;
- Regras de negócio que utilizam valores mínimos e máximos, lista de valores enumerados, um valor qualquer da lista de opções disponíveis, expressão JavaScript, classes de equivalência e dependência entre campos;
- Utilização de mensagens globais e específicas;
- Fluxos alternativos que retornam ou não ao fluxo principal;
- Testes padrão com valores nos limites mínimo e máximo, valores abaixo do limite, testes com preenchimento somente de campos obrigatórios e sem o preenchimento de algum campo obrigatório.

4.4 Sistema de Ex-Alunos

O segundo software participante dos experimentos, utilizado para gerenciar informações e atividades de ex-alunos de uma universidade, comporta dados como premiações, cargos, organizações e setores de atuação.

4.4.1 Seleção dos Casos de Uso

Foram selecionados para o experimento três casos de uso caracterizados por conter subfuncionalidades CRUD: “Gerir Cargo”, “Gerir Curso” e “Gerir Instituição”. Por apresentarem desdobramentos e resultados similares, optou-se por descrever o processo de somente um deles – o caso de uso “ Gerir Cargo”. Essa seleção visou exercitar casos de uso do nível de Resumo para gerenciar subfuncionalidades CRUD de uma entidade.

4.4.2 Especificação dos Requisitos no Ambiente EnvUCT

A documentação do sistema – composta por casos de uso textuais, descrição das IUs e lista de opções de menu do sistema – foi inserida no ambiente proposto. Informações sobre as extensões dos casos de uso foram obtidas a partir das regras de negócio, que vieram acopladas aos casos de uso.

As Listagens 53 e 54 exibem dois casos de uso especificados no ambiente EnvUCT: `AlterarCargo` e `PesquisarCargo`, subfuncionalidades de `GerirCargos`.

Listagem 53 – Caso de uso `AlterarCargo`

```
1  CasoDeUso  AlterarCargo (!) {
2    execucaopropria N
3    ator Secretaria
4    acionador {
5      Acesso link GerirCargos
6    }
7    cenarioprincipal
8      1 "Usuário pesquisa o [cargoASerAlterado]" {
9        Executo PesquisarCargo com cargoPesquisado = "{:cargoASerAlterado}"
10     },
11     2 "Usuário clica em [Alterar], na linha do cargo pesquisado " {
12       Clico Alterar
13     },
14     3 "Sistema [exibe] tela [Alterar Cargo]" {
15       Visualizo texto = "Detalhes de Cargo"
16     },
17     4 "Usuário [informa] o campo [Nome] com o nome do novo cargo" {
18       Informo Nome com valor armazenado novoNomeCargo
19     },
20     5 "Usuário [clica] em [Salvar]" {
21       Clico Salvar
22     },
23     6 "Sistema [exibe mensagem] de sucesso" {
24       Visualizo mensagem msgDadosSalvos
25     },
26     7 "Usuário verifica que cargo foi alterado" {
27       Executo PesquisarCargo com cargoPesquisado = "{:cargoASerAlterado }" com
             extensao = "3a",
28       Executo PesquisarCargo com cargoPesquisado = "{:novoNomeCargo}"
29     }
30   extensoes
31     6 a "Testes padrao"
```

```
32        6a 1 "Sistema [exibe mensagem] de erro" {
33           Visualizo uma mensagem de erro de testes padrão
34        }
35  }
```

Listagem 54 – Caso de uso `PesquisarCargo`

```
1   CasoDeUso  PesquisarCargo (-) {
2     contexto "Pesquisar cargo"
3     execucaopropria N
4     ator Secretaria
5     acionador {
6           Acesso link GerirCargos
7     }
8     cenarioprincipal
9       1 "Usuário informa o cargo a ser pesquisado" {
10         Informo Nome com valor armazenado cargoPesquisado
11      },
12      2 "Usuário solicita a busca" {
13        Clico Buscar
14      },
15      3 "Sistema exibe o cargo pesquisado na lista de resultados." {
16        Visualizo valor armazenado cargoPesquisado
17      }
18    extensoes
19      3 a "Cargo não cadastrado"
20        3a 1 "Sistema [exibe mensagem] de item não encontrado" {
21           Visualizo mensagem msgNenhumItemEncontrado
22      }
23  }
```

Cada caso de uso que representa uma funcionalidade CRUD foi tratado como sendo de nível de "Objetivo do Usuário", enquanto o caso de uso `GerirCargos`, como nível "Resumo" (Listagem 55). Casos de uso do tipo "Resumo" descrevem de que forma os casos de uso de nível "Objetivo de Usuário" operam e se relacionam, e são úteis para se descrever processos. No contexto desse trabalho, podem ser utilizados para orquestrar as chamadas aos casos de uso de nível "Objetivo de Usuário", mantendo uma sequência para a execução dos testes, com maior agilidade e organização. Ao se testar funcionalidades CRUD, por exemplo, é comum a execução, em sequência, das funcionalidades de inclusão, pesquisa, alteração e exclusão. Assim, reaproveita-se os dados cadastrados e finaliza-se com o banco de dados no estado original. Como pode ser observado na Listagem 55, o caso de uso `GerirCargos` faz esse papel. A utilização de casos de uso "Resumo" para tal fim é opcional. Optando-se em não utilizá-los, as operações necessárias à execução de cada caso de uso são obtidas pela execução das pré-condições, que é suprimida quando o caso de uso é invocado por outro.

Listagem 55 – Caso de uso `GerirCargos`

```
1   CasoDeUso GerirCargos (+) {
2     contexto "Gerenciar os cargos do sistema (Exercitar CRUDs)"
3     execucaopropria S
4     ator Secretaria
```

```
5  cenarioprincipal
6    1 "Usuário cria cargo"  {
7      Armazeno "CargoABC" como nomecargo,
8      Executo CriarCargo com cargoASerIncluido = "{:nomecargo}"
9    },
10   2 "Usuário tenta criar cargo existente" {
11     Executo CriarCargo com cargoASerIncluido = "{:nomecargo}" com extensao = "5a"
12   },
13   3 "Usuário solicita exclusão de cargo, mas cancela pedido " {
14     Executo ExcluirCargo com cargoASerExcluido = "{:nomecargo}" com extensao = "4a"
15   },
16   4 "Usuário altera cargo para nova descrição" {
17     Armazeno "CargoDEF" como "novonomecargo",
18     Executo AlterarCargo com cargoASerAlterado = "{:nomecargo}", novoNomeCargo = "{:
           novonomecargo}"
19   },
20   5 "Usuário exclui cargo " {
21     Executo ExcluirCargo com cargoASerExcluido = "{:novonomecargo}"
22   }
23 }
```

4.4.3 Execução dos Testes / Ajustes na Especificação

Foram gerados os *scripts* de teste para os casos de uso e sua execução apresentou quatro erros, decorrentes de incorreções na especificação: um erro referente à identificação incorreta de um campo da interface, e outros três relacionados a mensagens de texto divergentes das apresentadas pelo SST. Algumas dessas incorreções aparentam ser consequência de cópias de textos de casos de uso similares, sem o ajuste posterior. As demais incorreções sugerem a situação dos textos da especificação estarem obsoletos, ou seja, teriam sido modificados somente no SST.

4.4.4 Observações sobre o experimento

Esse experimento permitiu exercitar as seguintes características da abordagem proposta:

- Organização de casos de uso em níveis e utilização do nível de Resumo;
- Passo do tipo Armazenamento;
- Passagem de parâmetro na invocação de casos de uso.

Constatou-se que o cadastro de casos de uso com funcionalidades CRUD, utilizando um caso de uso (do tipo Resumo) para ordenar as funcionalidades e garantir o estado do banco de dados ao final da execução, é trabalhosa e apresenta certa complexidade para um usuário comum, como na passagem de parâmetros entre os casos de uso, com uso de variáveis. Além disso, os parâmetros não são validados, ficando na responsabilidade do usuário indicá-los corretamente.

Conclui-se que a criação de um mecanismo que facilitasse esses testes, baseado em indicações na especificação dos casos de uso que possibilitassem gerar automaticamente o código necessário para exercer o papel apresentado nesse experimento pelo caso de uso Resumo, seria bastante útil.

Verificou-se que as funcionalidades do primeiro grupo (com ações do tipo CRUD) apresentavam um mesmo padrão de navegação, o que levou à constatação de que um mecanismo que permitisse a parametrização dos casos de uso seria interessante para esse conjunto.

Apesar de os experimentos apontarem complexidade na utilização de casos de uso do tipo Resumo, considera-se que seu uso é importante para especificar fluxos complexos ou que exercitem situações específicas, que passem por mais de um caso de uso. A simplificação (ou automatização) desses, como citada acima, mostra-se desejável especialmente quando se tratando de casos de uso similares, o que resultaria, também, em casos de uso Resumo similares. Em geral, sistemas de software tendem a apresentar um conjunto de CRUDs com mesma navegação.

5 TRABALHOS RELACIONADOS

Casos de uso proveem uma armação que conecta informações em diferentes partes dos requisitos, ajudando a ligar informações de perfis de usuário, regras de negócio ou requisitos de formato de dados (COCKBURN, 2005). Assim, diversas pesquisas trabalham na análise de casos de uso, buscando extrair informações para derivação de novos artefatos. Em Fortuna, Borges et al. (2008), a partir dos fluxos de informação de casos de uso são derivados "info cases"(ICs), contendo somente a troca de dados entre os atores e o sistema de software, e um dicionário de itens. A partir dos ICs, gera-se um modelo de domínio, por meio de regras semi-formais. Já Savic et al. (2011) apresenta uma linguagem para especificação de casos de uso, baseada em Xtext (XTEXT, 2014), denominada SilabReq. A partir dos casos de uso são gerados o modelo de domínio, a lista de operações do sistema, o modelo de caso de uso UML e os diagramas de estado, de atividades e de sequência.

O trabalho proposto aproxima-se do artigo apresentado em Fortuna, Borges et al. (2008), por abordar de forma integrada casos de uso e modelo de domínio, e propor uma formalização que permite identificar os dados trocados entre os atores e o sistema. Assim como Savic et al. (2011), esta dissertação propõe uma linguagem para a especificação de casos de uso, utilizando como base o *framework* Xtext. Porém, como o foco é a geração de testes executáveis, a proposta aborda um conjunto de elementos divergente do explorado em SilabReq.

Fortuna, Borges et al. (2008) destaca a importância de se ter um ambiente com casos de uso e modelo de domínio integrados, para se manter a consistência entre os mesmos. Em Somé (2006) são recebidos, como entrada, casos de uso textuais, descritos em uma linguagem controlada, e um modelo de domínio. Verificam-se inconsistências dos dados, como entidades descritas com mais de um nome, operações no caso de uso que não se refiram a uma Operação de Conceito no modelo, entre outras. São gerados cenários com a composição dos casos de uso, e simulada sua execução em uma máquina de estados, em uma ferramenta chamada UCEd. Assim como em Somé (2006), este trabalho propõe um modelo de domínio com informações de valores possíveis, porém com formas variadas em sua declaração. Outro ponto em comum é a geração de cenários com o sequenciamento dos casos de uso, porém, na abordagem aqui apresentada, visa-se a execução real de testes.

De acordo com Smialek (2005), para contemplar as necessidades de todos os papéis envolvidos em um projeto de desenvolvimento de software é necessário especificar casos de uso em várias notações. Então, é proposto um conjunto de notações, baseadas em texto estruturado, diagramas de interação e de atividades, com regras definidas para transformação de um para outro. Savic et al. (2012) propõe a divisão da especificação de casos de uso em níveis de diferentes

graus de abstração, para atender às necessidades de pessoas com diversos papéis – desde usuários finais, engenheiros de requisitos e analistas de negócio, até projetistas, desenvolvedores e *testers*. Já em Sinha, Sutton e Paradkar (2010), casos de uso são especificados em um ambiente chamado Text2Test, baseado no *framework* Xtext, em que são realizadas validações em tempo de edição, de acordo com o perfil do usuário.

A proposta aqui apresentada, assim como Sinha, Sutton e Paradkar (2010) e Savic et al. (2012), utiliza um ambiente baseado em Xtext, com validações, para a especificação de casos de uso, mas diferencia-se por objetivar a execução de testes. E assim como nos trabalhos citados, a metodologia aqui sugerida trabalha diferentes níveis de abstração nos casos de uso. Cada passo do caso de uso no nível abstrato é seguido por seu detalhamento, onde são inseridos detalhes da interação com a IU. O nível abstrato indica o que é feito, enquanto o nível de detalhamento indica como o passo deve ser realizado. A visualização do nível de detalhamento é opcional, podendo ser expandido ou resumido, para melhor atender ao perfil do usuário que esteja acessando a especificação.

Diversos trabalhos buscam utilizar casos de uso para geração de cenários ou casos de teste. Em Nebut et al. (2006), é proposta uma abordagem para geração de cenários de testes a partir de casos de uso com contratos formalizados em OCL e diagramas de sequência. Em Some e Cheng (2008) são gerados cenários, com sequenciamento baseado nas pré e pós-condições de casos de uso textuais, descritos em linguagem controlada. Em Gois, Farias e Oliveira (2010) são gerados casos de teste a partir de casos de uso, que são tratados em diagramas próprios, chamados TSD, os quais trazem como diferencial a possibilidade de se associar os dados de testes, organizados por classes de equivalência, aos passos oriundos de uma especificação de caso de uso. Já em Gutiérrez et al. (2008) são gerados diagramas de atividades a partir de casos de uso, e desses, os cenários possíveis para os casos de uso, por meio de uma ferramenta denominada TestGen.

Diferentemente da abordagem aqui proposta, a formalização necessária em Nebut et al. (2006) eleva o nível técnico necessário para entendimento e manutenção dos casos de uso. Apesar do sequenciamento entre casos de uso ser um fator em comum entre a presente abordagem e a de Some e Cheng (2008), os cenários apresentados no artigo estão em alto nível de abstração, diferentemente do que se propõe nesta dissertação. Ademais, apesar de a metodologia proposta tratar valores de teste organizados por classes de equivalência assim como em Gois, Farias e Oliveira (2010), e de utilizar a ferramenta apresentada em Gutiérrez et al. (2008) para a geração de cenários de casos de uso, diferencia-se por gerar, desses cenários, *scripts* de teste executáveis.

Alguns trabalhos geram testes a partir de artefatos correlatos aos casos de uso. Em Luna, Rossi e Garrigós (2011) é apresentada Webspec, uma linguagem visual com diagramas, utilizada para capturar navegação, interação e características de interfaces para aplicações *web*. Webspec

foi construída como um *plug-in* no Eclipse e utiliza *mockups* (rascunhos de IU) para simulação. A ferramenta possibilita a geração de código, além de testes para o Selenium. Já Huang e Chen (2006) propõe a ferramenta WASATT, para geração de testes para aplicações web a partir de um diagrama de atividades estendido, contendo informações sobre as IUs. O resultado são *scripts* de teste utilizando a API HttpUnit[1].

Assim como em Luna, Rossi e Garrigós (2011), o presente trabalho gera testes executáveis para o Selenium utilizando um ambiente no IDE Eclipse. Porém, diferencia-se dos trabalhos citados por basear-se em uma linguagem descritiva, em vez de em diagramas.

Alguns trabalhos utilizam-se, para a identificação dos elementos contidos em estruturas textuais, de processamento de linguagem natural (PLN). Jiang e Ding (2011) apresenta um *framework* para geração automática de casos de teste a partir de casos de uso textuais, escritos no formato Inteiramente Completo, apresentado em Cockburn (2005). Os textos dos passos dos casos de uso são analisados por um parser de linguagem natural, que identifica a estrutura gramatical das sentenças, escritas em um dos formatos pré-estabelecidos. Como resultado, são identificados a ação executada no passo, o ator, o receptor, e a pré-condição relacionada, caso exista. Então é gerada uma EFSM (máquina de estados finitos estendida), da qual são derivados os casos de teste. Em Pedemonte, Mahmud e Lau (2012) e Thummalapenta et al. (2012), casos de testes manuais são transformados em casos de testes automatizados, semiautomaticamente, por meio de aprendizado de máquina e PLN. Havendo ambiguidade durante o processamento das sentenças, é solicitada a intervenção do usuário, sendo as escolhas armazenadas para auxiliar em decisões futuras.

Como similaridade entre Jiang e Ding (2011) e o trabalho aqui apresentado, tem-se o formato escolhido para os casos de uso, e sua transformação em casos de teste. Porém, as duas abordagens divergem na forma da geração dos cenários de teste, sendo uma por meio de uma EFSM e outra, por meio de um diagrama de atividades. Além disso, o trabalho proposto diverge dos demais por não utilizar aprendizado de máquina ou PNL em seu processo, e por obter testes executáveis, além de, diferentemente de Pedemonte, Mahmud e Lau (2012) e Thummalapenta et al. (2012), trabalhar a partir de casos de uso e não de casos de testes.

Já outros trabalhos propõem linguagens controladas para a identificação de elementos nos casos de uso textuais. Matos e Sousa (2010) apresenta uma ferramenta capaz de gerar testes funcionais a partir de cenários de casos de uso especificados em uma linguagem natural controlada, em que os fluxos são compostos por ações (clicar, preencher, etc.) realizadas por um ator em um objeto. Além dos casos de uso, são requeridos um glossário com o mapeamento das entidades da aplicação e especificações da IU. A especificação é realizada em um ambiente

[1] HttpUnit: API utilizada para testes de interfaces web <http://httpunit.sourceforge.net/>

próprio baseado em ANTLRWorks[2], que permite a validação do texto em tempo de edição. A partir desses artefatos são gerados testes funcionais para o Selenium, além de protótipos de páginas HTML.

Como similaridade entre o trabalho aqui apresentado e Matos e Sousa (2010), tem-se o uso de linguagem controlada para os requisitos, o uso de glossário e especificação de IUs, complementares aos casos de uso. Além disso, ambos oferecem um editor para validar a especificação em tempo de edição e geram scripts que serão executados pelo Selenium. Como divergências, tem-se a não realização, em Matos e Sousa (2010), da geração de valores para os testes com base na especificação, explorando verificações do comportamento do SST diante da variação de valores válidos ou quando são informados valores inválidos. Além disso, em Matos e Sousa (2010), os passos dos casos de uso são compostos diretamente pelas ações que acessam a IU, diferentemente da metodologia proposta, em que se utiliza um nível abstrato seguido pelo seu detalhamento. Ademais, em Matos e Sousa (2010) são especificados os cenários de casos de uso, em vez de esses serem gerados a partir da estrutura do caso de uso.

Em sua dissertação de mestrado, Pessoa (2011) apresenta uma ferramenta em que os casos de uso são cadastrados em um formulário, com opções controladas e pré-definidas para os passos dos fluxos de eventos. Desses passos são gerados, de forma automática, *scripts* de teste, que podem ser posteriormente executados pela ferramenta, que faz integração com o Selenium.

Assim como em Pessoa (2011), a proposta aqui apresentada gera *scripts* de teste automaticamente a partir dos fluxos de casos de uso – descritos em linguagem controlada – e os executa, via integração com o Selenium. Porém, a especificação dos casos de uso é feita via editor de texto, o que possibilita que casos de uso existentes sejam aproveitados em parte – trazidos ao editor e adaptados ao formato proposto. Outros pontos divergentes são: (i) a abordagem de Pessoa (2011) não trata o sequenciamento de casos de uso, o que gera a necessidade de se cadastrar cenários completos (desde a autenticação), resultando em casos de uso muito extensos e de difícil leitura. Já na abordagem aqui apresentada, casos de uso podem ser sequenciados por meio da invocação de um por outro; (ii) quanto à identificação dos elementos da interface, na abordagem aqui apresentada, esses são especificados em separado, seguindo o padrão *Page Object*[3], favorecendo assim a organização e a manutenção; e (iii) outra diferença encontra-se na forma de declaração dos dados de teste: Em Pessoa (2011), são indicados valores fixos, enquanto na abordagem proposta podem ser indicadas regras que determinam os valores permitidos para cada dado de teste, permitindo sua geração de forma automática.

