UNIVERSIDADE FEDERAL DE CAMPINA GRANDE
CENTRO DE ENGENHARIA ELÉTRICA E INFORMÁTICA
COORDENAÇÃO DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

DISSERTAÇÃO DE MESTRADO

UMA ESTRATÉGIA PARA A ESCRITA DE CENÁRIOS DE CASO DE USO VOLTADOS À GERAÇÃO AUTOMÁTICA DE CASOS DE TESTE

MAKELLI ARAÚJO JUCÁ

CAMPINA GRANDE-PB
JUNHO-2009

Universidade Federal de Campina Grande
Centro de Engenharia Elétrica e Informática
Coordenação de Pós-Graduação em Ciência da Computação

Uma Estratégia para a Escrita de Cenários de Caso de Uso Voltados à Geração Automática de Casos de Teste

Makelli Araújo Jucá

Dissertação submetida à Coordenação do Curso de Pós-Graduação em Ciência da Computação da Universidade Federal de Campina Grande - Campus I como parte dos requisitos necessários para obtenção do grau de Mestre em Ciência da Computação.

Área de Concentração: Ciência da Computação

Linha de Pesquisa: Engenharia de Software

Patrícia Duarte de Lima Machado

(Orientadora)

Campina Grande, Paraíba, Brasil

©Makelli Araújo Jucá, 01/06/2009

J919e
2009 Jucá, Makelli Araújo

Uma estratégia para a escrita de cenários de caso de uso voltados à geração automática de casos de teste / Makelli Araújo Jucá. - Campina Grande, 2009.

97p. : il. Color.

Dissertação (Mestrado em Informática) - Universidade Federal de Campina Grande, Centro de Engenharia Elétrica e Informática.

Referências.

Orientadores : Prof[a]. Dr[a]. Patrícia Duarte Lima Machado.

1. Software - Modelo. 2. Modelo - Teste. 3. Uso - Caso. 4. Informática. I. Título.vspace1pt

CDU 004.4(043)

"UMA ESTRATÉGIA PARA A ESCRITA DE MODELOS VOLTADOS À GERAÇÃO AUTOMÁTICA DE CASOS DE TESTE"

MAKELLI ARAÚJO JUCÁ

DISSERTAÇÃO APROVADA EM 01.07.2009

Patrícia Duarte de Lima Machado

PROFª PATRÍCIA DUARTE DE LIMA MACHADO , Ph.D
Orientadora

Rohit Gheyi

PROF. ROHIT GREYI, Dr.
Examinador

Flávia de Almeida Barros

PROFª FLÁVIA DE ALMEIDA BARROS, Ph.D
Examinadora

CAMPINA GRANDE – PB

Resumo

A utilidade do modelo produzido abrange uma diversidade de atividades no ciclo de vida do produto, variando de aprimoramento da qualidade da especificação à geração de teste. Teste baseado em modelos é uma das técnicas de teste de *software* que se beneficia da utilização do modelo de *software*. A idéia é comparar comportamentos de entrada e saída do modelo do sistema com o sistema propriamente dito. Neste processo, o modelo atua como o plano de teste e a manutenção dos casos dos testes devido à mudanças de requisitos se reflete em alterações no modelo. O processo de teste se constitui, portanto, de uma atividade que ocorre mais cedo dentro do processo de desenvolvimento. Além do mais, a automação ocasiona um maior número de casos de teste. Embora tenha características promissoras, a adoção industrial tem sido bastante limitada. O modelo elaborado com propósito de testar figura como elemento principal dos obstáculos encontrados pela introdução da nova técnica. Neste sentido, foram levantados problemas específicos associados ao contexto do ambiente Motorola. Este trabalho tem por objetivo geral propor uma abordagem de construção do modelo de entrada utilizado pela técnica MBT. O problema fundamental é facilitar o processo de escrita de casos de uso a partir de documentos de requisitos e ao mesmo tempo tornar o processo mais apropriado e completo para a geração de casos de teste. Para isso, elaboramos uma metodologia de escrita de cenários de caso de uso utilizando como formalismo diagramas de seqüência da versão 2 do *framework* UML onde propomos um procedimento para utilização de padrões de casos de teste dentro da atividade de elaboração do modelo de teste, uma notação para relacionar casos de uso e uma abordagem para descrever casos de uso em dois níveis de abstração.