Por fim, em Pinto e Staa (2013), é apresentada FunTester, uma ferramenta para a geração, execução e análise de testes funcionais, de forma automática, a partir de casos de uso textuais e

[2] ANTLRWorks: <http://www.antlr3.org/works/>

[3] PageObject: Padrão de projeto para organização de testes funcionais, em que cria-se um objeto para cada página *web*, encapsulando campos e ações de cada pagina. <http://martinfowler.com/bliki/PageObject.html>

detalhamento de suas regras de negócio. O processo é realizado por meio das seguintes etapas: (i) geração dos cenários para cada caso de uso, por meio da combinação dos caminhos percorridos em sua execução; (ii) combinação dos cenários entre casos de uso, por meio da aplicação de ordenação topológica em um grafo acíclico dirigido, representando as dependências existentes entre os casos de uso (indicadas nas pré-condições ou nos fluxos); (iii) geração de casos de teste semânticos valorados e com oráculos, utilizando a informação dos cenários e do detalhamento das regras de negócio; (iv) transformação dos casos de teste semânticos em código-fonte para a linguagem de programação e *frameworks* de testes escolhidos (atualmente, são suportados os arcabouços TestNG[4] e FEST[5], para testes de aplicações com interface gráfica Swing); (v) execução do código-fonte, gerando um arquivo de log de execução no formato particular do *framework* de testes escolhido; (vi) coleta, pré-análise e transformação dos resultados da execução, para que seja lido pela ferramenta; (vii) coleta, análise e apresentação dos resultados.

A indicação dos valores para os dados de teste pode ser feita utilizando valores fixos ou regras que definam o tipo de dado e os valores admitidos pelo elemento, além de, opcionalmente, mensagens esperadas do SST caso alguma definição seja violada. A ferramenta gera automaticamente casos de teste para essas situações de violação, o que permite reduzir o número de fluxos alternativos necessários para descrever erros de uso.

Assim como em Pinto e Staa (2013), a abordagem aqui proposta gera *scripts* de teste automatizados a partir de casos de uso textuais, porém utiliza-se um editor de texto próprio para tal, com validações em tempo de edição. Também assim como em Pinto e Staa (2013), gera-se dados de teste de forma automática, a partir de regras que definem os valores permitidos para cada elemento. Porém, permite-se o uso de expressões, com a possibilidade de utilização de métodos próprios em JavaScript, o que amplia as possibilidades de especificação. Além disso, em Pinto e Staa (2013), as informações de domínio são misturadas às informações da IU, não sendo permitida sua reutilização, além de não serem tratadas classes de equivalência, extensões de telas ou formas alternativas para identificação de elementos da IU.

Já outro ponto em comum entre as abordagens é a geração de sequenciamento dos casos de uso, por meio de suas relações de dependência, gerando cenários completos de teste. Entretanto, um recurso diferencial no trabalho aqui apresentado é a possibilidade de invocação de um caso de uso indicando-se que esse seja executado por um cenário específico.

Além disso, assim como em Pinto e Staa (2013), gera-se automaticamente, a partir das descrições dos elementos, casos de teste que possibilitam verificar o comportamento do SST diante de valores limítrofes, valores aleatórios válidos e inválidos, assim como diante da ausência de valores obrigatórios, reduzindo a necessidade de descrição de fluxos alternativos para esse fim.

[4] TestNG: <http://testng.org>
[5] FEST: <http://code.google.com/p/fest/>

As avaliações das abordagens seguem a mesma proposta, de se verificar a efetividade dos casos de teste gerados, utilizando análise de mutantes. Entretanto, diferentemente desta dissertação, a análise em Pinto e Staa (2013) não se aprofunda na verificação de quais mutantes especificamente foram mortos em cada execução, sendo a eficácia dos testes gerados pela ferramenta, intuída pelos resultados do experimento (número de falhas geradas nos testes após a inserção de cada mutação).

Por fim, Pinto e Staa (2013) apresenta como diferencial o uso de extensões, que realizam a conversão de testes semânticos para a linguagem e *framework* de testes alvo. A abordagem permite definir um vocabulário composto pelos termos esperados por sua extensão – para a transformação dos testes úteis em código-fonte – e pelos termos correspondentes – usados na descrição textual dos casos de uso. Assim, permite-se variar os termos utilizados na especificação, assim como adaptá-los para diferentes arcabouços de teste.

Conforme relatado, várias alternativas são utilizadas para a identificação dos elementos contidos em casos de uso textuais. Alguns trabalhos optam pelo uso de PNL para análise de sentenças gramaticais, como Jiang e Ding (2011), Pedemonte, Mahmud e Lau (2012) e Thummalapenta et al. (2012); outros, pela utilização de linguagem controlada, como esta dissertação e Matos e Sousa (2010), além de Pessoa (2011) e Pinto e Staa (2013), que o fazem por meio de formulários com opções pré-definidas. Há ainda trabalhos, como Huang e Chen (2006), que optaram em não tratar especificações textuais e sim modelos UML, devido à facilidade na identificação de seus elementos, por meio de sua exportação para XMI.

Entretanto, há pesquisas que buscam identificar elementos presentes em especificações textuais legadas, como a apresentada em Rauf, Antkiewicz e Czarnecki (2011). Nesse trabalho, são lidos documentos rich-text, e, a partir de um template, extraem-se partes (estruturas lógicas) dos documentos, gerando–as em XML. Tal recurso mostra-se interessante pois, uma vez identificados os componentes da especificação, esses podem ser utilizados em transformações e processamentos posteriores diversos que visem, inclusive, a geração de testes.

6 CONCLUSÕES E TRABALHOS FUTUROS

Este capítulo apresenta as considerações finais da dissertação, ressaltando os pontos principais da abordagem proposta, suas contribuições e linhas de trabalhos futuros. O capítulo está organizado como descrito a seguir. A Seção 5.1 descreve os pontos mais significativos da solução proposta, a Seção 5.2, suas principais contribuições, enquanto a Seção 5.3, as limitações encontradas e possíveis trabalhos futuros.

6.1 Visão Geral da Solução Proposta

O processo que busca garantir a qualidade de um software passa por verificações de consistência entre a implementação realizada e sua especificação. Em uma dessas verificações – na fase de testes de sistema – simula-se a operação normal do sistema, testando suas funções de forma mais próxima possível do que irá ocorrer no ambiente do usuário. Testes realizados por meio da interface do sistema e com foco nos requisitos funcionais do software (testes funcionais / caixa-preta) são comumente executados nessa fase.

Sendo um sistema especificado por meio de casos de uso, uma vez que esses descrevem uma expectativa de interação com a interface, é possível automatizar o processo de simulação, transformando tais expectativas em verificações.

Assim, este trabalho buscou desenvolver uma solução que permitisse a realização desse processo, derivando testes funcionais automatizados a partir de casos de uso. A solução envolveu o desenvolvimento de uma DSL para a formalização da descrição dos casos de uso, dicionário de dados e interfaces do sistema, e uma ferramenta com dois módulos, sendo um para a transformação dos casos de uso em testes e outro, para a execução dos testes, análise e armazenamento do seu resultado.

A utilização do ambiente proposto mostrou-se viável para sistemas reais, de acordo com estudo de caso realizado. Os experimentos demonstraram a eficácia e corretude dos testes gerados pela abordagem proposta no apontamento de mutações inseridas no SST (que simularam defeitos reais do software). Da mesma forma, ao se introduzir modificações na especificação e regerar os testes, os pontos divergentes entre o SST e a especificação modificada foram apontados corretamente pelo novo conjunto de testes.

Anteriormente aos experimentos com mutações, quando realizou-se a especificação no ambiente EnvUCT tendo como base a documentação do sistema, foram apontados pelos testes, vários pontos de divergência entre a documentação e o SST. Entre eles, *labels* ou limites

divergentes, campos existentes no SST que não existiam na documentação e vice-versa. Em todos os casos, aparentemente, o SST passou por mudanças que não foram ajustadas na documentação. Além desses, encontrou-se um *bug* relacionado ao preenchimento de um dos campos, ao utilizar um valor com o tamanho máximo permitido para o mesmo.

O fato de terem sido encontrados pontos não atualizados na documentação do sistema utilizado nos experimentos reforça um dos objetivos deste trabalho que é o de incentivar a manutenção da documentação, uma vez que ela passaria a ser a base para os testes funcionais automatizados.

6.2 Contribuições

As principais contribuições desta dissertação são:

- Definição de uma DSL (LUCT) para a descrição de casos de uso, dicionário de dados e interfaces de usuário, possibilitando sua formalização e, por conseguinte, a identificação de seus elementos para derivações posteriores;
- Disponibilização de um ambiente (EnvUCT) com validações em tempo de edição que auxiliam o usuário durante a especificação, favorecendo o aumento da produtividade e da qualidade das especificações de caso de uso;
- Disponibilização de módulo na LUCTool (UC2Test) que gera, de forma automática, cenários de teste a partir de casos de uso, e, desses cenários, casos de teste que são convertidos em testes funcionais automatizados. A geração de cenários combina cenários de casos de uso por meio de suas relações de dependência e a geração de casos de teste envolve estratégias que possibilitam explorar várias situações, como o uso de valores limítrofes, valores aleatórios válidos, inválidos e a ausência de valores obrigatórios, cobrindo um grande número de casos práticos, que seriam trabalhosos se feitos manualmente;
- Disponibilização de módulo na LUCTool (UC2Test-Exec) que executa, analisa e armazena os resultados dos testes, com evidências que auxiliam na descoberta de motivos de falhas na execução, que podem indicar possíveis defeitos no software;
- Incentivar a documentação de software, uma vez que se propõe a derivação de testes automatizados a partir da especificação atualizada;
- Auxiliar na redução da complexidade e custos na geração e manutenção de testes funcionais automatizados;
- Estudo de caso demonstrando a viabilidade técnica do uso da abordagem proposta para dois sistemas de software reais; e

- Publicação de artigo: “An environment for automatic tests generation from use case specifications.” *Proceedings of the International Conference on Software Engineering Research and Practice (SERP)*. The Steering Committee of The World Congress in Computer Science, Computer Engineering and Applied Computing (WorldComp), 2014. Cunha, C. D., & Song, M. A. J.

6.3 Limitações e Trabalhos Futuros

Esta seção apresenta limitações encontradas na solução proposta neste trabalho de pesquisa e aponta possíveis trabalhos futuros.

6.3.1 Ambiente EnvUCT

- Criar validação semântica para verificar a corretude na indicação de parâmetros na invocação de casos de uso;

- Facilitar a indicação do item de dicionário relacionado a um campo da interface. A especificação das interfaces do sistema em separado do dicionário de dados mostrou-se interessante por favorecer a organização e permitir reutilização. Nos experimentos, pôde-se referenciar itens do dicionário em mais de uma interface de usuário, além de utilizar a indicação de `ClasseBase` no dicionário, mantendo elementos em comum em uma mesma classe. Porém, a especificação mostrou-se trabalhosa na necessidade de apontar o item de dicionário em cada campo da interface, principalmente porque, na maioria dos casos, indicava-se um elemento do dicionário com o mesmo nome do campo. Assim, tornar essa indicação automática (ou sugerida), tornaria a especificação mais produtiva;

- Lidar com modelagem para as telas, como no Webspec (citado na Seção 5), uma vez que as tecnologias dos dois trabalhos são relacionadas (XText com EMF e GMF);

- Gerar facilidades para importação de especificações cadastradas em documentos *rich-text*. Utilizar trabalhos de pesquisa nessa linha, como o de Rauf, Antkiewicz e Czarnecki (2011), em que indica-se o padrão com que o documento foi escrito para extrair dele as informações por áreas de conteúdo.

6.3.2 Estrutura do SST

- Permitir informar mais de um valor em campo múltiplo;

- Permitir especificar variação do fluxo em mais um nível, ou seja, poder indicar um fluxo alternativo dentro de outro;

- Tratar de forma diferenciada sistemas que exibem todas as mensagens de validação em conjunto, dos que exibem uma mensagem por vez;
- Disponibilizar possibilidades de interação com a IU além das exploradas, como downloads, uploads de arquivos, ou manipulação de elementos por meio de mecanismo *drag and drop*;
- Possibilitar verificações ainda não tratadas, como a de características de elementos da interface ou o envio de e-mails pelo SST;
- Realizar novos experimentos com sistemas de software reais, identificando novos pontos a acrescentar na linguagem LUCT e na ferramenta LUCTool, para atender situações ainda não exploradas.

6.3.3 Execução de testes

- Tratar fluxos de espera não ocupada;
- Disponibilizar opção de execução de testes em que se selecione somente os casos de uso/cenários que tenham tido pontos alterados, entre duas versões. A identificação de pontos com alteração é possível por meio de integração com um gerenciador de versões. Essa opção seria útil quando a alteração na especificação estiver refletindo uma alteração no SST. Apesar das opções atuais (descritas na Subseção B.3) trazerem bastante flexibilidade na indicação de casos de teste a serem executados, essa opção traria maior facilidade nesses casos.

6.3.4 Linguagem LUCT

- Disponibilizar um tipo de Passo Informar que preencha todos os campos de uma tela, para tornar mais prática a especificação de fluxos simples. Apesar de, com essa mudança, os nomes dos campos não ficarem visíveis no caso de uso, os mesmos poderiam ser visualizados na descrição da interface (para a qual há um *link*, no início do caso de uso, onde indica-se a interface acessada);
- Permitir renomear cenários ou gerar um novo nome (ou descrição) dos cenários com base nas extensões pela quais ele passa. Para a segunda opção, utilizaria-se abreviações cadastradas nas extensões;
- Incorporar à DSL expressões matemáticas com os operadores básicos, permitindo, como operandos, os tipos básicos tratados na especificação – numérico, texto e data;
- Possibilitar tratar expressões JavaScript como parte integrante da DSL – por um mecanismo de extensão;

- Tratar pré-condições específicas de um fluxo de extensão.

6.3.5 Ferramenta LUCTool

- Permitir apontar variações de cenários com pouca influência na execução de casos de uso que o invoquem. A indicação poderia ser feita em cada ponto de invocação. Havendo essa indicação, no momento de montar o sequenciamento, a ferramenta escolheria um cenário aleatoriamente, resultando em um único sequenciamento dos cenários, reduzindo o número total de casos de teste. Quando escolhe-se a execução por nível 3, também é escolhido um cenário aleatoriamente, porém, é executado um único caso de teste por sequenciamento e não há forma de parametrizar quais invocações variar. A sugestão colocada aqui permitiria tal parametrização, uma vez que podem haver cenários com relacionamentos de diferentes graus de influência;

- Derivar automaticamente testes para exercitar a variação de classes de equivalência, quando ela não trouxer alteração substancial no fluxo a ser percorrido. Atualmente cadastra-se extensões para exercitar as classes de equivalência, mesmo quando não há mudança real no fluxo. Um exemplo foi visto nos experimentos. Além da criação das extensões ser trabalhosa, ela leva à duplicação de cenários;

- Disponibilizar mecanismo para parametrizar casos de uso similares, que tenham o mesmo padrão de navegação. Em Cockburn (2005), é descrita uma parametrização em que o comportamento do caso de uso é descrito uma única vez e os detalhes de cada uso, como "qualidades pesquisáveis" e "valores de exibição", são cadastrados à parte. Como nessa abordagem já tem-se o cadastro dos elementos da IU, a parametrização mostra-se uma solução interessante, principalmente para casos de uso CRUD simples, se esses forem numerosos;

- Permitir indicar um tipo de teste específico a ser executado em um ponto de invocação. Atualmente, é possível indicar somente uma extensão. No decorrer dos experimentos realizados, optou-se por executar um caso de uso invocado com o menor cenário possível. Foi então criada uma extensão para esse fim, que foi indicada na invocação. Se houvesse a possibilidade de indicação de um tipo de teste, isso não seria necessário, pois o tipo de teste `SO_OBRIGATORIOS` atenderia à necessidade. Para testes mais elaborados que simulem situações específicas, essa possibilidade pode se mostrar útil;

- Permitir a interação com conteúdos exibidos em outros meios além do HTML, como por exemplo, arquivos PDF;

- Criar mecanismo para capturar a estrutura e elementos das interfaces do sistema de forma automática. Tal mecanismo seria útil para facilitar a especificação, pois mesmo em casos em que há uma especificação das telas, ela pode estar desatualizada ou incompleta. O

usuário navegaria pelas telas das quais gostaria de capturar os elementos, e esses seriam mostrados em uma lista, para uso na especificação. Além de facilitar a especificação, tal mecanismo resolveria a questão de alguns campos que os testes não conseguem apontar sua falta, ao deixarem de ser especificados.

REFERÊNCIAS

ATI. *Automated Testing Institute*. 2014. Disponível em: <http://www.automatedtestinginstitute.com/>. Acesso em: 10 jan. 2014.

BINDER, R. V. *Testing object-oriented systems: models, patterns, and tools*. Boston: Addison-Wesley Professional, 2000.

BOOCH, G.; RUMBAUGH, J.; JACOBSON, I. *UML-Guia Do Usuário*. 2nd. ed. Rio de Janeiro: Elsevier, 2005.

BRAY, T. et al. *Extensible Markup Language (XML) 1.0 (Fifth Edition)*. 2014. Disponível em: <http://www.w3.org/TR/xml/>. Acesso em: 02 mar. 2014.

COCKBURN, A. *Escrevendo casos de usos eficazes: Um guia prático para desenvolvedores de software*. Porto Alegre: Bookman, 2005.

DELAMARO, M. E.; MALDONADO, J. C.; JINO, M. *Introdução ao teste de software*. Rio de Janeiro: Campus, 2007.

FEWSTER, M.; GRAHAM, D. *Software test automation*. New York: Addison-Wesley Professional, 1999.

FORTUNA, M. H.; BORGES, M. R. et al. Info cases: integrating use cases and domain models. In: IEEE INTERNATIONAL REQUIREMENTS ENGINEERING CONFERENCE, 16., 2008, Barcelona, Catalunya. ***Proceedings...*** Los Amitos, CA: IEEE, 2008. p. 81–84.

GARCíA, B.; DUEÑAS, J. C. Automated functional testing based on the navigation of web applications. In: INTERNATIONAL WORKSHOP ON AUTOMATED SPECIFICATION AND VERIFICATION OF WEB SYSTEMS - WWV, 7., 2011, Reykjavik. ***Proceedings...*** Iceland, 2011. p. 49–65.

GOIS, F. N. B.; FARIAS, P.; OLIVEIRA, R. Test script diagram–um modelo para geração de scripts de testes. In: SIMPóSIO BRASILEIRO DE QUALIDADE DE SOFTWARE, 9., 2010, Belém. ***Anais...*** Porto Alegre: SBC, 2010. p. 73–87.

GRADY, R. B. *Practical software metrics for project management and process improvement*. New Jersey: Prentice-Hall, Inc., 1992.

GUTIÉRREZ, J. et al. Derivation of test objectives automatically. In: WOJTKOWSKI, W. et al. (Ed.). *Advances in Information Systems Development*. US: Springer, 2007. v. 2, p. 435–446.

GUTIÉRREZ, J. J. et al. A case study for generating test cases from use cases. In: INTERNATIONAL CONFERENCE ON RESEARCH CHALLENGES IN INFORMATION SCIENCE (RCIS), 2., 2008, Marrakech. ***Proceedings...*** Morocco: IEEE, 2008. p. 209–214.

HARROLD, M. J. Reduce, reuse, recycle, recover: Techniques for improved regression testing. In: IEEE INTERNATIONAL CONFERENCE ON SOFTWARE MAINTENANCE, 2009,Edmonton, Alberta. ***Proceedings...*** USA: IEEE, 2009. p. 5.

HEUMANN, J. Generating test cases from use cases. *The rational edge*, v. 6, n. 1, 2001.

HUANG, C.-h.; CHEN, H. Y. A tool to support automated testing for web application scenario. In: INTERNATIONAL CONFERENCE ON SYSTEMS, MAN AND CYBERNETICS (SMC), 2006,Taipei. ***Proceedings...*** US, CA: IEEE, 2006. v. 3, p. 2179–2184.

IEEE. *IEEE Standard for Software and System Test Documentation - IEEE Std 829-2008*. IEEE Computer Society, 2008.

IEEE. *IEEE Standard for Systems and software engineering - Vocabulary - IEEE Std 24765-2010*. IEEE Computer Society, 2010.

JIANG, M.; DING, Z. Automation of test case generation from textual use cases. In: INTERNATIONAL CONFERENCE ON INTERACTION SCIENCES (ICIS),4., 2011,Busan. ***Proceedings...*** Korea: IEEE, 2011. p. 152–157.

LARMAN, C. *Utilizando UML e padrões: Uma introdução à análise e ao projeto orientados a objetos e ao desenvolvimento interativo*. 3rd. ed. Porto Alegre: Bookman, 2007.

LUNA, E. R.; ROSSI, G.; GARRIGÓS, I. Webspec: a visual language for specifying interaction and navigation requirements in web applications. *Requirements Engineering*, Springer, v. 16, n. 4, p. 297–321, 2011.

MARIANI, L. et al. Autoblacktest: a tool for automatic black-box testing. In: INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING - ICSE, 33., 2011, Waikiki. ***Proceedings...*** New York, USA: ACM, 2011. p. 1013–1015.

MATOS, E. C. B. de; SOUSA, T. C. From formal requirements to automated web testing and prototyping. *Innovations in Systems and Software Engineering*, Springer, v. 6, n. 1-2, p. 163–169, 2010.

MYERS, G. J.; SANDLER, C.; BADGETT, T. *The art of software testing*. 3rd. ed. New Jersey: John Wiley & Sons, 2011.

NEBUT, C. et al. Automatic test generation: A use case driven approach. *IEEE Transactions on Software Engineering*, IEEE, v. 32, n. 3, p. 140–155, 2006.

NETO, P. de Alcântara dos S. *MODEST: Um Método de Teste Baseado em Modelos*. 138 f. Tese (Doutorado) — Universidade Federal de Minas Gerais, Programa de Pós-Graduação em Ciência da Computação, Belo Horizonte, 2006.

PAGEOBJECT. *Page Object*. 2014. Disponível em: <http://martinfowler.com/bliki/PageObject.html/>. Acesso em: 12 jan. 2014.

PEDEMONTE, P.; MAHMUD, J.; LAU, T. Towards automatic functional test execution. In: ACM INTERNATIONAL CONFERENCE ON INTELLIGENT USER INTERFACES, 2012, Lisbon. ***Proceedings...*** New York, USA: ACM, 2012. p. 227–236.

PESSOA, M. B. *Geração e execução automática de scripts de teste para aplicações web a partir de casos de uso direcionados por comportamento*. 99 f. Tese (Dissertação de mestrado) — Pontifícia Universidade Católica do Rio de Janeiro, Programa de pós-graduação em Informática, Rio de Janeiro, 2011.

PEZZÈ, M.; YOUNG, M. *Teste e análise de software: processos, princípios e técnicas*. Porto Alegre: Bookman, 2008.

PINTO, T. D.; STAA, A. V. Functional validation driven by automated tests. In: BRAZILIAN SYMPOSIUM ON SOFTWARE ENGINEERING (SBES), 27., 2013, Brasília. ***Proceedings...*** Porto Alegre: SBC, 2013. p. 41–48.

PRESSMAN, R. S. *Engenharia de software*. 7th. ed. Porto Alegre: McGraw Hill Brasil, 2011.

RAUF, R.; ANTKIEWICZ, M.; CZARNECKI, K. Logical structure extraction from software requirements documents. In: INTERNATION CONFERENCE ON REQUIREMENTS ENGINEERING (RE), 19., 2011, Trento. ***Proceedings...*** US: IEEE, 2011. p. 101–110.

RIOS, E.; FILHO, T. M. *Teste de software*. 2nd. ed. Rio de Janeiro: Alta Books, 2006.

RUP. *Rational Unified Process*. 2014. Disponível em: <http://www.wthreex.com/rup/>. Acesso em: 20 jan. 2014.

SAVIC, D. et al. Language for use case specification. In: SOFTWARE ENGINEERING WORKSHOP (SEW), 34., 2011, Limelick. ***Proceedings...*** US: IEEE, 2011. p. 19–26.

SAVIC, D. et al. Use case specification at different levels of abstraction. In: INTERNATIONAL CONFERECEN ON THE QUALITY OF INFORMATION AND COMMUNICATIONS TECHNOLOGY (QUATIC), 8., 2012, Lisbon. ***Proceedings...*** Milwaukee, WI, USA: American Society for Quality - ASQ, 2012. p. 187–192.

SINHA, A.; SUTTON, S.; PARADKAR, A. Text2test: Automated inspection of natural language use cases. In: INTERNATIONAL CONFERENCE ON SOFTWARE TESTING, VERIFICATION AND VALIDATION (ICST), 3., 2010, Paris. US, CA: IEEE, 2010. p. 155–164.

SMIALEK, M. Accommodating informality with necessary precision in use case scenarios. *Journal of Object Technology*, v. 4, n. 6, p. 59–67, 2005.

SOMÉ, S. S. Supporting use case based requirements engineering. *Information and Software Technology*, Elsevier, v. 48, n. 1, p. 43–58, 2006.

SOME, S. S.; CHENG, X. An approach for supporting system-level test scenarios generation from textual use cases. In: ACM SYMPOSIUM ON APPLIED COMPUTING, 2008, Fortaleza. ***Proceedings...*** New York: ACM, 2008. p. 724–729.

THUMMALAPENTA, S. et al. Automating test automation. In: INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING - ICSE, 34., 2012, Zurich. ***Proceedings...*** US, CA: IEEE, 2012. p. 881–891.

WANG, X.; XU, P. Build an auto testing framework based on selenium and fitnesse. In: INTERNATIONAL CONFERENCE ON INFORMATION TECHNOLOGY AND COMPUTER SCIENCE (ITCS), 2009, Kiev, Ukraine. ***Proceedings...*** Los Alamitos, CA: IEEE, 2009. p. 436–439.

XTEXT. *Xtext*. 2014. Disponível em: <http://www.eclipse.org/Xtext>. Acesso em: 10 mar. 2014.

APÊNDICE A – DSL LUCT

Este apêndice apresenta as regras definidas na DSL proposta (LUCT). São exibidos trechos completos da DSL, além de informações complementares à Seção 3.2, relacionadas à especificação dos casos de uso, interfaces de usuário e dicionário de dados, tratadas respectivamente pelas Seções A.1, A.2 e A.3.

A.1 Descrição dos Casos de Uso

A Listagem 56 exibe o trecho da DSL LUCT que trata da estrutura principal da definição de casos de uso.

Listagem 56 – Trecho da DSL LUCT para definição de casos de uso

```
1  CasoDeUso =
2    "CasoDeUso"  ID  [NivelObjetivo]  "{"
3    ["contexto"  STRING ]
4    ["palavraschave"  STRING ]
5    ["prioridade"  INT ]
6    ["execucaopropria"  ("S"|"N") ]
7    ["tipo"  TipoCasoUsoLogin ]
8    ["ator"  ID ]
9    ["escopo"  STRING ]
10   ["stakeholders"  STRING ]
11   ["acionador"  Acionador ]
12   ["precondicao"  PreCondicao ]
13   (("cenarioprincipal"  CenarioPrincipal) | (PENDENTE  STRING))
14   ["extensoes"  Extensoes ]
15   ["garantiasminimas"  PassoGarantias ]
16   ["garantiasdesucesso"  PassoGarantias ]
17   ["listadevariacoes"  STRING ]
18   ["informacaorelacionada"  STRING ]
19   "}"
20 ;
21 NivelObjetivo = ( "(+)" | "(!)" | "(-)" ) ;
22 PENDENTE = "PENDENTE" ;
```

A Listagem 57 exibe trechos da DSL LUCT que tratam da definição dos fluxos do cenário principal e extensões, que são compostos por passos de caso de uso (`PassoUC`) e esses, por passos de detalhamento (`Passo`). Os passos de detalhamento são de um dos super-tipos: `Ação`, `Verificação` ou `Controle`.

Listagem 57 – Trecho da DSL LUCT para definição do cenário principal e extensões

```
1  CenarioPrincipal = PassoUC  { ","  PassoUC }* ;
2  PassoUC = INT  STRING  "{"  Passo  { ","  Passo }*  "}" ;
3
4  Extensoes = CondicaoEPassosFluxoAlt  { ","  CondicaoEPassosFluxoAlt }* ;
```

```
CondicaoEPassosFluxoAlt = FluxoAlternativoCondicao
        { PassoUCAlt }+  [ PassoUCAltFinal ] ;
FluxoAlternativoCondicao = INT  LETRA
        [ ("erro" | "erro-teste-padrao") ]  STRING
        [ "extensão de tela"  ID ] ;
PassoUCAlt = INT  LETRA   INT  STRING
        [ "{"  Passo  { ,  Passo }*   "}"       ] ;
PassoUCAltFinal = INT  LETRA  INT  PassoUCAltExecutaFinal ;
PassoUCAltExecutaFinal = Executa passo  INT ;


Passo = Acao | Verificacao | Controle | PENDENTE ;
```

Os tipos de passo disponíveis por cada super-tipo são descritos no Quadro 14. A Listagem 58 exibe o trecho da DSL LUCT que trata dessa definição.

Listagem 58 – Trecho da DSL LUCT para definição de passos

```
Acao = ( Acessar | Informar | Clicar | Aguardar | Executar | ResponderAlerta ) ;

Informar = ( InformarCampo | InformarCampos |
        InformarIncorretamenteCampo | InformarIncorretamenteCampos ) ;
InformarCampo = "Informo"  ID  [ "com"  Valor ]  ;
InformarCampos = "Informo["  ID  { ","  ID }*  "]" ;
InformarIncorretamenteCampo = "InformoIncorretamente"  ID  [ "com"  Valor ];
InformarIncorretamenteCampos = "InformoIncorretamente["  ID  { ","  ID }*  "]" ;
Valor = ( ValorFixo | ValorDinamico | "valor armazenado" ID ) ;
ValorFixo = STRING ;
ValorDinamico = ID "." ID [ "." ID ] ;


Acessar = "Acesso"  ("menu"  ID | "link"  ID) ;
Clicar = "Clico"  ID ;
Aguardar = ( AguardarTempo | AguardarCampo ) ;
AguardarTempo = "Aguardo"  "tempo="  INT ;
AguardarCampo = "Aguardo"  ( "campo="  ID | "texto="  STRING) ;
Executar = "Executo"  ID  [ PreExecucaoParms ]
        [ "com"  "extensao"  "="  STRING ] ;
PreExecucaoParms = "com"  { ID  "="  STRING  "," }*
        ID  "="  STRING ;
ResponderAlerta = "Respondo alerta com "  ( "ok" | "cancel" ) ;
AcesseiTela = "AcesseiTela" ID;

Controle = ArmazenarValor | AcesseiTela;
ArmazenarValor = "Armazeno"  ( Valor | ValorTela | "campo" ID )  "como"  ID ;

Verificacao = ( VerificacaoNaTela | VerificacaoDeMsgErroTestesPadrao ) ;
VerificacaoNaTela = (VerificacaoNaTelaPrincipal | VerificacaoNaTelaAlerta);
VerificacaoNaTelaPrincipal = "Visualizo "  ( "mensagem"  ID | "texto = "  STRING | "
    valor armazenado" ID )  [ ValorTela ] ;
VerificacaoNaTelaAlerta = "Visualizo alerta com "  ( "mensagem"  ID |   "texto = "
    STRING | "valor armazenado" ID ) ;

VerificacaoDeMsgErroTestesPadrao =
        ( VerificacaoDeMsgErroTestesPadraoNaTelaPrincipal |
        VerificacaoDeMsgErroTestesPadraoNaTelaAlert ) ;
```

```
37 VerificacaoDeMsgErroTestesPadraoNaTelaPrincipal =
38         "Visualizo uma mensagem de erro de testes padrão"  [ ValorTela ] ;
39 VerificacaoDeMsgErroTestesPadraoNaTelaAlert =
40         "Visualizo alerta com uma mensagem de erro de testes padrão" ;
41
42 ValorTela = ( "no campo"  ID | "na tabela"  TabelaLinhaColuna ) ;
43 TabelaLinhaColuna = [ ("num"  INT | "id"  ID) ]
44         [ "Linha"  ("num"  INT | "id"  ID) ]
45         [ "Coluna"  ("num"  INT | "id"  ID) ] ;
```

Quadro 14 – Tipos de Passos

Passo (Ação)	**Descrição**
Acessar	Clica em um *link* ou opção de menu.
Informar	
Informar Campo	Preenche um campo da IU.
Informar Campos	Preenche vários campos da IU.
Informar Campo Incorretamente	Preenche incorretamente um campo da IU.
Informar Campos Incorretamente	Preenche incorretamente vários campos da IU.
Clicar	Clica em um elemento da IU.
Aguardar	
Aguardar Tempo	Aguarda por N segundos.
Aguardar Campo	Aguarda que um determinado campo esteja disponível ou que um determinado texto seja exibido na IU.
Executar	Invoca um caso de uso.
Responder Alerta	Responde a um pedido de confirmação (Clica em *ok* ou *cancel* em uma tela de confirmação do navegador).
Passo (Verificação)	**Descrição**
Visualização na Tela Principal	Verifica se um texto (comum ou de mensagem pré cadastrada) é exibido na tela ou em um determinado elemento da IU.
Visualização em Tela de Alerta	Verifica se um texto (comum ou de mensagem pré cadastrada) é exibido em uma tela de confirmação do navegador.
Visualização Msg Erro de Teste Padrão na Tela Principal	Verifica se a mensagem de erro relacionada a um teste padrão é exibida na tela ou em um determinado elemento da IU. (ver Subseção 3.3.2.2)
Visualização Msg Erro de Teste Padrão em Tela de Alerta	Verifica se a mensagem de erro relacionada a um teste padrão é exibida em uma tela de confirmação do navegador. (ver Subseção 3.3.2.2)
Passo (Controle)	**Descrição**
Armazenar Valor	Armazena valores para uso posterior, podendo ser um valor gerado dinamicamente ou buscado do texto de um elemento da IU.
Acessei Tela	Utilizado em pré-condições, indica a obrigatoriedade de que determinada tela tenha sido acessada, antes da invocação desse caso de uso por outros.

Por fim, a Listagem 59 exibe trechos da DSL LUCT que tratam da definição dos elementos `PréCondicao`, `Acionador`, e pós-condições (`GarantiasMínimas` e `GarantiasDeSucesso`).

Listagem 59 – Trecho da DSL LUCT para definição do acionador do caso de uso, pré e pós-condições

```
1  Acionador = "{"  PassoAcionador  { ","  PassoAcionador }*  "}" ;
2  PassoAcionador = Acessar | Clicar | PENDENTE ;
3
4  PreCondicao = PassosPreCondicao  { ","  PassosPreCondicao }* ;
5  PassosPreCondicao = STRING [ "{" PassoPreCondicao  { ","  PassoPreCondicao}* "}" ];
6  PassoPreCondicao = ( Executar | ArmazenarValor | Acessar | AcesseiTela | PENDENTE );
7
8  PassoGarantias = STRING  "{"  PassoPos  { ","  PassoPos }*  "}" ;
9  PassoPos = ( ArmazenarValor | Verificacao| PENDENTE ) ;
```

A.2 Descrição das Interfaces de Usuário

A.2.1 Menus e Links

A especificação de menus ou *links* (que podem fazer o papel de menu) seguem as regras da DSL LUCT exibidas na Listagem 60. A descrição de um menu é composta por itens de menu, que contêm:

- Identificador interno, que será utilizado para referenciar esse elemento nos casos de uso;
- Identificação do elemento na tela, que pode ser dada de uma das formas:
  - Por sua propriedade *id* (Identifica tanto elementos com a propriedade HTML *id* quanto com a propriedade HTML *name*),
  - Pela expressão XPath (XML Path Language)[1] que leva a encontrar o elemento;
- Indicação da tela que é acessada ao clicar no item de menu.

Listagem 60 – Trecho da DSL LUCT para definição de links e menus

```
1  Menu = "Menu"  [ Menu1oNivel ]  "{"  ItemMenu  { ItemMenu }*  "}" ;
2  Menu1oNivel = ( "id1oNivel="  ID | "xpath1oNivel="  STRING ) ;
3  ItemMenu = ID  ("id="  ID | "xpath="  STRING )  "tela="  ID ;
4
5  Link = "Link"  ID  ("url="  STRING | "texto="  STRING) "tela="  ID  ;
```

Quando o menu é formado por dois níveis, deve ser informada também a identificação do 1^o nível ou nível superior. Identificar o elemento superior é necessário para que o mesmo possa receber o foco e então serem exibidas as opções do 2^o nível.

Para SSTs em que os elementos do menu não são identificáveis por *id*, esses devem ser identificados por meio de expressões XPath. Para obter a expressão XPath de um elemento

[1] XPath: Linguagem para identificar partes de um documento XML. Uma das formas de identificação de elementos para automação de testes de software em plataforma *web*. http://www.w3.org/TR/xpath/

sugere-se o uso de um complemento para navegador que realize a inspeção do código fonte HTML, como o Firebug[2]. Com esse recurso, expressões Xpath são facilmente obtidas, e mesmo a tarefa de encontrar o identificador (propriedade *id*) de um elemento torna-se mais rápida.

A decrição de um *link* deve conter:

- Identificador interno, que será utilizado para referenciar esse elemento nos casos de uso;
- Um *url* (que é acessado diretamente) ou o texto do *link* (para identificação do elemento na IU);
- Indicação da tela que é acessada ao clicar no *link*.

A.2.2 Áreas e Mensagens Globais

A DSL LUCT permite a descrição de elemento globais ao sistema, como mensagens e áreas da tela (Listagem 61).

Listagem 61 – Trecho da DSL LUCT para definição de áreas globais e mensagens padrões

```
1  AreasGlobais = "AreasGlobais"  "{"  Campo  { ","  Campo }*  "}" ;
2
3  MensagensGlobais = "MensagensGlobais"  "{"  Mensagem  { ","  Mensagem }*  "}" ;
4
5  MensagensErroTestesPadrao = "MensagensErroTestesPadrao"  "{"
6      MensagemErroTesteCamposObrigatorios  MensagemErroTesteCamposAbaixoLimite
7      MensagemErroTesteCamposAcimaLimite  MensagemErroTesteFormatoCampos  "}" ;
8
9  MensagemErroTesteCamposObrigatorios = "MSG_ERRO_CAMPO_OBRIGATORIO ="  STRING ;
10
11 MensagemErroTesteCamposAbaixoLimite = "MSG_ERRO_CAMPO_ABAIXO_LIMITE ="  STRING ;
12
13 MensagemErroTesteCamposAcimaLimite = "MSG_ERRO_CAMPO_ACIMA_LIMITE ="  STRING ;
14
15 MensagemErroTesteFormatoCampos = "MSG_ERRO_CAMPO_FORA_FORMATO ="  STRING ;
```

Áreas (ou campos) globais são compostas por uma lista de campos, normalmente do tipo HTML = CAMPO_AREA_MENSAGEM. São úteis quando o SST apresenta um padrão na definição de áreas da tela para determinados fins, como: para a exibição do usuário autenticado, para o título da IU atualmente em execução, ou para mensagens geradas pelo SST para confirmações, erros ou alertas. Detalhes sobre a forma como elementos do tipo Campo devem ser descritos podem ser obtidos na Subseção A.2.3.

Mensagens globais são compostas por um identificador interno (que será utilizado para referenciar essa mensagem, nos casos de uso) e por um texto que é a mensagem, propriamente.