Abstract

The usefulness of the model produced on software development covers a range of activities in the product life cycle, ranging from improving the quality of the specification to the test generation . Model Based Testing is one of the software test technique that benefits from the use of software model. The MBT idea is to compare behavior of input and output of the system model with the real system. In this process, the model serves as the testing plan and the test cases maintenance due to requirements changes are reflected in model changes. Therefore, the testing process is an activity that occurs early in the development process. Moreover, automation leads to a greater number of test cases. Although promising characteristics, the industry adoption has been very limited. The model developed with test purpose is included as part of the main obstacles encountered by the introduction of new technology. Accordingly, we raised specific problems with regard to environmental Motorola. This work aims to general propose an approach to constructing the model used by the MBT technical. The fundamental problem is to facilitate the process of writing use cases from requirements documents, while making the process more appropriate and complete for the test cases generation. For this, we develop a strategy for writing use case scenarios using the formalism of sequence diagrams of UML version 2 where we propose a procedure for use of test cases patterns in the activity of producing the test model, a notation to describe use cases relationship and an approach to describe use cases in two levels of abstraction.

Agradecimentos

Reconheço que este não é o trabalho de uma pessoa sozinha.

Agradeço a Deus por me conceder esta realização.

Agradeço a professora Patrícia Duarte de Lima Machado pela oportunidade trabalhar ao lado de uma pessoa de conhecimento imensurável e de caráter exemplo.

Aos meus queridos amigos Emanuela e Neto, pela paciência diária e pelo imenso apoio.

A Ana Lúcia Guimarães, pelo seu trabalho paciente e honrado

A todos da família CINBTCRD.

Aos meus irmãos e minha mãe, presentes em todos os momentos.

A Elias, incondicionalmente do meu lado.

Em especial, agradeço ao meu pai que nunca deixou de estar presente durante todos os dias da minha vida.

Conteúdo

1 Introdução **1**

1.1 Objetivo 5

1.2 Contribuições 6

1.3 Contexto 7

1.4 Estrutura dos capítulos 8

2 Fundamentação teórica **10**

2.1 Processos de Automação de Teste de Software 10

2.1.1 Processo de teste manual 11

2.1.2 Captura e repetição 12

2.1.3 Processo de teste Baseado em *Script* 13

2.1.4 Processo de teste baseado em palavras-chave 15

2.1.5 Teste baseado em modelos 15

2.2 Diagramas de seqüencia 18

2.2.1 Elementos 19

2.2.2 Fragmentos combinados 20

2.2.3 Modularização de diagramas de seqüencia 22

2.3 *Labeled Transition Systems* 22

2.3.1 LTS Anotado 23

2.4 Caso de Uso 24

2.4.1 Relação entre casos de uso 27

2.5 Padrões de teste 30

2.6 Processo de teste em aplicações de celulares 31

2.7 Considerações Finais 35

3 Padrões de especificação de caso de teste **38**

3.1 Introdução 38

3.2 Estratégia de escrita de modelo de teste 39

3.2.1 Utilização 42

3.2.2 Adição de padrões 43

3.2.3 Atualização 45

3.3 Exemplo 46

3.4 Conclusão 46

4 Procedimento de Testes Baseados em Modelos **52**

4.1 Geração e Seleção de Casos de Teste com LTS-BT 52

4.2 Relações entre casos de uso 57

4.2.1 Relação de extensão 57

4.2.2 Relação de inclusão 61

4.3 Casos de uso em dois níveis de detalhes 61

4.4 Conclusão 62

5 Estudo de Caso **67**

5.1 Métricas e Procedimento 67

5.1.1 MÉTRICAS 67

5.1.2 PROCEDIMENTOS 71

5.2 Ambiente e execução 73

5.3 Análise dos resultados 74

5.4 Conclusão 76

6 Conclusão **78**

6.1 Trabalhos relacionados 79

6.2 Trabalhos futuros 81

Lista de Figuras

1.1 Visão geral do processo de Teste baseado em Modelos no CInBTCRD . . . 8

2.1 Instância de um teste manual . . . 12
2.2 Instância de um *script* de teste . . . 14
2.3 Exemplo de diagrama de seqüencia . . . 22
2.4 Exemplo de LTS . . . 24
2.5 *Exemplo de template de caso de uso* . . . 26
2.6 Exemplo de relação *include* . . . 28
2.7 Exemplo de relação *extend* . . . 29
2.8 Exemplo de relação *extend* com *extension point* . . . 29
2.9 Exemplo de relação generalização . . . 30
2.10 *Features* . . . 32
2.11 *Exemplo de casos de teste de* feature *com redundância sintática* . . . 34
2.12 *Exemplo de casos de teste de* feature *com redundância semântica* . . . 35
2.13 Exemplo de LTS Anotado . . . 36