[2] Firebug: Complemento para navegadores com recursos que permitem depurar e inspecionar o HTML da página exibida. http://getfirebug.com/

Já as mensagens de erro para testes padrão devem especificar mensagens de erro para situações em que os elementos da interface não sejam informados corretamente (quando estão fora de formato, fora do limite especificado, ou quando não são informados e são obrigatórios). Como são globais, nos textos das mensagens pode-se utilizar expressões para indicar pontos onde exibe-se o nome do campo que está sendo testado.

A.2.3 Interfaces de Usuário

As regras para a definição de uma IU são apresentadas na Listagem 62.

Listagem 62 – Trecho da DSL LUCT para definição de IUs

```
1  Tela = "Tela"  ID  "{"
2     Campo  { ","  Campo }*
3    {ExtensaoTela}*  [MensagensTela]  "}" ;
4
5  ExtensaoTela = "Extensao"  ID  "{"  Campo  { ","  Campo }*  "}" ;
6
7  Campo = ["*"]  ID  ":"  TipoElementoHtml  IdentificacaoNaTela
8   [ "descricao ="  STRING ]
9   [ "valores ="  ValorDinamico ]
10  [ "(carrega valores após"  ID  ")" ]
11  [ "aciona tela"  ID ]
12  [ "preenchimento forçado"]
13  { "["  MensagemErroCampo  "]" }* ;
14
15 IdentificacaoNaTela = ( "label ="  STRING | "id ="  STRING |
16   "xpath ="  STRING | "identificacao ="  PENDENTE ) ;
17
18 TipoElementoHtml = ("CAMPO_TEXTO" | "CAMPO_LISTA" | "CAMPO_RADIO" | "CAMPO_CHECKBOX"
       | "CAMPO_AREA_DE_TEXTO" |  "CAMPO_BOTAO" | "CAMPO_LINK" | "CAMPO_AREA_MENSAGEM"
       | PENDENTE  ) ;
19
20 ValorDinamico = ID "." ID [ "." ID ] ;
21
22 MensagensTela =  "Mensagens" "{"  Mensagem  { ", Mensagem }*  "}"  ;
23 Mensagem =  ID STRING  ;
24
25 MensagemErroCampo = MensagemErroCampoObrigatorio | MensagemErroCampoAbaixoLimite |
26         MensagemErroCampoAcimaLimite | MensagemErroFormatoCampo ;
27 MensagemErroCampoObrigatorio = "MSG_ERRO_CAMPO_OBRIGATORIO =" STRING ;
28 MensagemErroCampoAbaixoLimite = "MSG_ERRO_CAMPO_ABAIXO_LIMITE =" textoMsg=STRING ;
29 MensagemErroCampoAcimaLimite = "MSG_ERRO_CAMPO_ACIMA_LIMITE =" textoMsg=STRING ;
30 MensagemErroFormatoCampo = "MSG_ERRO_CAMPO_FORA_FORMATO ="  textoMsg=STRING ;
```

Uma IU é composta por campos, mensagens e extensões. Para cada campo devem ser descritos:

- Se é obrigatório (indicado por '*', antes da identificação do campo);

- Seu identificador interno, que será utilizado para referenciar esse elemento nos casos de uso;

- Seu tipo HTML. Se é um campo texto (*input*), lista de seleção (*select*), caixa de seleção (*checkbox*), botão de rádio (*radio*), botão (*button*), *link* (*a - anchor*), área de texto (*textarea*) ou área de mensagem (normalmente um *div* ou *td*);

- Sua identificação na tela, que pode ser dada de uma das formas:
  - O texto do rótulo (*label*) relacionado ao campo (Pode ser utilizada essa forma de identificação quando os HTMLs do SST utilizam elementos *label*[3]). Essa opção deve ser preferencialmente utilizada, para evitar inserir detalhes de implementação na especificação;
  - Sua propriedade *id* (Identifica tanto campos com a propriedade *id* quanto com a propriedade *name*); ou
  - A expressão XPath que leva a encontrar o campo, quando necessário (Para situações em que o campo ou área da tela não tem identificação previamente conhecida, como quando é necessário escolher um elemento por posição relativa a outro);

- Uma descrição (opcional);

- A indicação da regra do dicionário de dados que gera os valores permitidos para o campo;

- A indicação de que esse campo depende ou não de outro, para carregar seus valores válidos (situação comum em listas de seleção). Nesse caso, aguarda-se que as opções da lista de seleção dependente seja carregada, após o preenchimento do campo principal.

- A indicação de que esse campo funciona como um acionador de uma nova tela;

- A indicação de que esse campo tem sempre um valor preenchido, seja internamente pelo SST ou devido à sua apresentação na IU – como um elemento inicializado que não pode ter seu valor retirado, como por exemplo um elemento *Radio* ou um elemento *Select* sem opção vazia. Para campos com essa característica não são gerados testes padrão do tipo `VALOR_FALTANTE`;

- Mensagens de erro para quando o campo recebe um valor incorreto (fora do formato, fora do limite especificado) ou quando o campo não é informado. Utilizado quando a mensagem exibida pelo sistema nessas situações é específica para o campo, não seguindo o padrão da mensagem global.

[3] Elementos rótulo (*label*) são associados a elementos da IU. Quando clica-se no rótulo, o elemento associado recebe o foco. A associação se dá por meio do atributo *for*, que deve referenciar o identificador (atributo *id*) do elemento correspondente.

A.3 Descrição do Dicionário de Dados (Valores Válidos)

A Listagem 63 exibe o trecho da DSL LUCT que trata da definição de um dicionário de dados.

Listagem 63 – Trecho da DSL LUCT para definição do dicionário de dados

```
1  ClasseDicionario = "Classe"  ID   [ "ClasseBase"  ID ]  "{"
       PropriedadeDicionario  { ","  PropriedadeDicionario }*  "}" ;
2
3  PropriedadeDicionario = ID  ":"  ( ValoresValidos |
4           "[" ClassesEquivalenciaDicionario  "]"  | PENDENTE ) ;
5
6  ClassesEquivalenciaDicionario = ClasseEquivalenciaDicionario  { ","
       ClasseEquivalenciaDicionario }* ;
7  ClasseEquivalenciaDicionario = ID  ":"  ValoresValidos ;
8
9  ValoresValidos = (TipoComLimiteInfSup | ListaDeValores | Regex | ExpressaoJS | SQL);
10
11 ListaDeValores = ( ListaDeValoresEnumerados | ListaDeValoresValorQualquerDaLista ) ;
12 ListaDeValoresEnumerados = "valores ("  STRING  { ","  STRING }*  ")" ;
13 ListaDeValoresValorQualquerDaLista = "valores da lista (1..n)" ;
14 TipoComLimiteInfSup = TipoDeDado
15     "min"  (STRING | ExpressaoJS | "valor armazenado" ID | "campo" ID )
16     "max"  (STRING | ExpressaoJS | "valor armazenado" ID | "campo" ID ) ;
17 TipoDeDado =  ( "Data" | "Texto" | "Numerico" ) ;
18 Regex = "expressao"  STRING ;
19 ExpressaoJS = "expressaoJS ("  STRING  ")" ;
20 SQL = "sql "  STRING ;
```

O conjunto de valores válidos de um item de dicionário pode ser definido a partir de informações do tipo do campo e seus limites mínimo e máximo, de uma lista de valores enumerados, da lista de opções carregadas para um campo de seleção, de uma expressão regular, de uma expressão JavaScript ou de um SQL. Cada uma dessas formas é descrita nas subseções a seguir.

A.3.1 Tipo do Campo e Limites Mínimo e Máximo

Valores válidos gerados por essa regra, de acordo com o tipo do campo, compreendem:

- Campo numérico: Todos os números inteiros dentro do limite, inclusive;
- Campo data: Todas as datas dentro do limite, inclusive;
- Campo texto: Textos com comprimento dentro do limite, inclusive. Os valores mínimo e máximo aqui referem-se ao número de caracteres do texto. Os caracteres utilizados na formação do texto são aleatórios.

Quando os limites mínimo e máximo permitidos para um campo dependem de valores informados para outros campos, pode-se descrever essa relação indicando, como limites mínimo ou máximo, uma expressão JavaScript que invoque uma função que receba esses campos como parâmetros e retorne o limite mínimo ou máximo calculado a partir dos parâmetros recebidos. Essa situação é especialmente comum para campos do tipo data, que têm relação de início e término. Funções para esse fim são disponibilizadas pelo ambiente.

A.3.2 Expressão Regular

Valores válidos gerados por essa regra compreendem todos os valores originados a partir de uma expressão regular. A derivação de valores permitidos por uma expressão regular é normalmente realizada transformando-a em um autômato finito, assim como ocorre no utilitário Xeger[4], incorporado à ferramenta para tal derivação.

A.3.3 Lista de Valores Enumerados

Valores válidos gerados por essa regra compreendem todos os valores enumerados em uma lista. Útil quando o conjunto de valores é de natureza estável.

A.3.4 Lista com Todos os Valores Carregados em Campo de Seleção

Valores válidos gerados por essa regra compreendem todas as opções carregadas em um campo de seleção. Verifica-se o número de opções exibidas no campo e seleciona-se uma delas, aleatoriamente.

Essa forma de indicação de valores válidos é útil quando o número de opções para o campo é alto – dificultando seu cadastramento como uma lista de valores enumerados – ou quando a natureza da lista de opções não é estável, e a opção a ser escolhida não gera diferença no direcionamento dos testes.

A.3.5 Expressão JavaScript

Valores válidos gerados por essa regra compreendem todos os valores retornados por uma expressão JavaScript, que pode conter um cálculo ou chamada a funções criadas pelo usuário ou disponibilizadas pelo ambiente. As funções disponibilizadas pelo ambiente possibilitam recuperar a data atual, somar dias/meses/anos a uma data, gerar um número aleatório dentro de um intervalo e gerar um CPF válido. Os valores preenchidos em campos anteriores ficam disponíveis para uso.

[4] Xeger: Biblioteca java para gerar valores randômicos a partir de expressões regulares. https://code.google.com/p/xeger/

A.3.6 SQL

Valores válidos gerados por essa regra compreendem todos os valores retornados na execução de um SQL. Os valores preenchidos em campos anteriores ficam disponíveis para uso.

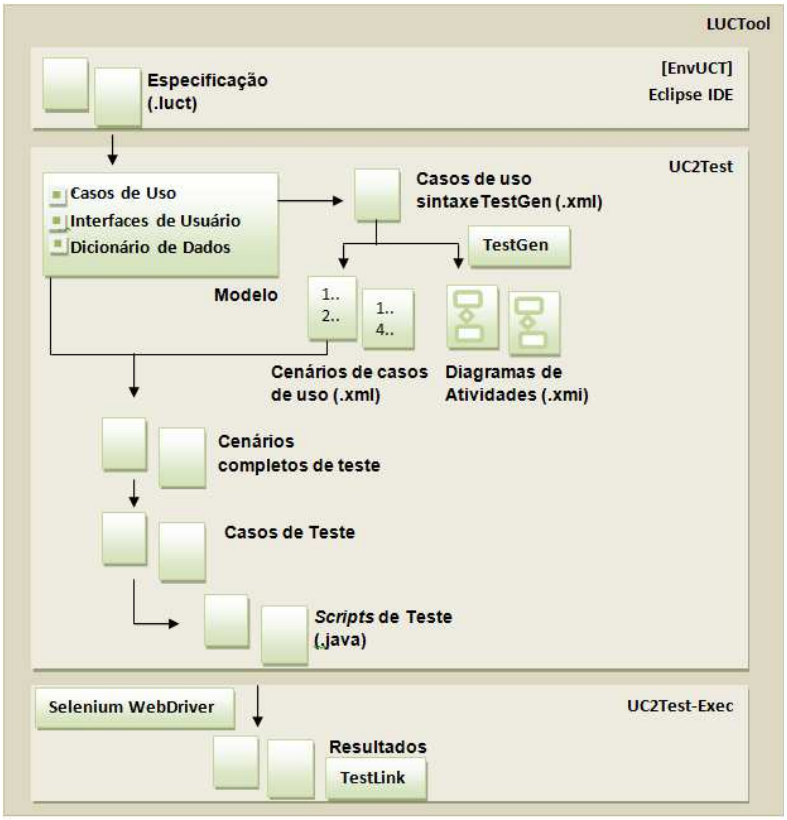
A.3.7 Algumas considerações

Os modos de geração “Expressão JavaScript” e “SQL” podem ser úteis quando o número de opções para o campo é alto ou quando a natureza da lista de opções não é estável, dificultando seu cadastramento como uma lista de valores enumerados. Porém, nesses casos, se a opção a ser escolhida não gerar diferença no direcionamento dos testes, deve ser considerado o modo de geração “Lista com Todos os Valores Carregados em Campo de Seleção”, para evitar um aumento de complexidade na especificação.

APÊNDICE B – LUCTOOL

Esta Seção apresenta a ferramenta LUCTool, desenvolvida para esta pesquisa. A ferramenta é composta pelo ambiente EnvUCT (*Environment for Use Cases to Tests*) – onde é realizada a especificação dos casos de uso, pelo módulo UC2Test (*Use Case to Test*) – que realiza a derivação dos casos de uso em testes, e o módulo UC2Test-Exec – que executa os *scripts* gerados e armazena seus resultados. A Figura 20 exibe a estrutura do ambiente proposto e os artefatos utilizados durante o processo.

Figura 20 – Estrutura e artefatos do ambiente proposto



A Seção B.2 apresenta o ambiente EnvUCT, a Seção B.2, o processo de geração dos *scripts* de teste automatizados, a Seção B.3, as possibilidades de execução dos testes, a Seção B.4, os detalhes técnicos de implementação da ferramenta e por fim, a Seção B.5, o algoritmo que gera as possibilidades de sequenciamento de casos de uso.

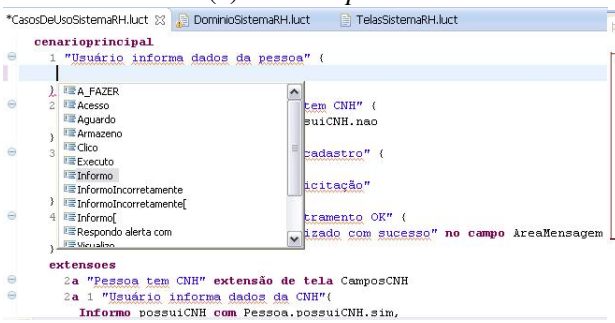
B.1 O Ambiente (EnvUCT)

Esta subseção apresenta o ambiente EnvUCT, implementado neste trabalho de pesquisa para a especificação dos casos de uso e demais componentes (interfaces de usuário e dicionário de dados) de acordo com as regras da DSL LUCT.

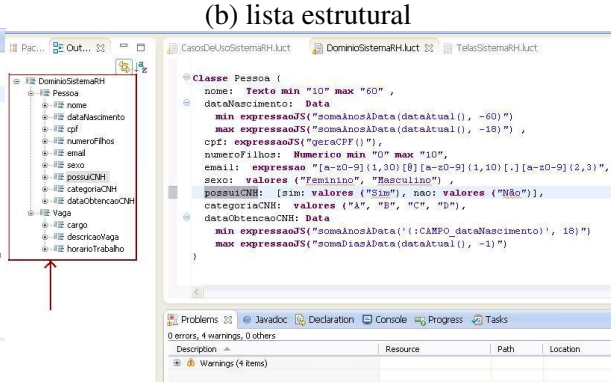
O ambiente é baseado no *framework* open-source Xtext (descrito na seção 2.4), por meio do qual obteve-se um editor específico para a linguagem LUCT, no IDE Eclipse. Dessa forma, o ambiente EnvUCT aproveita recursos característicos desse IDE, que favorecem a produtividade e a usabilidade, como: suporte a conclusão de código (*autocomplete* – Figura 21a), formatação, coloração de sintaxe, visão estrutural dos elementos (Figura 21b) e opção de exibição expandida ou resumida (*folding* – Figura 21c), além da facilidade no gerenciamento de versões dos artefatos.

Figura 21 – Ambiente EnvUCT

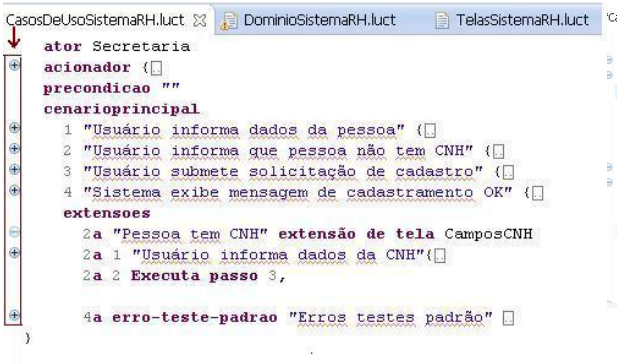
(a) *autocomplete*



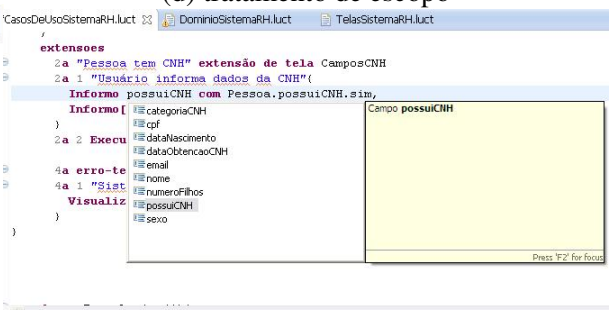
(b) lista estrutural



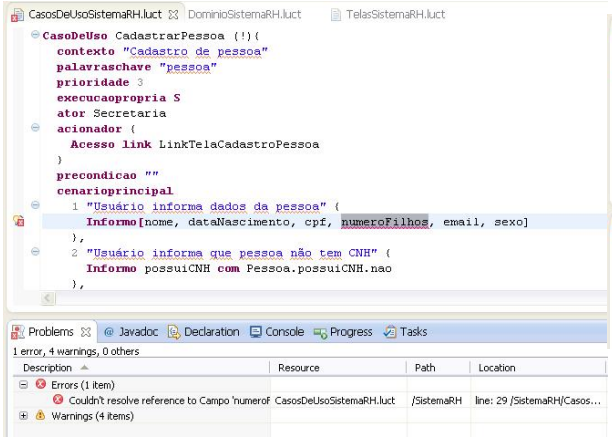
(c) *folding*



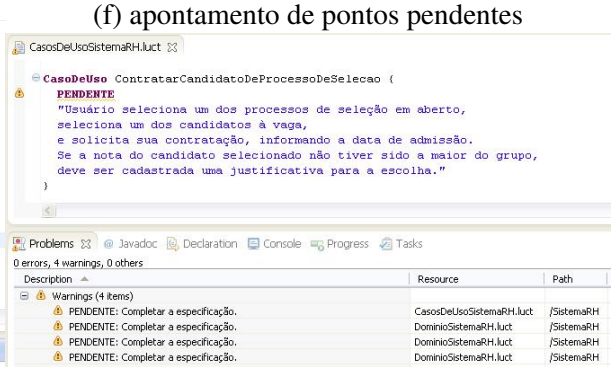
(d) tratamento de escopo



(e) apontamento de erro na especificação



(f) apontamento de pontos pendentes



O ambiente valida o texto informado em tempo de edição, de acordo com a gramática fornecida. Durante a entrada dos dados, são apresentadas as opções permitidas em cada ponto (recurso *autocomplete* – Figura 21a). Quando o ponto é de referência cruzada, são fornecidas as opções do tipo referenciado (Figura 21d).

Para pontos em que as opções permitidas são dependentes de contexto, foram implementadas regras de escopo, filtrando as opções exibidas e garantindo especificações corretas semanticamente. Assim, campos de uma interface podem ser referenciados em casos de uso que acessem essa interface (via menu ou *link*), enquanto campos de uma extensão de interface, somente em fluxos alternativos relacionados à mesma (Figura 21d).

O acesso à especificação de itens referenciados é simples – por meio de link no item referenciado, assim como a busca por um elemento – auxiliada por uma aba contendo a lista estrutural dos elementos presentes (Figura 21b).

Mudanças no dicionário, nas interfaces ou nos casos de uso, se gerarem quebra de referências, geram erros para o usuário, indicados no próprio texto e em uma aba que lista todos os erros e avisos do projeto. A Figura 21e exibe o apontamento de erro em um caso de uso que referencia um campo retirado da interface. Pontos em que a especificação deve ser complementada (elementos `PENDENTE`) são também apontados nessa aba (Figura 21f).

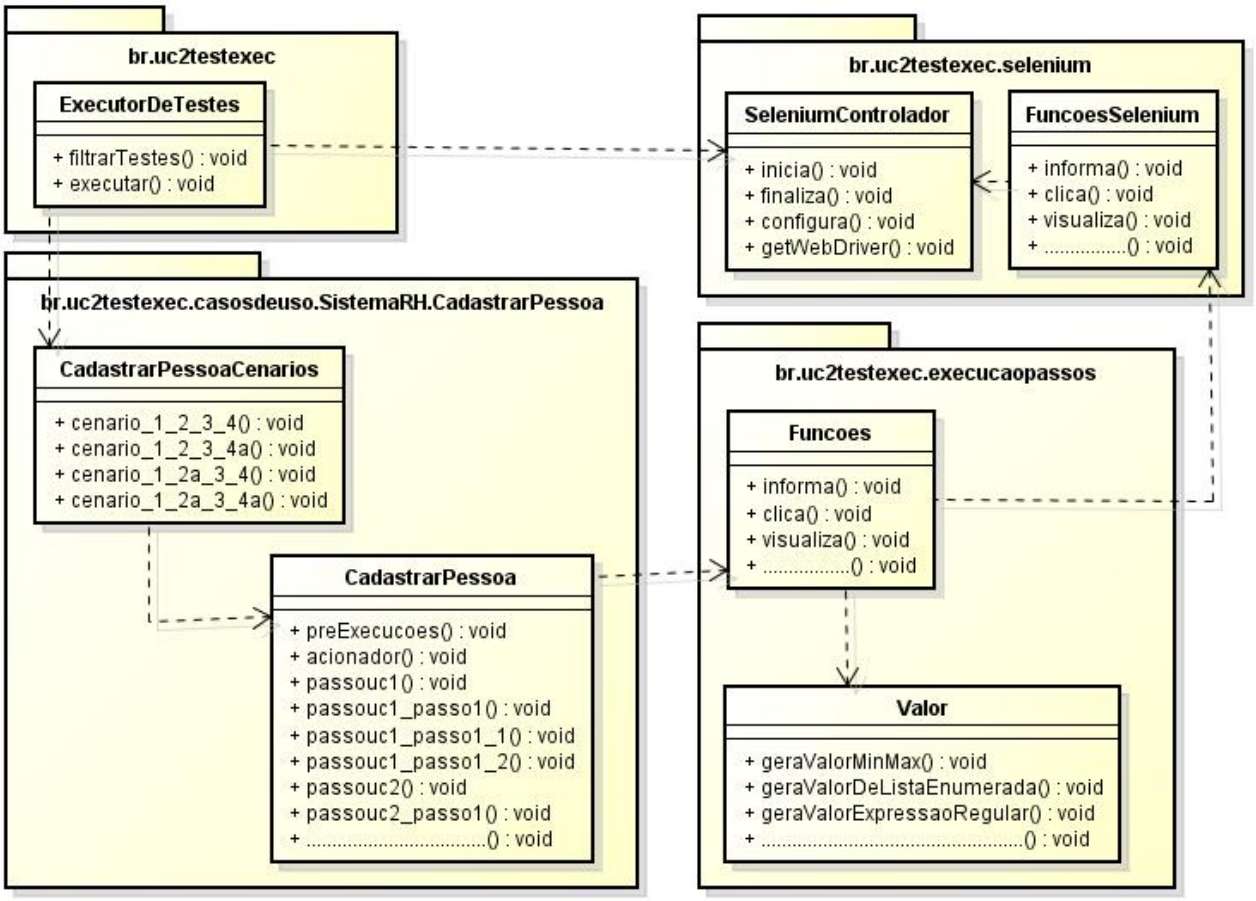
Devido às validações ocorrerem em tempo de edição, o autor pode refinar o caso de uso interativamente. A padronização e correção na escrita dos casos de uso são auxiliadas e guiadas pelo ambiente, melhorando sua qualidade e facilitando a identificação de seus componentes nas etapas posteriores, em que realiza-se a transformação automática do modelo em casos de teste.

B.2 Geração de Scripts de Teste

Para cada caso de uso, é gerado um pacote com o seguinte formato: br. uc2testexec. casosdeuso. <nome_sistema>. <nome_casodeuso>. A Figura 22 exibe a estrutura para a execução dos *scripts* de teste e o pacote `br.uc2testexec.casosdeuso.SistemaRH.CadastrarPessoa`, gerado para o caso de de uso `CadastrarPessoa`, apresentado na Listagem 13.

Os *scripts*, gerados em Java, refletem a estrutura de passos do caso de uso. O diagrama da Figura 22 exibe a estrutura da classe `CadastrarPessoa` e a Listagem 64, trecho do código correspondente aos passos 1 e 2 do caso de uso. Como pode ser observado, métodos são gerados para os passos do caso de uso (`passouc1`, `passouc2`, etc.), para os passos de detalhamento (`passouc1_passo1`, `passouc2_passo1`, etc.) e, caso o passo de detalhamento trate de múltiplas ações (como no tipo de ação `InformarCampos`, utilizado no `passouc1`), são gerados métodos para cada uma (`passouc1_passo1_1`, `passouc1_passo1_2`). O corpo

Figura 22 – Estrutura para a execução dos *scritps* de teste



de cada método é composto por invocações aos métodos que o desdobram, até que se chegue a métodos que solicitam a execução de interações com o SST.

Listagem 64 – Trecho de *script* de teste gerado para o caso de uso `CadastrarPessoa`

```
1  public static void passouc1() throws Throwable {
2          passouc1_passo1();
3  }
4  public static void passouc1_passo1() throws Throwable {
5          passouc1_passo1_1();
6          passouc1_passo1_2();
7  }
8  public static void passouc1_passo1_1() throws Throwable {
9          Funcoes.informar("infocampo_UC_CadastrarPessoa_passouc1_passo1_1");
10 }
11 public static void passouc1_passo1_2() throws Throwable {
12         Funcoes.informar("infocampo_UC_CadastrarPessoa_passouc1_passo1_2");
13 }
14
15
16 public static void passouc2() throws Throwable {
17         passouc2_passo1();
```

```
18 }
19 public static void passouc2_passo1() throws Throwable {
20         Funcoes.informar("infocampo_UC_CadastrarPessoa_passouc2_passo1");
21 }
22 ......
```

Optou-se por criar essa estrutura, em vez de gerar *scripts* Selenium diretamente, para possibilitar:

- a execução de testes com valores variados a cada execução, de acordo com a especificação;
- manter a estrutura dos casos de uso, gerando informação útil na geração de evidências de erro;
- separar as camadas do processo, favorecendo a possibilidade de inclusão, no futuro, de novas funcionalidades e novas ferramentas para a interação com o SST ou armazenamento dos resultados dos testes.

As informações para a execução de cada instrução – referentes à identificação do campo da IU e às regras de geração de valores válidos – são serializadas na geração dos testes e carregadas em memória durante sua execução, à medida da necessidade.

A execução dos *scripts* gerados passa por uma primeira camada que gera dinamicamente valores de acordo com a especificação e então repassa os pedidos de interação com a IU para a camada que interage com a API do SeleniumWebDriver. A primeira camada gera também valores inválidos para as situações de testes padrão de campos, além das mensagens esperadas em cada teste.

B.3 Opções de Execução

Há várias opções para a execução dos testes. Uma delas é selecionar os casos de teste de acordo com atributos especificados no caso de uso – indicando palavras-chave (*tags*), prioridade, ou a tela acessada. Outra possibilidade é solicitar a execução de casos de teste que tiveram sua última execução resultando em falha.

Já na execução de casos de teste por nível (de 1 a 3) – que refere-se à estrutura de identificadores dos casos de teste (Subseção 3.3.2.3) – é selecionado um caso de teste (aleatório) por nível solicitado, ou seja, se indicado o nível 1, seleciona-se um caso de teste por caso de uso, e assim por diante. Por fim, há a opção de solicitar a execução de casos de teste de *ids* específicos ou um conjunto de casos de teste, informando um *id* de nível mais alto.

A execução considerada o tipo de teste mais comum é a execução com o nível 3 (um caso de teste por sequenciamento). Considera-se que testes completos (em que verifica-se todas as situações abordadas nos testes padrão relacionados aos campos – detalhes na Subseção 3.3.2.2) têm sua importância no momento em que um caso de uso ou cenário é disponibilizado para testes, ou antes da liberação de uma versão do sistema. Nos demais momentos, sendo desejável a redução do número de testes a executar visando diminuir o tempo de execução (situação comum para sistemas com muitas funcionalidades), a opção de execução por nível mostra-se interessante, uma vez que a cobertura total pode ser atingida com o decorrer do tempo, em execuções sucessivas, de forma incremental.

B.4 Detalhes Técnicos

Esta seção apresenta alguns detalhes técnicos sobre a ferramenta LUCTool. Para o desenvolvimento (java) dos módulos UC2Test e UC2Test-Exec, e também para o desenvolvimento da linguagem LUCT e sua posterior execução, utilizou-se o IDE Eclipse, versão Kepler, na opção DSLTools – que já é disponibilizada com o *plugin* Xtext.

Para a execução dos testes foi utilizado o navegador Firefox e para a interação com os elementos das interfaces, a API Java do Selenium WebDriver. As versões dos dois componentes devem ser compatíveis, e as mesmas foram mantidas atualizadas durante o projeto. Ao seu término, encontravam-se nas versões 36.0 e 2.45.0, respectivamente.

Para o armazenamento e gerenciamento dos testes foi utilizada a ferramenta TestLink (versão 1.9.9). Para que os casos de teste e resultados de execuções fossem armazenados por meio de automação, foi utilizada a API-cliente Java do TestLink, com a versão 1.9.4.

A execução de expressões regulares – demandada quando há especificação do tipo `Expressão Regular` – foi suportada pelo utilitário Xeger, na sua versão 1.0. Já a execução de código JavaScript – demandada quando há especificação do tipo `Expressão JavaScript` – foi suportada pela própria JVM, que contém uma *JavaScript Engine*. Para o armazenamento dos dados em disco foi utilizado o utilitário Xtream[1], na sua versão 1.4.7.

B.5 Algoritmo de Geração de Sequenciamento entre Casos de Uso

Para cada cenário de caso de uso são selecionados os cenários possíveis para cada ponto de invocação, e então, por meio da combinação desses cenários, são geradas todas as possibilidades de sequenciamento, ou cenários completos de teste. O pseudocódigo completo para essa geração é apresentado na Listagem 65.

[1] Xtream: Utilitário para transformação de classes Java em XML. <http://xstream.codehaus.org/>

Listagem 65 – Algoritmo que gera as possibilidades de sequenciamento de casos de uso

```
1  FUNCTION GeraSequenciamentosPorCenarioDeCasoDeUso()
2  BEGIN
3    FOR EACH casoDeUso IN casosDeUso DO
4      cenarios <- casoDeUso.cenarios
5      FOR EACH cenario IN cenarios DO
6            listaDeInvocacoesDeUCs <- CALL CarregaListaDeInvocacaoDeUCs(cenario)
7        cenario.nroInvocacoesDeUC <- listaDeInvocacoesDeUCs.size
8        listaDeOpcoesDeExecucaoDeUCInvocados <- CALL
             GeraOpcoesDeExecucaoDeUCsInvocados()
9        cenario.listaDeOpcoesDeExecucaoDeUCInvocados <-
             listaDeOpcoesDeExecucaoDeUCInvocados
10       cenario.sequenciamentos <- CREATE sequenciamento list
11       CALL cenario.GeraCombinacoes( 1, NULL)
12     END FOR
13   END FOR
14 END
15
16 FUNCTION CarregaListaDeInvocacaoDeUCs (cenario)
17 BEGIN
18   listaDeInvocacoesDeUCs <- CREATE ExecutarUC List
19   //Carregando lista de invocações de casos de uso cadastradas nas pré-condições
20   preCondicoes <- cenario.preCondicoes
21   FOR EACH preCondicao IN preCondicoes DO
22     IF(preCondicao IS ExecutarUC)
23       executarUC <- preCondicao
24       ADD executarUC TO listaDeInvocacoesDeUCs
25     END IF
26   END FOR
27   //Carregando lista de invocações de casos de uso cadastradas nos passos
28   passos <- cenario.passos
29   FOR EACH passo IN passos DO
30     IF(passo IS ExecutarUC)
31       executarUC <- passo
32       ADD executarUC TO listaDeInvocacoesDeUCs
33     END IF
34   END FOR
35   RETURN listaDeInvocacoesDeUCs
36 END
37
38 FUNCTION GeraOpcoesDeExecucaoDeUCsInvocados(listaExecutarUC)
39 BEGIN
40   //Gera as opções de execução para cada invocação
41   nroInvocacoesDeUC <- listaDeInvocacoesDeUCs.size;
42   listaDeOpcoesDeExecucaoDeUCInvocados <- CREATE OpcoesExecucaoDeUCInvocado list
43   FOR indiceInvocacao = 1 TO nroInvocacoesDeUC
44     executarUC <- listaExecutarUC[indiceInvocacao]
45     nomeCasoUsoExecutar <- executarUC.nomeCasoUso
46     extensoesCasoUsoExecutar <- executarUC.extensoesCasoUso
47     IF(extensoesCasoUsoExecutar == NULL)
48       opcoesExecucaoDoUCInvocado <- CasosUso.getCenariosSucesso(nomeCasoUsoExecutar)
49     ELSE
50       opcoesExecucaoDoUCInvocado <-
51       CasosUso.getCenariosComExtensao(nomeCasoUsoExecutar, extensoesCasoUsoExecutar)
52     END IF
53     listaDeOpcoesDeExecucaoDeUCInvocados[indiceInvocacao] <-
```

```
        opcoesExecucaoDoUCInvocado
  END FOR
  RETURN listaDeOpcoesDeExecucaoDeUCInvocados
END

FUNCTION GeraCombinacoes (indiceInvocacao, opcaoExecucaoDeUCAnterior)
BEGIN
  //Gera combinações entre a invocação anterior e as opções de invocação do índice
      informado
  opcoesExecucaoDoUCInvocado <- THIS.listaDeOpcoesDeExecucaoDeUCInvocados[
      indiceInvocacao]
  nroOpcoesExecucaoDoUCInvocado <- opcoesExecucaoDoUCInvocado.size
  FOR EACH opcaoExecucaoDoUCInvocado IN opcoesExecucaoDoUCInvocado
    IF opcaoExecucaoDeUCAnterior IS NOT NULL
      opcaoExecucaoDoUCInvocado.invocacaoAnterior <- opcaoExecucaoDeUCAnterior
    END IF
    IF indiceInvocacao < THIS.nroInvocacoesDeUC
      CALL GeraCombinacoes (indiceInvocacao + 1, opcaoExecucaoDoUCInvocado)
    ELSE //Última invocação
      CALL SalvaSequenciamento (opcaoExecucaoDoUCInvocado)
    END IF
  END FOR
END

FUNCTION SalvaSequenciamento (opcaoExecucaoDoUCInvocado)
BEGIN
  //Salva sequenciamento na lista de sequenciamentos do cenário
  sequenciamento <- CREATE opcaoExecucaoDoUCInvocado list
  ADD opcaoExecucaoDoUCInvocado TO sequenciamento
  opcaoExecucaoDeUCAnterior <- opcaoExecucaoDoUCInvocado.invocacaoAnterior
  WHILE opcaoExecucaoDeUCAnterior IS NOT NULL
    ADD opcaoExecucaoDeUCAnterior TO sequenciamento
    opcaoExecucaoDeUCAnterior <- opcaoExecucaoDeUCAnterior.invocacaoAnterior
  END WHILE
  ADD sequenciamento.IN_REVERSE_ORDER TO THIS.sequenciamentos
END
```

APÊNDICE C – ESPECIFICAÇÃO – SISTEMA DE INFORMAÇÕES DE EXTENSÃO

Esta seção exibe a especificação completa dos casos de uso, interfaces de usuário e dicionário de dados utilizados nos experimentos apresentados no Capítulo 4.

As Listagens 66, 67, 68 e 69 exibem casos de uso utilizados durante os experimentos apresentados na Seção 4.3, para o Sistema de Informações de Extensão. Os experimentos envolveram especificar os casos de uso selecionados no ambiente EnvUCT a partir da documentação existente – adaptando a especificação quando necessário – até a obtenção de um conjunto de testes que funcionasse adequadamente. Em seguida, trabalhou-se na verificação da eficácia dos testes gerados, por meio de alterações na especificação e no SST (uso de mutantes).

Listagem 66 – Caso de uso `InserirCurso`

```
1  CasoDeUso InserirCurso (!){
2    execucaopropria S
3    ator Docente
4
5    acionador {
6      Acesso link LinkInserirCurso
7    }
8    cenarioprincipal
9      1 "Usuário informa campos do novo curso (informações gerais de ação de extensão)
           " {
10          Informo titulo,
11          Informo[dataInicio,previsaoTermino],
12          Informo anoPrimeiraApresentacao,
13          Informo[unidade],
14          Informo[departamento],
15          Informo[areaTematicaPrincipal],
16          Informo[areaTematicaAfim],
17          Informo[linhaExtensao,grandeArea],
18          Informo[bolsaExterna,bolsaFAPEMIG,bolsaFUMP],
19          Informo[bolsaInstitucional,bolsaPBEXT,bolsaPROEXT],
20          Informo[palavrasChave],
21          Informo[apresentacaoJustificativa,objetivosGerais,objetivosEspecificos],
22          Informo[metodologia,formasAvaliacao],
23          Informo[site,origemPublicoAlvo,caracterizacaoPublicoAlvo],
24          Informo[vinculoEdital,vinculoPoliticaPublica],
25          Informo[planoAtividades,planoAcompanhamento,processoAvaliacao],
26          Informo informacoesAdicionais
27          },
28     2 "Usuário informa que novo curso é de caracterização geral e a carga horária" {
29          Informo caracterizacao com Curso.caracterizacao.geral,
30          Informo cargaHoraria com Curso.cargaHoraria.geral
31     },
32     3 "Usuário informa demais campos do novo curso (informações específicas de
          cursos)" {
33          Informo[subCaracterizacao, programa,projeto],
```

```
34          Informo[estruturaCurricular,infraEstruturaFisica,formaAvaliacaoAluno],
35          Informo[localInscricao,gratuito,publicoEstimado]
36      },
37      4 "Usuário solicita inclusão do novo curso" {
38          Clico Salvar
39      },
40      5 "Sistema exibe mensagem de sucesso" {
41          Visualizo mensagem DadosDoCursoSalvos
42      }
43      extensoes
44        1a "Usuário informa somente o título do curso"
45        1a 1 "Usuário informa o título do curso" {
46          Informo titulo
47        }
48        1a 2 Executa passo 4,
49
50        2a "Novo curso é de aperfeiçoamento"
51        2a 1 "Usuário informa que novo curso é de aperfeiçoamento e a carga horária"
                {
52          Informo caracterizacao com Curso.caracterizacao.aperfeicoamento,
53          Informo cargaHoraria com Curso.cargaHoraria.aperfeicoamento
54        }
55        2a 2 Executa passo 3,
56
57        5a erro-teste-padrao "Usuário informa dados incorretos"
58        5a 1 "Sistema exibe mensagem de erro" {
59          Visualizo uma mensagem de erro de testes padrão no campo AreaMensagemErro
60        }
61
62        garantiasdesucesso "Curso inserido - Data início e fim armazenados" {
63          Armazeno campo dataInicio como dataInicioCurso,
64          Armazeno campo previsaoTermino como dataTerminoCurso
65        }
66  }
```