3.1 Visão Geral da abordagem de estratégia de escrita de modelo de testes com utilização de padrões. . . . 40
3.2 Integração do *Lucene* com as aplicações. . . . 42
3.3 Exemplo de utilização . . . 43
3.4 Adição de novos padrões . . . 44
3.5 Processamento da Entrada . . . 45
3.6 Padrão *Criar mensagem simples de texto* . . . 47
3.7 Padrão *Adicionar um novo contato para a Agenda* . . . 48
3.8 Padrão instanciado *Criar mensagem simples de texto* . . . 49

3.9 Exemplo 50
3.10 Padrão instanciado *Adicionar um novo contato para a Agenda* 51

4.1 *Arquitetura da LTS-BT* 53
4.2 Ligar para o número de telefone embutido de uma mensagem. 55
4.3 LTS resultante da Figura 4.2 56
4.4 Relação entre casos de uso (pontos de extensão). 58
4.5 Caso de Uso: Exibir Caixa de Entrada. 59
4.6 Caso de Uso: Inserir PIN. 60
4.7 LTS resultante das Figuras 4.5 e 4.6 63
4.8 Casos de testes resultantes da Figura 4.7 64
4.9 Exemplo 65
4.10 LTS resultante da Figura 4.9 sem detalhes 66

5.1 Goal 69
5.2 *GQM Abstraction Sheet* 70
5.3 Modelo GQM 72

Lista de Tabelas

5.1 *Perfis* 73
5.2 *Dados coletados do designers* 74
5.3 *Dados coletados do testador1 (com padrões)* 74
5.4 *Dados coletados do testador2 (sem padrões)* 75

Capítulo 1

Introdução

Um problema presente em toda organização de desenvolvimento de *software* é como reduzir orçamento, tempo e recurso sem que seja perdida a qualidade do produto final. É de senso comum que processo de teste de *software* aliado ao processo de desenvolvimento de *software* contribui com a garantia de melhoria da qualidade do produto final. Neste sentido, o desafio se encontra em práticas efetivas de teste de forma que seja de execução prática. A automação do processo de testes se torna, portanto, não apenas desejável, mas uma necessidade. O uso de abstração, cuja idéia contribui para simplificação, é uma técnica também promissora neste cenário.

Modelagem consiste na simplificação do comportamento e/ou estrutura do *software* através de utilização de abstração. Representado por uma variedade de formalismos, o modelo produzido abrange uma diversidade de utilidade em diferentes atividades dentro do ciclo de desenvolvimento do produto. Testes Baseados em Modelos ou MBT (*Model-Based Testing*) [23] é uma técnica de teste de *software* onde a geração dos testes ocorre a partir do modelo de *software*. A idéia de MBT é comparar comportamentos de entrada e saída do modelo do sistema com o sistema propriamente dito. Em outras palavras, é o uso do modelo abstrato para testar uma implementação concreta.

O que diferencia MBT, quanto à automação, das outras práticas de teste é o nível atingido. A automação se antecipa à execução, chegando ao *design* dos casos de teste. Neste processo, o modelo atua como o plano de teste e a manutenção dos casos de teste devido à mudanças de requisitos se reflete em alterações no modelo. O processo de teste se constitui, portanto, de uma atividade que ocorre mais cedo dentro do processo de desenvolvimento. Além do

mais, a automação ocasiona um maior número de casos de teste. A antecipação do processo de teste no ciclo de vida do sistema, bem como o nível de automação atingido, colocam como promessa de MBT a diminuição de custos associados a teste, com maior eficiência nas métricas que definem a qualidade do produto.