Listagem 67 – Caso de uso `InserirParceria`

```
1   CasoDeUso InserirParceria (!) {
2     execucaopropria S
3     ator Docente
4     acionador {
5       Acesso link LinkTelaParcerias
6     }
7     precondicao "Usuário insere novo curso" {
8       Executo InserirCurso  com extensao = "1a"
9     }
10    cenarioprincipal
11      1 "Usuário informa dados da nova parceria" {
12        Informo[nome,caracterizacao,formaDeParceria]
13      },
14      2 "Usuário solicita inclusão da parceria" {
15        Clico botaoIncluir
16      },
17      4 "Sistema exibe mensagem de sucesso" {
18        Visualizo mensagem ParceriaSalva
19      }
20
```

```
    extensoes
      3a erro-teste-padrao "Usuário informa dados incorretos"
      3a 1 "Sistema exibe mensagem de erro" {
        Visualizo uma mensagem de erro de testes padrão no campo AreaMensagemErro
      }
}
```

Listagem 68 – Caso de uso `InserirAbrangencia`

```
CasoDeUso InserirAbrangencia (!) {
  execucaopropria S
  ator Docente
  acionador {
    Acesso link LinkTelaAbrangencia
  }
  precondicao "Usuário insere novo curso" {
    Executo InserirCurso com extensao = "1a"
  }
  cenarioprincipal
    1 "Usuário informa dados da nova abrangência" {
      Informo[identificacao,estado,municipio,detalhes,cep]
    },
    2 "Usuário solicita inclusão da abrangência" {
      Clico botaoIncluir
    },
    3 "Sistema exibe mensagem de sucesso" {
      Visualizo mensagem AbrangenciaSalva
    }
  extensoes
    3a erro-teste-padrao "Usuário informa dados incorretos"
    3a 1 "Sistema exibe mensagem de erro" {
      Visualizo uma mensagem de erro de testes padrão no campo AreaMensagemErro
    }
}
```

Listagem 69 – Caso de uso `AlterarCursoESubmeterAAprovacao`

```
CasoDeUso AlterarCursoESubmeterAAprovacao (!){
  execucaopropria S
  ator Docente

  acionador {
    Acesso link LinkTelaAlterarCurso
  }
  precondicao "" {
    Acesso link LinkInserirCurso,
    Executo InserirCurso com extensao = "1a",
    Acesso link LinkTelaAbrangencia,
    Executo InserirAbrangencia,
    Acesso link LinkTelaParcerias,
    Executo InserirParceria
  }
  cenarioprincipal
    1 "Usuário altera campos do curso (informações gerais de ação de extensão)" {
        Informo[dataInicio,previsaoTermino],
        Informo anoPrimeiraApresentacao,
        Informo[unidade],
```

```
21          Informo[departamento],
22          Informo[areaTematicaPrincipal],
23          Informo[areaTematicaAfim],
24          Informo[linhaExtensao,grandeArea],
25          Informo[bolsaExterna,bolsaFAPEMIG,bolsaFUMP],
26          Informo[bolsaInstitucional,bolsaPBEXT,bolsaPROEXT],
27          Informo[palavrasChave],
28          Informo[apresentacaoJustificativa,objetivosGerais,objetivosEspecificos],
29          Informo[metodologia,formasAvaliacao],
30          Informo[site,origemPublicoAlvo,caracterizacaoPublicoAlvo],
31          Informo[vinculoEdital,vinculoPoliticaPublica],
32          Informo[planoAtividades,planoAcompanhamento,processoAvaliacao],
33          Informo informacoesAdicionais
34      },
35      2 "Usuário informa que novo curso é de caracterização geral e a carga horária" {
36          Informo caracterizacao com Curso.caracterizacao.geral,
37          Informo cargaHoraria com Curso.cargaHoraria.geral
38      },
39      3 "Usuário altera demais campos do curso (informações específicas de cursos)" {
40          Informo[subCaracterizacao, programa,projeto],
41          Informo[estruturaCurricular,infraEstruturaFisica,formaAvaliacaoAluno],
42          Informo[localInscricao,gratuito,publicoEstimado]
43      },
44      4 "Usuário solicita alteração e submissão à aprovação do curso" {
45          Clico Submeter
46      },
47      5 "Sistema exibe mensagem de sucesso" {
48          Visualizo mensagem SubmissaoFeitaComSucesso no campo AreaMensagemSucesso
49      }
50
51      extensoes
52        2a "Curso é de aperfeiçoamento"
53        2a 1 "Usuário informa que curso é de aperfeiçoamento e a carga horária"   {
54          Informo caracterizacao com Curso.caracterizacao.aperfeicoamento,
55          Informo cargaHoraria com Curso.cargaHoraria.aperfeicoamento
56        }
57        2a 2 Executa passo 3,
58
59        5a erro-teste-padrao "Usuário informa dados incorretos"
60        5a 1 "Sistema exibe mensagem de erro" {
61          Visualizo uma mensagem de erro de testes padrão no campo AreaMensagemErro
62        }
63  }
```

Já a Listagem 70, assim como a Listagem 66 anteriormente apresentada, exibem casos de uso utilizados como exemplos ao longo da Seção 4.2, em que se avaliou a expressividade da DSL LUCT por meio da análise de sua aplicação prática, utilizando casos de uso de sistemas de software reais.

Listagem 70 – Caso de uso `InserirParticipacao`

```
1  CasoDeUso InserirParticipacao (!) {
2    execucaopropria S
3    ator Docente
4    acionador {
```

```
5     Acesso link LinkTelaEquipe
6   }
7   precondicao "Usuário insere novo curso"       {
8     Executo InserirCurso com extensao = "1a"
9   }
10  cenarioprincipal
11  1 "Usuário informa dados de participação" {
12    Informo categoria,
13    Informo nome com Participacao.parteValidaDeNome,
14    Informo listaDeNomes,
15    Informo[datainicio, dataTermino]
16  },
17  2 "Usuário submete pedido de inclusão" {
18   Clico botaoincluir
19  },
20  3 "Sistema exibe mensagem de cadastramento feito com sucesso" {
21    Visualizo mensagem MSG_PARTICIPACAO_INCLUIDA no campo AreaMensagemSucesso
22  }
23  extensoes
24    3a erro-teste-padrao "Usuário informa dados incorretos"
25    3a 1 "Sistema exibe mensagem de erro" {
26      Visualizo uma mensagem de erro de testes padrão no campo AreaMensagemErro
27    }
28 }
```

Por fim, a Listagem 71 exibe a descrição das interfaces de usuário relacionadas aos casos de uso apresentados neste apêndice, e a Listagem 72, o dicionário de dados correspondente.

Listagem 71 – Interfaces de Usuário

```
1  MensagensErroTestesPadrao {
2    MSG_ERRO_CAMPO_OBRIGATORIO = "O campo {:LABEL_CAMPO_TESTE_PADRAO} é obrigatório e
         não foi preenchido!"
3    MSG_ERRO_CAMPO_ABAIXO_LIMITE ="Campo {:ID_CAMPO_TESTE_PADRAO} abaixo do limite."
4    MSG_ERRO_CAMPO_ACIMA_LIMITE = "Campo {:ID_CAMPO_TESTE_PADRAO} acima do limite."
5    MSG_ERRO_CAMPO_FORA_FORMATO = "Campo {:ID_CAMPO_TESTE_PADRAO} fora do formato."
6  }
7
8  Link LinkInserirCurso texto="Inserir Curso" tela= TelaInserirCurso
9  Link LinkTelaAbrangencia texto="Abrangência" tela= TelaAbrangencia
10 Link LinkTelaParcerias texto="Parcerias" tela= TelaParceria
11 Link LinkTelaAlterarCurso texto="Descrição" tela= TelaAlterarCurso
12 Link LinkTelaEquipe texto="Equipe" tela= TelaEquipe
13
14
15 Tela TelaInserirCurso {
16
17   * titulo: CAMPO_TEXTO label = "título" valores = Curso.titulo ,
18   dataInicio: CAMPO_TEXTO label = "Data de Início"  valores = Curso.dataInicio
19   [MSG_ERRO_CAMPO_ABAIXO_LIMITE ="A data de início é muito antiga. São aceitas datas
          a partir de 01/01/1950"],
20   previsaoTermino: CAMPO_TEXTO label = "Previsão de Término"   valores = Curso.
         previsaoTermino [MSG_ERRO_CAMPO_ABAIXO_LIMITE ="A data do campo Data Início é
         superior ou igual à data do campo Previsão de término"],
21   anoPrimeiraApresentacao: CAMPO_TEXTO label = "Ano em que se iniciou a ação"
         valores = Curso.anoPrimeiraApresentacao
```

```
22          [MSG_ERRO_CAMPO_ABAIXO_LIMITE ="Ano em que se iniciou a ação inválido(a)"],
23      unidade: CAMPO_LISTA label = "Unidade" valores = Curso.unidade,
24      departamento: CAMPO_LISTA label = "Departamento"  valores = Curso.departamento (
            carrega valores após unidade),
25      areaTematicaPrincipal: CAMPO_LISTA label = "Principal Área Temática de Extensão"
            valores = Curso.areaTematicaPrincipal,
26      areaTematicaAfim: CAMPO_LISTA label = "Área Temática de Extensão Afim"  valores =
            Curso.areaTematicaAfim (carrega valores após areaTematicaPrincipal),
27      linhaExtensao: CAMPO_LISTA label = "Linha de Extensão"  valores = Curso.
            linhaExtensao,
28      grandeArea: CAMPO_LISTA label = "Grande Área do Conhecimento"  valores = Curso.
            grandeArea,
29      bolsaPBEXT: CAMPO_TEXTO label = "PBEXT"   valores = Curso.bolsaPBEXT,
30      bolsaFUMP: CAMPO_TEXTO label = "FUMP"   valores = Curso.bolsaFUMP ,
31      bolsaInstitucional: CAMPO_TEXTO label = "Institucional da PROEX"   valores = Curso
            .bolsaInstitucional ,
32      bolsaPROEXT: CAMPO_TEXTO label = "PROEXT"   valores = Curso.bolsaPROEXT ,
33      bolsaFAPEMIG: CAMPO_TEXTO label = "FAPEMIG"  valores = Curso.bolsaFAPEMIG ,
34      bolsaExterna: CAMPO_TEXTO label = "Outras"   valores = Curso.bolsaExterna,
35      palavrasChave: CAMPO_TEXTO label = "Palavras-Chave" valores = Curso.palavrasChave,
36      apresentacaoJustificativa: CAMPO_TEXTO label = "Apresentação e justificativa"
            valores = Curso.apresentacaoJustificativa,
37      objetivosGerais: CAMPO_TEXTO label = "Objetivos gerais"  valores = Curso.
            objetivosGerais,
38      objetivosEspecificos: CAMPO_TEXTO label = "Objetivos específicos" valores = Curso.
            objetivosEspecificos,
39      metodologia: CAMPO_TEXTO label = "Metodologia"  valores = Curso.metodologia,
40      formasAvaliacao: CAMPO_TEXTO label = "Forma de avaliação"  valores = Curso.
            formasAvaliacao
41          [MSG_ERRO_CAMPO_OBRIGATORIO = "O campo Formas de avaliação da ação de
                Extensão é obrigatório e não foi preenchido!"],
42      site: CAMPO_TEXTO label = "Site"  valores = Curso.site ,
43      origemPublicoAlvo: CAMPO_LISTA label = "Origem do público-alvo"   valores = Curso.
            origemPublicoAlvo,
44      caracterizacaoPublicoAlvo: CAMPO_TEXTO label = "Caracterização do público-alvo"
            valores = Curso.caracterizacaoPublicoAlvo ,
45      vinculoEdital: CAMPO_RADIO label = "Captação por edital de fomento" valores =
            Curso.vinculoEdital,
46      vinculoPoliticaPublica: CAMPO_RADIO label = "Articulado com política pública"
            valores = Curso.vinculoPoliticaPublica,
47      planoAtividades: CAMPO_TEXTO label = "Plano de atividades previstas"  valores =
            Curso.planoAtividades,
48      planoAcompanhamento: CAMPO_TEXTO label = "Plano de acompanhamento e orientação"
            valores = Curso.planoAcompanhamento ,
49      processoAvaliacao: CAMPO_TEXTO label = "Processo de avaliação"  valores = Curso.
            processoAvaliacao ,
50      informacoesAdicionais: CAMPO_TEXTO label = "Informações adicionais"  valores =
            Curso.informacoesAdicionais,
51
52      //Dados especificos de curso
53      caracterizacao: CAMPO_LISTA  label = "Caracterização"  valores = Curso.
            caracterizacao,
54      subCaracterizacao: CAMPO_LISTA  label = "SubCaracterização"  valores = Curso.
            subCaracterizacao (carrega valores após caracterizacao),
55      programa: CAMPO_LISTA  label = "Programa vinculado"  valores = Curso.programa,
56      projeto: CAMPO_LISTA  label = "Projeto vinculado"  valores = Curso.projeto,
57      estruturaCurricular: CAMPO_TEXTO  label = "Estrutura curricular"  valores = Curso.
```

```
        estruturaCurricular,
58    infraEstruturaFisica: CAMPO_TEXTO  label = "Infra-estrutura física"  valores =
        Curso.infraEstruturaFisica,
59    formaAvaliacaoAluno: CAMPO_TEXTO  label = "Forma de avaliação do aluno do curso"
        valores = Curso.formaAvaliacaoAluno,
60    localInscricao: CAMPO_TEXTO  label = "Local de inscrição"  valores = Curso.
        localInscricao,
61    gratuito: CAMPO_TEXTO  label = "Gratuito para o público"  valores = Curso.gratuito
        ,
62    cargaHoraria: CAMPO_TEXTO  label = "Carga horária"  valores = Curso.cargaHoraria
63          [MSG_ERRO_CAMPO_ABAIXO_LIMITE ="Carga horária não pode ser inferior a"],
64    publicoEstimado: CAMPO_TEXTO  label = "Público estimado/vagas"  valores = Curso.
        publicoEstimado,
65
66    Salvar: CAMPO_BOTAO label = "Salvar e Avançar",
67    Submeter: CAMPO_BOTAO label = "Submeter à Aprovação"
68
69    Mensagens {
70      DadosDoCursoSalvos "Dados da Descrição salvos com sucesso"
71    }
72  }
73
74
75
76  Tela TelaAlterarCurso {
77
78    * titulo: CAMPO_TEXTO label = "Título"  valores = Curso.titulo ,
79    * dataInicio: CAMPO_TEXTO label = "Data de Início"   valores = Curso.dataInicio
80          [MSG_ERRO_CAMPO_ABAIXO_LIMITE ="A data de início é muito antiga. São aceitas
                datas a partir de 01/01/1950"],
81    * previsaoTermino: CAMPO_TEXTO label = "Previsão de Término"   valores = Curso.
        previsaoTermino
82          [MSG_ERRO_CAMPO_ABAIXO_LIMITE ="A data do campo Data Início é superior ou
              igual à data do campo Previsão de término"],
83    anoPrimeiraApresentacao: CAMPO_TEXTO label = "Ano em que se iniciou a ação"
        valores = Curso.anoPrimeiraApresentacao
84           [MSG_ERRO_CAMPO_ABAIXO_LIMITE ="Ano em que se iniciou a ação inválido(a)"],
85    * unidade: CAMPO_LISTA label = "Unidade"  valores = Curso.unidade,
86    * departamento: CAMPO_LISTA label = "Departamento"  valores = Curso.departamento (
        carrega valores após unidade),
87    * areaTematicaPrincipal: CAMPO_LISTA label = "Principal Área Temática de Extensão"
          valores = Curso.areaTematicaPrincipal,
88    areaTematicaAfim: CAMPO_LISTA label = "Área Temática de Extensão Afim"  valores =
        Curso.areaTematicaAfim (carrega valores após areaTematicaPrincipal),
89    * linhaExtensao: CAMPO_LISTA label = "Linha de Extensão" valores = Curso.
        linhaExtensao,
90    * grandeArea: CAMPO_LISTA label = "Grande Área do Conhecimento"  valores = Curso.
        grandeArea,
91    bolsaPBEXT: CAMPO_TEXTO label = "PBEXT"   valores = Curso.bolsaPBEXT
        preenchimento forçado,
92    bolsaFUMP: CAMPO_TEXTO label = "FUMP"   valores = Curso.bolsaFUMP  preenchimento
        forçado,
93    bolsaInstitucional: CAMPO_TEXTO label = "Institucional da PROEX"   valores = Curso
        .bolsaInstitucional  preenchimento forçado,
94    bolsaPROEXT: CAMPO_TEXTO label = "PROEXT"   valores = Curso.bolsaPROEXT
        preenchimento forçado,
95    * bolsaFAPEMIG: CAMPO_TEXTO label = "FAPEMIG"  valores = Curso.bolsaFAPEMIG
```

```
96          [MSG_ERRO_CAMPO_OBRIGATORIO ="O campo Quantidade de bolsas de Extensão é
                obrigatório e não foi preenchido!"],
97    bolsaExterna: CAMPO_TEXTO label = "Outras"   valores = Curso.bolsaExterna
          preenchimento forçado,
98    * palavrasChave: CAMPO_TEXTO label = "Palavras-Chave"  valores = Curso.
          palavrasChave,
99    * apresentacaoJustificativa: CAMPO_TEXTO label = "Apresentação e justificativa"
          valores = Curso.apresentacaoJustificativa,
100   * objetivosGerais: CAMPO_TEXTO label = "Objetivos gerais"  valores = Curso.
          objetivosGerais,
101   objetivosEspecificos: CAMPO_TEXTO label = "Objetivos específicos"  valores = Curso
          .objetivosEspecificos,
102   * metodologia: CAMPO_TEXTO label = "Metodologia"  valores = Curso.metodologia,
103   * formasAvaliacao: CAMPO_TEXTO label = "Forma de avaliação"  valores = Curso.
          formasAvaliacao
104         [MSG_ERRO_CAMPO_OBRIGATORIO = "O campo Formas de avaliação da ação de
                Extensão é obrigatório e não foi preenchido!"],
105   site: CAMPO_TEXTO label = "Site"   valores = Curso.site ,
106   * origemPublicoAlvo: CAMPO_LISTA label = "Origem do público-alvo"   valores =
          Curso.origemPublicoAlvo,
107   * caracterizacaoPublicoAlvo: CAMPO_TEXTO label = "Caracterização do público-alvo"
            valores = Curso.caracterizacaoPublicoAlvo ,
108   vinculoEdital: CAMPO_RADIO label = "Captação por edital de fomento"  valores =
          Curso.vinculoEdital  preenchimento forçado,
109   vinculoPoliticaPublica: CAMPO_RADIO label = "Articulado com política pública"
          valores = Curso.vinculoPoliticaPublica  preenchimento forçado,
110   planoAtividades: CAMPO_TEXTO label = "Plano de atividades previstas"  valores =
          Curso.planoAtividades,
111   planoAcompanhamento: CAMPO_TEXTO label = "Plano de acompanhamento e orientação"
            valores = Curso.planoAcompanhamento ,
112   processoAvaliacao: CAMPO_TEXTO label = "Processo de avaliação"   valores = Curso.
          processoAvaliacao ,
113   informacoesAdicionais: CAMPO_TEXTO label = "Informações adicionais"  valores =
          Curso.informacoesAdicionais,
114
115   //Dados especificos de curso
116   * caracterizacao: CAMPO_LISTA  label = "Caracterização"  valores = Curso.
          caracterizacao,
117   * subCaracterizacao: CAMPO_LISTA  label = "SubCaracterização"  valores = Curso.
          subCaracterizacao (carrega valores após caracterizacao),
118   programa: CAMPO_LISTA  label = "Programa vinculado" valores = Curso.programa,
119   projeto: CAMPO_LISTA  label = "Projeto vinculado"  valores = Curso.projeto,
120   * estruturaCurricular: CAMPO_TEXTO  label = "Estrutura curricular"  valores =
          Curso.estruturaCurricular,
121   * infraEstruturaFisica: CAMPO_TEXTO  label = "Infra-estrutura física"  valores =
          Curso.infraEstruturaFisica,
122   * formaAvaliacaoAluno: CAMPO_TEXTO  label = "Forma de avaliação do aluno do curso"
            valores = Curso.formaAvaliacaoAluno,
123   localInscricao: CAMPO_TEXTO  label = "Local de inscrição"  valores = Curso.
          localInscricao,
124   gratuito: CAMPO_RADIO  label = "Gratuito para o público"  valores = Curso.gratuito
            preenchimento forçado,
125   * cargaHoraria: CAMPO_TEXTO  label = "Carga horária"  valores = Curso.cargaHoraria
126         [MSG_ERRO_CAMPO_ABAIXO_LIMITE ="Carga horária não pode ser inferior a"],
127   * publicoEstimado: CAMPO_TEXTO  label = "Público estimado/vagas"  valores = Curso.
          publicoEstimado
128         [MSG_ERRO_CAMPO_OBRIGATORIO = "O campo Público estimado é obrigatório e não
```

```
            foi preenchido!"],
129
130   Salvar: CAMPO_BOTAO label = "Salvar e Avançar",
131   Submeter: CAMPO_BOTAO label = "Submeter à Aprovação"
132
133   Mensagens {
134     SubmissaoFeitaComSucesso "Ação de Extensão submetida com sucesso!"
135   }
136 }
137
138
139
140
141 Tela TelaAbrangencia {
142   * identificacao:CAMPO_TEXTO label = "Identificação do Local" valores = Abrangencia
        .identificacao
143         [MSG_ERRO_CAMPO_OBRIGATORIO = "O campo Nome do local é obrigatório e não foi
              preenchido!"],
144   * estado:CAMPO_LISTA label = "Estado" valores = Abrangencia.estado,
145   * municipio:CAMPO_LISTA label = "Município" valores = Abrangencia.municipio
146         [MSG_ERRO_CAMPO_OBRIGATORIO = "O campo Municipio é obrigatório e não foi
              preenchido!"],
147   cep:CAMPO_TEXTO label = "CEP" valores = Abrangencia.cep ,
148   detalhes:CAMPO_TEXTO label = "Endereço" valores = Abrangencia.detalhes,
149   botaoIncluir:CAMPO_BOTAO label = "Incluir"
150
151   Mensagens {
152     AbrangenciaSalva "Abrangência inserida com sucesso!"
153   }
154 }
155
156
157
158 Tela TelaParceria {
159   * nome: CAMPO_AREA_DE_TEXTO label = "Nome do parceiro" valores = Parceria.nome,
160   * caracterizacao: CAMPO_LISTA label = "Caracterização" valores = Parceria.
        caracterizacao,
161   * formaDeParceria: CAMPO_CHECKBOX label = "Forma de parceria" valores = Parceria.
        formaDeParceria
162         [MSG_ERRO_CAMPO_OBRIGATORIO = "O campo Tipo de parceria é obrigatório e não
              foi preenchido!"],
163   botaoIncluir:CAMPO_BOTAO label = "Incluir"
164
165   Mensagens {
166     ParceriaSalva "Parceiro inserido com sucesso!"
167   }
168 }
169
170
171 Tela TelaEquipe {
172   * categoria: CAMPO_LISTA label ="Categoria" valores =Participacao.categoria,
173   * nome: CAMPO_AREA_DE_TEXTO label = "Nome",
174   * listaDeNomes: CAMPO_LISTA xpath = "//*[@id=\"listaNomeMembro\"]/tr[*]/td/a"
        valores = Participacao.listaDeNomes,
175   * formaparticipacao: CAMPO_LISTA label ="Forma de Participação" valores =
        Participacao.formaparticipacao,
176   * datainicio: CAMPO_AREA_DE_TEXTO label ="Data início" valores =Participacao.
```

```
      datainicio,
177   * dataTermino:CAMPO_AREA_DE_TEXTO label ="Data término" valores =Participacao.
      dataTermino,
178   botaoincluir:CAMPO_BOTAO id = "botaoMembro"
179
180   Mensagens {
181     MSG_PARTICIPACAO_INCLUIDA "Membro inserido com sucesso",
182   }
183 }
```