Expectativas promissoras e relatos comprovados de sucesso [8; 24; 47] não consolidaram a técnica no campo da indústria. A adoção industrial tem sido bastante limitada e os desafios das novas investigações são relatar e corrigir os problemas associados a esta restrição. O modelo elaborado com propósito de testar figura como elemento principal dos obstáculos encontrados pela introdução da nova técnica [9]. A tentativa de incorporar o processo de Testes Baseados em Modelos dentro do processo de desenvolvimento impõe à equipe de teste a elaboração de um modelo de teste dentro de um nível mais abstrato que a implementação real, sem perda da habilidade de gerar testes executáveis. Assim, existe a necessidade de tradução de níveis tanto no caminho no qual as entradas são extraídas do modelo e executadas no *software*, como na comparação entre as saídas produzidas pelo sistema e as saídas especuladas no modelo. Outro empecilho é encontrar um modo eficiente de combinar diferentes tipos de modelos [9], dado que um único modelo seja insuficiente para cobrir os aspectos que se deseja observar/testar. Resumindo, a especificação do modelo pode implicar num investimento bastante significativo dentro do processo.

O aspecto abstrato do modelo de teste - onde detalhes são encapsulados pela sintaxe das linguagens de alto nível - pode receber outra conotação significando a premeditada omissão de detalhes. Omissões no modelo implicam na não cobertura dos testes para estas partes omitidas [62]. Ou seja, ainda que tenha um caráter fortemente automático, MBT não garante cobertura completa das informações pretendidas como requisitos. Em outras palavras, geração automática de grandes quantidades de casos de teste não é suficiente para garantir a qualidade dos testes. É necessário representar os requisitos da maneira mais completa possível (e de forma testável) dentro do modelo de teste.

Em suma, metodologias baseadas em modelos para geração automática de artefatos têm como fator de sucesso de sua abordagem a qualidade do modelo, qualidade esta, por sua vez, que limita aos recursos existentes para a construção do modelo. No exemplo específico de testes baseados em modelos, os recursos para construção do modelo são os requisitos do sistema. O mal entendimento dos requisitos do sistema pode ocasionar a concretização

de um produto diferente do solicitado pelo cliente. Medidas mais rigorosas de processo de *software* podem contornar este problema. No entanto, os requisitos podem não estar descritos completamente ou estejar implicitamente encobertos por vícios de *expert* no domínio da aplicação.

O modelo de teste que utilizamos são casos de uso. Em nosso contexto, propomos o uso de cenários, não somente para captura e especificação de requisitos, como é feito em métodos de desenvolvimento orientado a objetos, mas especificamente para o desenvolvimento de casos de teste de sistema. Ou seja, a abordagem específica de testes baseados em modelo na qual nos enquadramos é a de geração casos de teste a partir de modelos do comportamento do sistema com informações sobre resultados esperados. Cenários em linguagem natural são utilizados para modelar os requisitos e dão uma visão centrada no usuário das funcionalidades e comportamento do sistema. Cenários são amplamente utilizados na fase de especificação de requisitos funcionais e, portanto, servem de base para as primeiras atividades relacionadas à teste. No entanto, este mapeamento pode não ser tão simples, pois, para sistemas maiores, os casos de uso se encontram em alto nível, enquanto a construção de casos de teste pode exigir passos mais detalhados.

Pesquisas acadêmicas na área de MBT têm sido conduzidas por volta de duas décadas. Mas, apesar de envolver muitas tarefas, o problema mais focado na literatura é a parte de seleção de casos de testes [12]. Trabalhos com abordagens para o modelo de entrada estão em número mais reduzido. Uma nova tendência é conduzir trabalhos com MBT num âmbito mais restrito ao domínio das aplicações [34; 56; 37; 29].

Baseados em estudos anteriores, tais como o apresentado por Nascimento e Machado [39], bem como nossa experiência inicial, observamos alguns obstáculos técnicos e não-técnicos para a implantação de MBT em larga escala. Acreditamos que as possibilidades de sucesso na implantação MBT irão melhorar através de soluções específicas de um domínio que foram adaptadas a um contexto específico. No nosso caso, o domínio são aplicações para celulares e o contexto são testes de *feature*[1].

A nossa idéia é propor uma estratégia de escrita de cenários de caso de uso utilizando como formalismo diagramas de seqüência da versão 2 do *framework* UML. Propomos, neste

[1]Uma *feature* é uma estrutura funcional com propriedades visíveis desenvolvida sem o conhecimento de outras *features* [60].

trabalho: um procedimento para utilização de padrões de casos de teste dentro da atividade de elaboração do modelo de teste (Capítulo 3), uma notação para relacionar casos de uso (Capítulo 4) e uma abordagem para descrever casos de uso em dois níveis de abstração para atender a dois tipos de perfis (Capítulo 4): testador experiente na aplicação em teste, onde a adição de detalhes nos casos de teste se torna desnecessária, e o testador inexperiente.