Listagem 72 – Dicionário de Dados

```
1  Classe AcaoExtensao {
2
3    titulo: expressaoJS("'titulo '+dataHoraAtual()"),
4    dataInicio:  Data
5                 min "01/01/1950" max "",
6    previsaoTermino:  Data
7                 min campo dataInicio  max "",
8    anoPrimeiraApresentacao:   Numerico min "1950" max "",
9    unidade:  valores da lista (1..n),
10   departamento:  valores da lista (1..n),
11   areaTematicaPrincipal: valores da lista (1..n),
12   areaTematicaAfim:  valores da lista (1..n),
13   linhaExtensao: valores da lista (1..n),
14   grandeArea:  valores da lista (1..n),
15   bolsaPBEXT:  Numerico min "0" max "9999999" ,
16   bolsaFUMP:   Numerico min "0" max "9999999" ,
17   bolsaInstitucional:  Numerico min "0" max "9999999" ,
18   bolsaPROEXT:  Numerico min "0" max "9999999" ,
19   bolsaFAPEMIG:  Numerico min "0" max "9999999" ,
20   bolsaExterna:  Numerico min "0" max "9999999" ,
21   palavrasChave: Texto min "1" max "200",
22   apresentacaoJustificativa:  Texto min "1" max "4000",
23   objetivosGerais: Texto min "1" max "500",
24   objetivosEspecificos:  Texto min "1" max "1500",
25   metodologia:  Texto min "1" max "3000",
26   formasAvaliacao: Texto min "1" max "1000",
27   site:  Texto min "1" max "100" ,
28   origemPublicoAlvo:  valores da lista (1..n),
29   caracterizacaoPublicoAlvo:   Texto min "1" max "500" ,
30   vinculoEdital: valores ("true","false"),
31   vinculoPoliticaPublica:   valores ("true","false"),
32   planoAtividades:  Texto min "1" max "1500" ,
33   planoAcompanhamento:  Texto min "1" max "700" ,
34   processoAvaliacao:   Texto min "1" max "700" ,
35   informacoesAdicionais:   Texto min "1" max "1000"
36 }
37
38
39
40
41 Classe Curso ClasseBase AcaoExtensao {
42
43   caracterizacao:  [aperfeicoamento: valores ("1"),
44                     geral: valores ("2","3","4")],
45   subCaracterizacao: valores da lista (1..n),
```

```
46   programa:  valores da lista (1..n),
47   projeto: valores da lista (1..n),
48   estruturaCurricular:  Texto min "1" max "7000",
49   infraEstruturaFisica:  Texto min "1" max "1000",
50   formaAvaliacaoAluno: Texto min "1" max "1000",
51   localInscricao:  Texto min "1" max "250",
52   gratuito:  valores ("true","false"),
53   cargaHoraria: [aperfeicoamento: Numerico min "180" max "99999",
54                  geral: Numerico min "8" max "99999"],
55   publicoEstimado:  Numerico min "0" max "999999999"
56 }
57
58
59 Classe Abrangencia {
60   identificacao: Texto min "1" max "250",
61   estado: valores da lista (1..n),
62   municipio: valores da lista (1..n),
63   cep: Texto min "1" max "10",
64   detalhes: Texto min "1" max "250"
65 }
66
67
68 Classe Parceria {
69    nome: Texto min "1" max "250",
70    caracterizacao: valores da lista (1..n),
71    formaDeParceria: [ financiamento: valores ("3"), geral: valores
          ("1","2","4","5","6")]
72 }
73
74 Classe Participacao {
75   categoria: valores da lista (1..n),
76   parteValidaDeNome: valores ("silva","rocha","miranda"),
77   listaDeNomes: valores da lista (1..n),
78   formaparticipacao: valores da lista (1..n),
79   datainicio: Data min valor armazenado dataInicioCurso  max valor armazenado
         dataTerminoCurso,
80   dataTermino: Data min campo dataInicio max valor armazenado dataTerminoCurso
81 }
```

APÊNDICE D – ESPECIFICAÇÃO – SISTEMA DE EX-ALUNOS

Esta seção exibe a especificação completa dos casos de uso, interfaces de usuário e dicionário de dados utilizados nos experimentos apresentados no Capítulo 4.

As Listagens 73, 74, 75, 76 e 77 exibem os casos de uso participantes do experimento apresentado na Seção 4.4, em que se exercitou casos de uso do nível de Resumo para gerenciar subfuncionalidades CRUD de uma entidade.

Listagem 73 – Caso de uso `GerirCargos`

```
1  CasoDeUso GerirCargos (+) {
2    contexto "Gerenciar os cargos do sistema (Exercitar CRUDs)"
3    ator Secretaria
4    precondicao  ""
5  cenarioprincipal
6    1 "Usuário cria cargo"  {
7      Armazeno "CargoABC" como "nomecargo",
8      Executo CriarCargo com cargoASerIncluido = "{:nomecargo}"
9    },
10   2 "Usuário tenta criar cargo existente" {
11     Executo CriarCargo com cargoASerIncluido = "{:nomecargo}" com extensao = "5a"
12   },
13   3 "Usuário solicita exclusão de cargo, mas cancela pedido " {
14     Executo ExcluirCargo com cargoASerExcluido = "{:nomecargo}" com extensao = "4a"
15   },
16   4 "Usuário altera cargo para nova descrição " {
17     Armazeno "CargoDEF" como "novonomecargo",
18     Executo AlterarCargo com cargoASerAlterado = "{:nomecargo}", novoNomeCargo = "{:
           novonomecargo}"
19   },
20   5 "Usuário exclui cargo " {
21     Executo ExcluirCargo com cargoASerExcluido = "{:novonomecargo}"
22   }
23 }
```

Listagem 74 – Caso de uso `CriarCargo`

```
1  CasoDeUso  CriarCargo (!) {
2    contexto "Criar um novo cargo"
3    ator Secretaria
4    acionador {
5      Acesso link GerirCargos
6    }
7    precondicao ""
8  cenarioprincipal
9    1 "Usuário [clica] em [Criar Cargo]" {
10     Clico CriarCargo
11   },
12   2 "Sistema [exibe] tela [Criar Cargo]" {
13     Visualizo texto = "Detalhes do Cargo"
14   },
```

```
15    3 "Usuário [informa] o campo [Nome] com o nome do novo cargo" {
16      Informo Nome com "{:cargoASerIncluido}"
17    },
18    4 "Usuário [clica] em [Salvar]" {
19      Clico Salvar
20    },
21    5 "Sistema [exibe mensagem] de sucesso" {
22      Visualizo mensagem msgDadosSalvos
23    },
24    6 "Usuário verifica que cargo foi incluído" {
25      Executo PesquisarCargo com cargoPesquisado = "{:cargoASerIncluido}"
26    }
27  extensoes
28    5 a "Cargo ja cadastrado"
29      5a 1 "Sistema [exibe mensagem] de erro" {
30        Visualizo mensagem msgCargoJaCadastrado
31      },
32
33    5 b "Testes padrão"
34      5b 1 "Sistema [exibe mensagem] de erro" {
35        Visualizo uma mensagem de erro de testes padrão
36      }
37  }
```

Listagem 75 – Caso de uso `AlterarCargo`

```
1   CasoDeUso  AlterarCargo (!) {
2     contexto "Altera um cargo"
3     ator Secretaria
4     acionador {
5           Acesso link GerirCargos
6     }
7     precondicao ""
8       cenarioprincipal
9       1 "Usuário pesquisa o [cargoASerAlterado]" {
10        Executo PesquisarCargo com cargoPesquisado = "{:cargoASerAlterado}"
11      },
12      2 "Usuário clica em [Alterar], na linha do cargo pesquisado " {
13        Clico Alterar
14      },
15      3 "Sistema [exibe] tela [Alterar Cargo]" {
16        Visualizo texto = "Detalhes de Cargo"
17      },
18      4 "Usuário [informa] o campo [Nome] com o nome do novo cargo" {
19        Informo Nome com "{:novoNomeCargo}"
20      },
21      5 "Usuário [clica] em [Salvar]" {
22        Clico Salvar
23      },
24      6 "Sistema [exibe mensagem] de sucesso" {
25        Visualizo mensagem msgDadosSalvos
26      },
27      7 "Usuário verifica que cargo foi alterado" {
28        Executo PesquisarCargo com cargoPesquisado = "{:cargoASerAlterado }" com
              extensao = "3a",
29        Executo PesquisarCargo com cargoPesquisado = "{:novoNomeCargo}"
30      }
```

```
31    extensoes
32      6 a "Testes padrao"
33        6a 1 "Sistema [exibe mensagem] de erro" {
34           Visualizo uma mensagem de erro de testes padrão
35        }
36  }
```

Listagem 76 – Caso de uso `ExcluirCargo`

```
1   CasoDeUso  ExcluirCargo (!) {
2     contexto "Exclui um cargo"
3     ator Secretaria
4     acionador {
5           Acesso link GerirCargos
6     }
7     precondicao ""
8     cenarioprincipal
9       1 "Usuário pesquisa o [cargoASerExcluido]" {
10        Executo PesquisarCargo com cargoPesquisado = "{:cargoASerExcluido}"
11      },
12      2 "Usuário clica em [Excluir], na linha do cargo pesquisado " {
13        Clico Excluir
14      },
15      3 "Sistema [exibe mensagem] de alerta" {
16        Visualizo alerta com mensagem msgAlertaExclusaoCargo
17      },
18      4 "Usuário [confirma] exclusão" {
19        Respondo alerta com ok
20      },
21      5 "Usuário verifica que dado foi excluido" {
22        Executo PesquisarCargo com cargoPesquisado = "{:cargoASerExcluido}"  com
              extensao = "3a"
23      }
24    extensoes
25      4 a "Desistencia de exclusão"
26        4a 1 "Usuário [cancela] exclusão" {
27          Respondo alerta com cancel
28        }
29        4a 2 "Usuário verifica que dado não foi excluído" {
30          Executo PesquisarCargo com cargoPesquisado = "{:cargoASerExcluido}"
31        },
32        5 a "Impossibilidade de exclusão (cargo em uso)"
33          5a 1 "Sistema [exibe mensagem] de erro" {
34            Visualizo mensagem msgCargoNaoPodeSerExcluido
35         }
36         5a 2 "Usuário verifica que dado não foi excluido" {

37           Executo PesquisarCargo com cargoPesquisado = "{:cargoASerExcluido}"
38         }
39  }
```

Listagem 77 – Caso de uso `PesquisarCargo`

```
1   CasoDeUso  PesquisarCargo (-) {
2     contexto "Pesquisar cargo"
3     ator Secretaria
4     acionador {
```

```
5           Acesso link GerirCargos
6     }
7     precondicao ""
8     cenarioprincipal
9       1 "Usuário [informa] o campo [Nome] com o cargo a ser pesquisado" {
10         Informo Nome com "{:cargoPesquisado}"
11      },
12      2 "Usuário [clica] em [Buscar]" {
13        Clico Buscar
14      },
15      3 "Sistema exibe o cargo pesquisado na lista de resultados." {
16        Visualizo texto = "{:cargoPesquisado}"
17      }
18    extensoes
19      3 a "Cargo não cadastrado"
20        3a 1 "Sistema [exibe mensagem] de item não encontrado" {
21           Visualizo mensagem msgNenhumItemEncontrado
22      }
23  }
```

Já as Listagens 78, 79 e 80 exibem casos de uso utilizados como exemplos ao longo da Seção 4.2, em que se avaliou a expressividade da DSL LUCT por meio da análise de sua aplicação prática, utilizando casos de uso de sistemas de software reais.

Listagem 78 – Caso de uso `CriarOrganizacao`

```
1  CasoDeUso CriarOrganizacao (!){
2    execucaopropria S
3    ator Secretaria
4    acionador {
5      Acesso menu GerirOrganizacoes
6    }
7    cenarioprincipal
8      1 "Usuário inicia processo de Criar Organização" {
9          Clico CriarOrganizacao
10     },
11     2 "Usuário informa dados da organização" {
12         Informo[ Nome, Sigla, NaturezaOrganizacao, SetorAtuacao]
13     },
14     3 "Usuário submete solicitação de inclusão" {
15         Clico Salvar
16     },
17     4 "Sistema exibe mensagem de cadastramento com sucesso" {
18         Visualizo mensagem msgDadosSalvos
19     },
20     5 "Usuário cadastra endereços eletrônicos" {
21         Clico AbaEnderecosEletronicos,
22         Executo InserirEnderecoEletronico
23     },
24     6 "Usuário vincula pessoas à organização" {
25         Clico AbaPessoas,
26         Executo VincularPessoaAOrganizacao
27     },
28     7 "Usuário submete solicitação para salvar os dados incluídos nas abas" {
29         Clico Salvar
```

```
30      },
31      8 "Sistema exibe mensagem de cadastramento dos itens com sucesso" {
32          Visualizo mensagem msgDadosSalvos
33      }
34    extensoes
35      4 a erro-teste-padrao "Dado informado incorretamente"
36      4a 1 "Sistema exibe mensagem de erro" {
37          Visualizo uma mensagem de erro de testes padrão
38      }
39  }
```

Listagem 79 – Caso de uso `VincularPessoaAOrganização`

```
1   CasoDeUso VincularPessoaAOrganizacao(!) {
2     execucaopropria N
3     ator Secretaria
4     acionador {
5       Clico Adicionar
6     }
7     precondicao "Acesso à tela de vinculação" {
8       AcesseiTela VinculacaoPesssoa
9     }
10    cenarioprincipal
11      1 "Usuário seleciona pessoa" {
12        Clico Alterar,
13        Clico Adicionar,
14        Clico SelecionarPessoa,
15        Executo SelecionarPessoa
16      },
17      2 "Usuário informa dados da vinculação da pessoa à organização" {
18        Informo[cargo,dataInicio,dataTermino,atual]
19      },
20      3 "Usuário submete solicitação de inclusão" {
21        Clico Inserir
22      },
23      4 "Sistema exibe dados da pessoa vinculada em lista" {
24        Visualizo valor armazenado pessoaSelecionada na tabela Linha num 1 Coluna num
              1
25      }
26      extensoes
27        4 a erro-teste-padrao "Dado informado incorretamente"
28        4a 1 "Sistema exibe mensagem de erro" {
29          Visualizo uma mensagem de erro de testes padrão
30        }
31  }
```

Listagem 80 – Caso de uso `InserirEnderecoEletronico`

```
1   CasoDeUso InserirEnderecoEletronico(!) {
2     execucaopropria N
3     ator Secretaria
4     acionador {
5       Clico Adicionar
6     }
7     precondicao "Acesso à tela de endereços eletrônicos" {
8       AcesseiTela EnderecosEletronicos
9     }
```

```
10  cenarioprincipal
11    1 "Usuário informa endereço eletrônico de categoria (geral)" {
12      Informo categoria com EnderecoEletronico.categoria.geral,
13      Informo enderecoEletronico com EnderecoEletronico.enderecoEletronico.geral
14    },
15    2 "Usuário submete solicitação de inclusão" {
16      Clico Inserir
17    },
18    3 "Sistema exibe endereco eletrônico cadastrado em lista" {
19      Armazeno campo enderecoEletronico como enderecoEletronicoCadastrado,
20      Visualizo valor armazenado enderecoEletronicoCadastrado na tabela Linha num 1
            Coluna num 1
21    }
22  extensoes
23    1 a "Categoria é de email / Cadastro de primeiro email como preferencial"
          extensão de tela CategoriaEmail
24    1a 1 "Usuário informa endereço eletrônico de categoria e-mail" {
25      Informo categoria com EnderecoEletronico.categoria.email,
26      Informo enderecoEletronico com EnderecoEletronico.enderecoEletronico.email
27    }
28    1a 2 "Usuário informa e-mail como preferencial" {
29      Informo preferencial com EnderecoEletronico.preferencial.sim
30    }
31    1a 3 Executa passo 2,
32
33    1 b erro "Categoria é de email/ Cadastro de primeiro email como NÃO preferencial
          " extensão de tela CategoriaEmail
34    1b 1 "Usuário informa endereço eletrônico de categoria e-mail" {
35      Informo categoria com EnderecoEletronico.categoria.email,
36      Informo enderecoEletronico com EnderecoEletronico.enderecoEletronico.email
37    }
38    1b 2 "Usuário informa e-mail como NÃO preferencial" {
39      Informo preferencial com EnderecoEletronico.preferencial.nao
40    }
41    1b 3 "Usuário submete solicitação de inclusão" {
42      Clico Inserir
43    }
44    1b 4 "Sistema mensagem de erro pois deve haver um e-mail preferencial" {
45      Visualizo mensagem msgDeveHaverUmEmailPreferencial
46    },
47
48
49    1 c erro "Categoria é de email/ Cadastro de segundo email" extensão de tela
          CategoriaEmail
50    1c 1 "Usuário cadastra um primeiro email" {
51      Executo InserirEnderecoEletronico com extensao = "1a"
52      //Passos para retornar para o estado de nova inclusão:
53      Clico Adicionar
54    }
55    1c 2 "Usuário informa endereço eletrônico de categoria e-mail" {
56      Informo categoria com EnderecoEletronico.categoria.email,
57      Informo enderecoEletronico com EnderecoEletronico.enderecoEletronico.email
58    }
59    1c 3 "Usuário informa se e-mail é preferencial" {
60      Informo preferencial
61    }
62    1c 4 Executa passo 2,
```

```
63
64
65      3 a erro-teste-padrao "Dado informado incorretamente"
66      3a 1 "Sistema exibe mensagem de erro" {
67         Visualizo uma mensagem de erro de testes padrão
68      }
69  }
```

Por fim, a Listagem 81 exibe a descrição das interfaces de usuário relacionadas aos casos de uso apresentados neste apêndice, e a Listagem 82, o dicionário de dados correspondente.