Dado que teste possui um processo específico em paralelo com o processo de desenvolvimento, é desejável que sejam identificados padrões também neste domínio. Uma vez que um sistema precisa ser testável e (pelo menos **black-box**) testes são realizados em relação aos requisitos, os padrões devem ser usadas em uma metodologia de desenvolvimento de sistema integrado. Assim, toda prática de teste precisa ser coletada por um esforço sistemático em que possam ser organizadas soluções efetivas recorrentes em um catálogo de padrões de teste [9]. Os maiores benefícios podem ser alcançados através de modelos já em fase de captura dos requisitos. A dificuldade de concretizar o sucesso deste trabalho é a extração dos padrões de forma correta e útil. No entanto, uma vez identificados os padrões, torna-se relativamente mais fácil testar os cenários que se utilizam do mesmo, devido ao intenso reuso.

A escolha do formalismo se deve ao fato de que diagramas de seqüência fornecem uma visão dinâmica do sistema por, através de gráficos, mostrar situações que podem ocorrer em tempo de execução quando objetos interagem para realizar tarefas. Dos treze modelos utilizados em UML 2.0, apenas diagramas de seqüência mostram as mensagens trocadas entre os objetos na ordem em que as mensagens ocorrem, resultando numa maior legibilidade e maior compreensão do modelo. Além do mais, várias melhorias foram incorporadas na nova versão do *framework* UML (detalhes em [43]).

Apesar de estarmos enquadrados especificamente na fase de modelagem do processo de Testes Baseados em Modelos, resultados precisam ser concretizados levando em consideração o procedimento completo. Para tanto, acrescentamos ao procedimento sistemático de geração e seleção de casos de teste automatizado pela ferramenta LTS-BT [16] o modelo resultante da abordagem de escrita de modelo de teste aqui porposto.

1.1 Objetivo

Partindo do pressuposto dos problemas gerais associados à adoção industrial do processo MBT, Nascimento realizou um estudo [22] experimental com intuito de reportar problemas associados ao contexto específico do ambiente CInBTCRD. Instrumentada por métricas avaliadas em um modelo de medição elaborado com características específicas, a autora realizou experimentos dentro do ambiente de teste da Motorola na intenção de avaliar a eficiência da adoção de MBT no processo de teste de *feature*. A análise experimental diagnosticou problemas relacionados à fase de tradução do documento de requisitos para o modelo de teste.

A intenção deste trabalho é de buscar soluções que facilitem a adoção do processo MBT de forma geral a partir de melhorias na fase de elaboração do modelo de teste. No entanto, foi conduzido especificamente dentro do projeto *CIn Brazil Test Center Research and Development*(CInBTCRD). Ou seja, o mérito do trabalho trará contribuições gerais, apesar de sua execução e validação ter ocorrido em ambiente específico.

Este trabalho tem por objetivo geral propor uma abordagem de construção do modelo de caso de uso utilizado como entrada pela técnica MBT. O objetivo está inserido no nível conceitual de especificação de requisitos para derivação de casos de teste. O problema fundamental a ser investigado é: como facilitar o processo de escrita de casos de uso a partir de documentos de requisitos e, ao mesmo tempo, tornar o processo mais apropriado e completo para a geração de casos de teste?

Para tal, os objetivos específicos deste trabalho são:

- Obtenção e formalização de padrões de requisitos do domínio (requisitos implícitos) que usualmente não são (completamente) definidos em documentos de requisitos.
- Auxiliar o processo de especificação de casos de uso a partir da síntese (combinação) de padrões de requisitos existentes com comportamentos específicos de um dado requisito.
- Auxiliar o processo de especificação de casos de uso a partir do uso de padrões de comportamento que detalham ações abstratas padrão para um domínio de aplicações.