Listagem 81 – Interfaces de Usuário

```
1   Tela GerirCargos {
2     * Nome:CAMPO_TEXTO label = "Nome" valores = GerirCargos.nome,
3     CriarCargo:CAMPO_BOTAO id = "formPesquisa:insercaoCargo",
4     Salvar:CAMPO_BOTAO label = "Salvar",
5     Buscar:CAMPO_BOTAO label = "Buscar",
6     Alterar:CAMPO_BOTAO label = "Alterar",
7     Excluir:CAMPO_BOTAO label = "Excluir"
8     Mensagens {
9           msgCargoJaCadastrado "Cargo já cadastrado",
10          msgAlertaExclusaoCargo "O comando \"Excluir\" foi acionado. O(a) Cargo \"{:
                cargoPesquisado}\" será excluído(a) do sistema. Gostaria realmente de
                continuar?"
11    }
12  }
13
14
15
16
17  Tela DadosCadastraisOrganizacao {
18    * Nome:CAMPO_TEXTO label = "Nome" valores = Organizacao.nome,
19    Sigla:CAMPO_TEXTO label = "Sigla" valores = Organizacao.sigla,
20    NaturezaOrganizacao:CAMPO_TEXTO label = "Natureza da Organização" valores =
          Organizacao.naturezaOrganizacao,
21    SetorAtuacao:CAMPO_TEXTO  label = "Setor de Atuação" valores = Organizacao.
          setorAtuacao,
22    AbaEnderecosEletronicos:CAMPO_LINK id = "formCadastro:
          abadetalhesOrganizacaoEnderecoEletronico_lbl" aciona tela EnderecosEletronicos
          ,
23    AbaPessoas:CAMPO_LINK id = "formCadastro:abadetalhesOrganizacaoPessoa_lbl" aciona
          tela VinculacaoPesssoa,
24    CriarOrganizacao:CAMPO_BOTAO id = "formPesquisa:insercaoOrganizacao",
25    Salvar:CAMPO_BOTAO label = "Salvar",
26    Buscar:CAMPO_BOTAO label = "Buscar",
27    Alterar:CAMPO_BOTAO label = "Alterar",
28    Excluir:CAMPO_BOTAO label = "Excluir"
29  }
30
31
32  Tela VinculacaoPesssoa {
33    cargo: CAMPO_LISTA label = "Cargo" valores = VinculacaoPesssoa.cargo,
34    dataInicio: CAMPO_TEXTO label = "Data de Início" valores = VinculacaoPesssoa.
          dataInicio,
```

```
35    dataTermino: CAMPO_TEXTO label = "Data de Término" valores = VinculacaoPesssoa.
          dataTermino,
36    * atual:CAMPO_RADIO label = "Atual" valores = VinculacaoPesssoa.atual,
37    SelecionarPessoa:CAMPO_BOTAO label = "Selecionar Pessoa ..." aciona tela
          SelecaoDePesssoa,
38    Inserir:CAMPO_BOTAO label = "Inserir",
39    Alterar:CAMPO_BOTAO label = "Alterar",
40    Adicionar:CAMPO_BOTAO xpath ="//*[@id=\"formCadastro:
          listagemdetalhesOrganizacaoPessoa:
          inserirDaTabelatabeladetalhesOrganizacaoPessoa\"]/span",
41  }
42
43
44  Tela EnderecosEletronicos {
45    * enderecoEletronico:CAMPO_TEXTO label ="Endereço eletrônico" valores =
          EnderecoEletronico.enderecoEletronico ,
46    * categoria:CAMPO_LISTA label ="Categoria" valores = EnderecoEletronico.categoria,
47    observacao:CAMPO_AREA_DE_TEXTO label ="Observação" valores = EnderecoEletronico.
          observacao,
48    Inserir:CAMPO_BOTAO label = "Inserir",
49    Alterar:CAMPO_BOTAO label = "Alterar",
50    Adicionar:CAMPO_BOTAO xpath ="//*[@id=\"formCadastro:
          listagemdetalhesOrganizacaoEnderecoEletronico:
          inserirDaTabelatabeladetalhesOrganizacaoEnderecoEletronico\"]/span"
51
52    Extensao CategoriaEmail{
53      * preferencial:CAMPO_RADIO label ="Preferencial" valores = EnderecoEletronico.
            preferencial
54    }
55    Mensagens {
56          msgEmailInvalido "Email inválido",
57          msgDeveHaverUmEmailPreferencial "Deve existir pelo menos um endereço
              eletrônico de categoria EMAIL cadastrado como preferencial."
58    }
59  }
```

Listagem 82 – Dicionário de Dados

```
1   Classe GerirCargos {
2     nome: Texto min "1" max "100"
3   }
4
5   Classe Organizacao {
6     nome: Texto min "1" max "100",
7     sigla:Texto min "1" max "10",
8     naturezaOrganizacao: valores da lista (1..n),
9     setorAtuacao: valores da lista (1..n)
10  }
11
12  Classe VinculacaoPesssoa {
13    cargo: valores da lista (1..n),
14    dataInicio: Data min "01/01/1900" max "",
15    dataTermino: Data min expressaoJS("somaDiasAData('{:CAMPO_dataInicio}',1)") max
          "",
16    atual: valores ("SIM","NAO")
17
18  }
```

```
19
20 Classe EnderecoEletronico {
21   enderecoEletronico: [geral: Texto min "1" max "200" ,
22         email: expressao "[a-z0-9]{1,30}[@][a-z0-9]{1,10}[.][a-z0-9]{2,3}"],
23   categoria: [geral: valores ("EnderecoEletronico_categoria2", "
       EnderecoEletronico_categoria3", "EnderecoEletronico_categoria4"),
24   email: valores ("EnderecoEletronico_categoria1") ],
25   preferencial: [sim: valores ("SIM"), nao: valores ("NAO")],
26   observacao: Texto min "0" max "200"
27 }
```

APÊNDICE E – LISTA DE CASOS DE TESTE – SISTEMA DE INFORMAÇÕES DE EXTENSÃO

Este apêndice exibe os casos de teste gerados durante os experimentos apresentados na Seção 4.3, para o Sistema de Informações de Extensão. Os experimentos envolveram especificar os casos de uso selecionados no ambiente EnvUCT a partir da documentação existente – adaptando a especificação quando necessário – até a obtenção de um conjunto de testes que funcionasse adequadamente. Em seguida, trabalhou-se na verificação da eficácia dos testes gerados, por meio de alterações na especificação e no SST (uso de mutantes).

Listagem 83 – Casos de teste gerados para o Sistema de Informações de Extensão

```
1  *** Nro de testes: 97 *** Lista de testes:
2  1 : CasodeUso InserirParceria [prioridade:0] [tags:] [TelaParceria]
3  1.1 : InserirParceria/Cenario_1_2_3
4  1.1.1 : sequenciamento: [LoginSiex/Cenario_1_2_3_4, InserirCurso/Cenario_1a_4_5]
5  1.1.1.1 : TP_VALIDOS_QUAISQUER[]
6  1.1.1.2 : TP_SO_OBRIGATORIOS[]
7  1.1.1.3 : TP_VALORES_VALIDOS_MIN[]
8  1.1.1.4 : TP_VALORES_VALIDOS_MAX[]
9  1.2 : InserirParceria/Cenario_1_2_3a
10 1.2.1 : sequenciamento: [LoginSiex/Cenario_1_2_3_4, InserirCurso/Cenario_1a_4_5]
11 1.2.1.1 : TP_VALOR_FALTANTE[nome]
12 1.2.1.2 : TP_VALOR_FALTANTE[caracterizacao]
13 1.2.1.3 : TP_VALOR_FALTANTE[formaDeParceria]
14 2 : CasodeUso InserirAbrangencia [prioridade:0] [tags:] [TelaAbrangencia]
15 2.1 : InserirAbrangencia/Cenario_1_2_3
16 2.1.1 : sequenciamento: [LoginSiex/Cenario_1_2_3_4, InserirCurso/Cenario_1a_4_5]
17 2.1.1.1 : TP_VALIDOS_QUAISQUER[]
18 2.1.1.2 : TP_SO_OBRIGATORIOS[]
19 2.1.1.3 : TP_VALORES_VALIDOS_MIN[]
20 2.1.1.4 : TP_VALORES_VALIDOS_MAX[]
21 2.2 : InserirAbrangencia/Cenario_1_2_3a
22 2.2.1 : sequenciamento: [LoginSiex/Cenario_1_2_3_4, InserirCurso/Cenario_1a_4_5]
23 2.2.1.1 : TP_VALOR_FALTANTE[identificacao]
24 2.2.1.2 : TP_VALOR_FALTANTE[estado]
25 2.2.1.3 : TP_VALOR_FALTANTE[municipio]
26 3 : CasodeUso AlterarCursoESubmeterAAprovacao [prioridade:0] [tags:] [
       TelaInserirCurso,TelaAbrangencia, TelaParceria, TelaAlterarCurso]
27 3.1 : AlterarCursoESubmeterAAprovacao/Cenario_1_2_3_4_5a
28 3.1.1 : sequenciamento: [LoginSiex/Cenario_1_2_3_4, InserirCurso/Cenario_1a_4_5,
       InserirAbrangencia/Cenario_1_2_3, InserirParceria/Cenario_1_2_3]
29 3.1.1.1 : TP_VALOR_INVALIDO_MIN_MENOS_UM[dataInicio]
30 3.1.1.2 : TP_VALOR_INVALIDO_MIN_MENOS_UM[previsaoTermino]
31 3.1.1.3 : TP_VALOR_INVALIDO_MIN_MENOS_UM[anoPrimeiraApresentacao]
32 3.1.1.4 : TP_VALOR_INVALIDO_MIN_MENOS_UM[cargaHoraria
33 3.1.1.5 : TP_VALOR_FALTANTE[titulo]
34 3.1.1.6 : TP_VALOR_FALTANTE[dataInicio]
35 3.1.1.7 : TP_VALOR_FALTANTE[previsaoTermino]
36 3.1.1.8 : TP_VALOR_FALTANTE[unidade]
37 3.1.1.9 : TP_VALOR_FALTANTE[departamento]
```

```
3.1.1.10 : TP_VALOR_FALTANTE[areaTematicaPrincipal]
3.1.1.11 : TP_VALOR_FALTANTE[linhaExtensao]
3.1.1.12 : TP_VALOR_FALTANTE[grandeArea]
3.1.1.13 : TP_VALOR_FALTANTE[bolsaFAPEMIG]
3.1.1.14 : TP_VALOR_FALTANTE[palavrasChave]
3.1.1.15 : TP_VALOR_FALTANTE[apresentacaoJustificativa]
3.1.1.16 : TP_VALOR_FALTANTE[objetivosGerais]
3.1.1.17 : TP_VALOR_FALTANTE[metodologia]
3.1.1.18 : TP_VALOR_FALTANTE[formasAvaliacao]
3.1.1.19 : TP_VALOR_FALTANTE[origemPublicoAlvo]
3.1.1.20 : TP_VALOR_FALTANTE[caracterizacaoPublicoAlvo]
3.1.1.21 : TP_VALOR_FALTANTE[caracterizacao]
3.1.1.22 : TP_VALOR_FALTANTE[cargaHoraria]
3.1.1.23 : TP_VALOR_FALTANTE[subCaracterizacao]
3.1.1.24 : TP_VALOR_FALTANTE[estruturaCurricular]
3.1.1.25 : TP_VALOR_FALTANTE[infraEstruturaFisica]
3.1.1.26 : TP_VALOR_FALTANTE[formaAvaliacaoAluno]
3.1.1.27 : TP_VALOR_FALTANTE[publicoEstimado]
3.2 : AlterarCursoESubmeterAAprovacao/Cenario_1_2_3_4_5
3.2.1 : sequenciamento: [LoginSiex/Cenario_1_2_3_4, InserirCurso/Cenario_1a_4_5,
    InserirAbrangencia/Cenario_1_2_3, InserirParceria/Cenario_1_2_3]
3.2.1.1 : TP_VALIDOS_QUAISQUER[]
3.2.1.2 : TP_SO_OBRIGATORIOS[]
3.2.1.3 : TP_VALORES_VALIDOS_MIN[]
3.2.1.4 : TP_VALORES_VALIDOS_MAX[]
3.3 : AlterarCursoESubmeterAAprovacao/Cenario_1_2a_3_4_5
3.3.1 : sequenciamento: [LoginSiex/Cenario_1_2_3_4, InserirCurso/Cenario_1a_4_5,
    InserirAbrangencia/Cenario_1_2_3, InserirParceria/Cenario_1_2_3]
3.3.1.1 : TP_VALIDOS_QUAISQUER[]
3.3.1.2 : TP_SO_OBRIGATORIOS[]
3.3.1.3 : TP_VALORES_VALIDOS_MIN[]
3.3.1.4 : TP_VALORES_VALIDOS_MAX[]
3.4 : AlterarCursoESubmeterAAprovacao/Cenario_1_2a_3_4_5a
3.4.1 : sequenciamento: [LoginSiex/Cenario_1_2_3_4, InserirCurso/Cenario_1a_4_5,
    InserirAbrangencia/Cenario_1_2_3, InserirParceria/Cenario_1_2_3]
3.4.1.1 : TP_VALOR_INVALIDO_MIN_MENOS_UM[dataInicio]
3.4.1.2 : TP_VALOR_INVALIDO_MIN_MENOS_UM[previsaoTermino]
3.4.1.3 : TP_VALOR_INVALIDO_MIN_MENOS_UM[anoPrimeiraApresentacao]
3.4.1.4 : TP_VALOR_INVALIDO_MIN_MENOS_UM[cargaHoraria]
3.4.1.5 : TP_VALOR_FALTANTE[titulo]
3.4.1.6 : TP_VALOR_FALTANTE[dataInicio]
3.4.1.7 : TP_VALOR_FALTANTE[previsaoTermino]
3.4.1.8 : TP_VALOR_FALTANTE[unidade]
3.4.1.9 : TP_VALOR_FALTANTE[departamento]
3.4.1.10 : TP_VALOR_FALTANTE[areaTematicaPrincipal]
3.4.1.11 : TP_VALOR_FALTANTE[linhaExtensao]
3.4.1.12 : TP_VALOR_FALTANTE[grandeArea]
3.4.1.13 : TP_VALOR_FALTANTE[bolsaFAPEMIG]
3.4.1.14 : TP_VALOR_FALTANTE[palavrasChave]
3.4.1.15 : TP_VALOR_FALTANTE[apresentacaoJustificativa]
3.4.1.16 : TP_VALOR_FALTANTE[objetivosGerais]
3.4.1.17 : TP_VALOR_FALTANTE[metodologia]
3.4.1.18 : TP_VALOR_FALTANTE[formasAvaliacao]
3.4.1.19 : TP_VALOR_FALTANTE[origemPublicoAlvo]
3.4.1.20 : TP_VALOR_FALTANTE[caracterizacaoPublicoAlvo]
3.4.1.21 : TP_VALOR_FALTANTE[caracterizacao]
3.4.1.22 : TP_VALOR_FALTANTE[cargaHoraria]
```

```
3.4.1.23 : TP_VALOR_FALTANTE[subCaracterizacao]
3.4.1.24 : TP_VALOR_FALTANTE[estruturaCurricular]
3.4.1.25 : TP_VALOR_FALTANTE[infraEstruturaFisica]
3.4.1.26 : TP_VALOR_FALTANTE[formaAvaliacaoAluno]
3.4.1.27 : TP_VALOR_FALTANTE[publicoEstimado]
4 : CasodeUso InserirCurso [prioridade:0] [tags:] [TelaInserirCurso]
4.1 : InserirCurso/Cenario_1_2_3_4_5a
4.1.1 : sequenciamento: [LoginSiex/Cenario_1_2_3_4]
4.1.1.1 : TP_VALOR_INVALIDO_MIN_MENOS_UM[dataInicio]
4.1.1.2 : TP_VALOR_INVALIDO_MIN_MENOS_UM[previsaoTermino]
4.1.1.3 : TP_VALOR_INVALIDO_MIN_MENOS_UM[anoPrimeiraApresentacao]
4.1.1.4 : TP_VALOR_INVALIDO_MIN_MENOS_UM[cargaHoraria]
4.1.1.5 : TP_VALOR_FALTANTE[titulo]
4.2 : InserirCurso/Cenario_1a_4_5
4.2.1 : sequenciamento: [LoginSiex/Cenario_1_2_3_4]
4.2.1.1 : TP_VALIDOS_QUAISQUER[]
4.2.1.2 : TP_SO_OBRIGATORIOS[]
4.3 : InserirCurso/Cenario_1a_4_5a
4.3.1 : sequenciamento: [LoginSiex/Cenario_1_2_3_4]
4.3.1.1 : TP_VALOR_FALTANTE[titulo]
4.4 : InserirCurso/Cenario_1_2_3_4_5
4.4.1 : sequenciamento: [LoginSiex/Cenario_1_2_3_4]
4.4.1.1 : TP_VALIDOS_QUAISQUER[]
4.4.1.2 : TP_SO_OBRIGATORIOS[]
4.4.1.3 : TP_VALORES_VALIDOS_MIN[]
4.4.1.4 : TP_VALORES_VALIDOS_MAX[]
4.5 : InserirCurso/Cenario_1_2a_3_4_5
4.5.1 : sequenciamento: [LoginSiex/Cenario_1_2_3_4]
4.5.1.1 : TP_VALIDOS_QUAISQUER[]
4.5.1.2 : TP_SO_OBRIGATORIOS[]
4.5.1.3 : TP_VALORES_VALIDOS_MIN[]
4.5.1.4 : TP_VALORES_VALIDOS_MAX[]
4.6 : InserirCurso/Cenario_1_2a_3_4_5a
4.6.1 : sequenciamento: [LoginSiex/Cenario_1_2_3_4]
4.6.1.1 : TP_VALOR_INVALIDO_MIN_MENOS_UM[dataInicio]
4.6.1.2 : TP_VALOR_INVALIDO_MIN_MENOS_UM[previsaoTermino]
4.6.1.3 : TP_VALOR_INVALIDO_MIN_MENOS_UM[anoPrimeiraApresentacao]
4.6.1.4 : TP_VALOR_INVALIDO_MIN_MENOS_UM[cargaHoraria]
4.6.1.5 : TP_VALOR_FALTANTE[titulo]
```