1.2 Contribuições

A automação de um processo não significa necessariamente no aumento da produtividade. Utilizando a mesma ferramenta de geração automática de casos de teste é possível resultados bem diferentes, tanto na cobertura de requisitos como pela eficiência em encontrar defeitos, a depender do conhecimento e habilidade de quem ficou responsável pela elaboração do modelo de teste. Além disso, uma ferramenta pode sistematizar todo o processo de rastreabilidade de requisitos, mas não há garantia de que os requisitos foram inseridos corretamente e de forma completa nos cenários que compõem o modelo de teste.

Portanto, a adoção de MBT pela indústria exige mais do que uma ferramenta de geração de casos de teste. É necessário um ambiente integrado de suporte onde é possível facilitar a escrita e a manutenção dos modelos de teste, pois o modelo consiste do artefato principal da abordagem.

Neste sentindo, contribuímos com a fase de especificação do modelo de teste no contexto de Testes Baseados em Modelos pelos seguintes resultados:

- **Uma estratégia de escrita de cenários de caso de uso utilizando como formalismo diagramas de seqüência da versão 2 do *framework* UML** - através da construção de um repositório onde é possível reutilizar modelos que representam ações padrões de um domínio, possibilitando compartilhar conhecimento. Como a reutilização das idéias e do conhecimento se torna mais e mais fundamental, um padrão pode ser uma forma eficaz de transmitir experiências sobre problemas recorrentes no domínio de desenvolvimento de Modelo de *Software*. Longe da pretensão de tornar o testador experiente, mas de reutilizar conhecimento de forma explícita. Um ganho significante decorrente resultado foi a elaboração de modelos mais consistentes entre si; diminuindo, portanto, as variações produzidas por testadores individuais.

- **Uma notação para relacionar casos de uso** - A notação proposta para definir relações entre casos de uso tornou a legibilidade dos documentos muito melhor e ocasionou mais agilidade na fase de inspeção. A idéia da notação é tornar explícita (legível) as relações entre os casos de uso, pois nosso modelo teste é elaborado de forma manual pela escrita de cenários utilizando a linguagem natural e, portanto, a inspeção realizada ocorre pela leitura do modelo de forma não-automática, geralmente, por uma pessoa

diferente da que elaborou o modelo.

- **Estudo de caso** - Realizar estudo de caso dentro do ambiente de execução real é um grande desafio. É preciso não interferir muito na execução normal das atividades e, ao mesmo tempo, coletar valores significativos. O momento precisa ser bem definido e as pessoas devem estar dispostas e/ou disponíveis para colaborar com o experimento. Geralmente, não há possibilidade de repetição e o planejamento é crucial. Como conseqüência,o experimento pode acabar com escopo reduzido e com poucos integrantes colaborando(ver Capítulo 5). O que, no entanto, não diminui o seu valor experimental.

Limitações e dificuldades foram encontradas devido a utilização de documentos e informações que são propriedade intelectual da Motorola. No entanto, o fato de trabalhar em parceria com uma empresa, permitiu-nos realizar estudos de caso com aplicações reais e propor soluções de problemas também reais.

1.3 Contexto

Este trabalho faz parte do *CIn Brazil Test Center Research and Development* (CInBTCRD), uma cooperação entre a Motorola *Inc.*, o Centro de Informática da Universidade Federal de Pernambuco (CIn/UFPE), contando ainda com a colaboração do Centro de Engenharia Elétrica e Informática da Universidade Federal de Campina Grande (CEEI/UFCG). Esta cooperação foi iniciada em 2003 tendo por objetivo inicial a criação de recursos humanos especializados na área de teste de *software*.

Basicamente, o projeto está dividido em 3 áreas: treinamento e educação formal, operação e pesquisa. Esta dissertação é um dos resultados pertencentes ao grupo de pesquisa. O principal objetivo do time de pesquisa é analisar os artefatos e o processo da Motorola, buscando por possíveis melhorias e propondo soluções viáveis para automatizar algumas das atividades, como especificação de requisitos, *design* de teste e criação de artefatos de *design*. O projeto também propõe maneiras de avaliar cobertura de casos de teste, de melhorar a seleção dos casos de teste, além de guiar o plano de execução com estimativas.

No ambiente *CIn Brazil Test Center Research and Development* (CInBTCRD), o contexto MBT está resumido na Figura 1.1. O documento base de onde são retiradas as

informações que irão compor o modelo de teste é o documento de requisitos. Existem duas representações para o modelo de teste: documentos de caso de uso, se a ferramenta utilizada for TaRGeT [15], ou diagramas de seqüência, caso seja utilizada a ferramenta LTS-BT [16]. Dada a formalização dos requisitos em alguma das representações, um modelo intermediário (escrito em *Labelled Transition Systems* (LTS) [58]) é gerado servindo de base para a geração automática dos casos de teste.

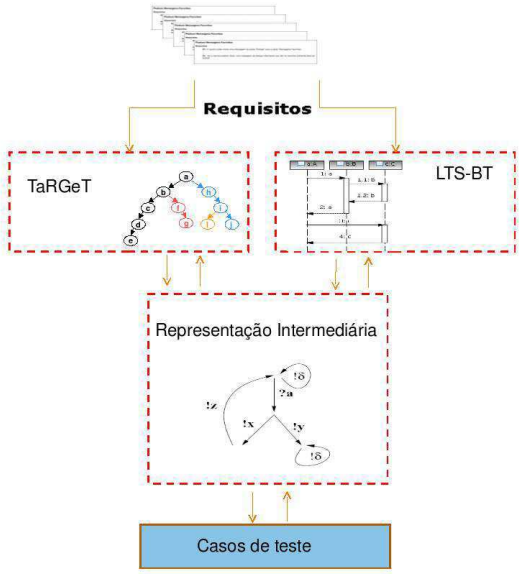


Figura 1.1: Visão geral do processo de Teste baseado em Modelos no CInBTCRD

1.4 Estrutura dos capítulos

As partes seguintes deste documento estão estruturadas da seguinte forma:

Capítulo 2: Fundamentação teórica Neste capítulo são descritos alguns dos conceitos essenciais que serviram de base teórica para formação do nosso trabalho. Descrevemos alguns processos existentes para automação de testes, diagramas de seqüência, caso de uso, LTS e padrões de testes.

Capítulo 3: Padrão de especificação de caso de teste Este capítulo tem por objetivo

explicar o processo de catálogo de padrões no contexto de aplicações para celulares. Apresentamos o contexto de aplicações de celulares e a estratégia de especificação do modelo de teste com auxílio de padrões. É mostrado como ocorre a adição e atualizações dos padrões. Por fim, temos um exemplo ilustrativo da técnica.

Capítulo 4: Procedimento de Testes Baseados em Modelos Neste capítulo, explicamos nossa estratégia de escrita de modelo casos de uso dentro de um contexto completo do processo de Testes Baseados em Modelos. Acrescentamos o processo de escrita ao processo sistemático da ferramenta LTS-BT.

Capítulo 5: Estudo de Caso Neste capítulo, apresentamos os resultados obtidos de um estudo de caso com a definição de um modelo de medição.

Capítulo 6: Conclusão Este capítulo final apresenta algumas considerações sobre o trabalho desenvolvido e as perspectivas para trabalhos futuros.

Capítulo 2

Fundamentação teórica

Nossa técnica está inserida num contexto de teorias que serão abordadas neste capítulo. Pretendemos, nas seções seguintes, descrever brevemente alguns dos conceitos essenciais que serviram de base para formação do nosso trabalho.

2.1 Processos de Automação de Teste de Software

O desenvolvimento de métodos de desenvolvimento de *software* tem colocado desafios para testes de *software* como parte da verificação da qualidade do *software*. Tem havido numerosas tentativas de melhorar a qualidade de verificação de *software* de várias formas, pois, teste de *software* está associado a um custo de até 50% do custo total de um projeto [7]. Neste cenário, a automação do processo de testes se torna não apenas desejável, mas uma necessidade.

A automação é dividida em três gerações [35]:

- **A primeira geração:** composta basicamente de testes de GUI com ferramentas de captura e repetição, onde os scripts possuem passos e dados de teste juntos no código;
- **A segunda geração:** começam-se a construir *frameworks*, preocupando-se com scripts de teste bem estruturados, documentados, robustos e fáceis de manter. A característica chave desta geração é a reusabilidade. Mas se há 10000 casos de teste, então haverá 10000 scripts de teste;

- **A terceira geração:** caracteriza-se por separar as atividades de projeto e de implementação de teste. O teste pode ser projetado por um testador que não é desenvolvedor, que conhece o domínio da aplicação e as técnicas de projeto de testes. O mesmo teste é aproveitado pelos desenvolvedores do *framework*, sem re-trabalho.

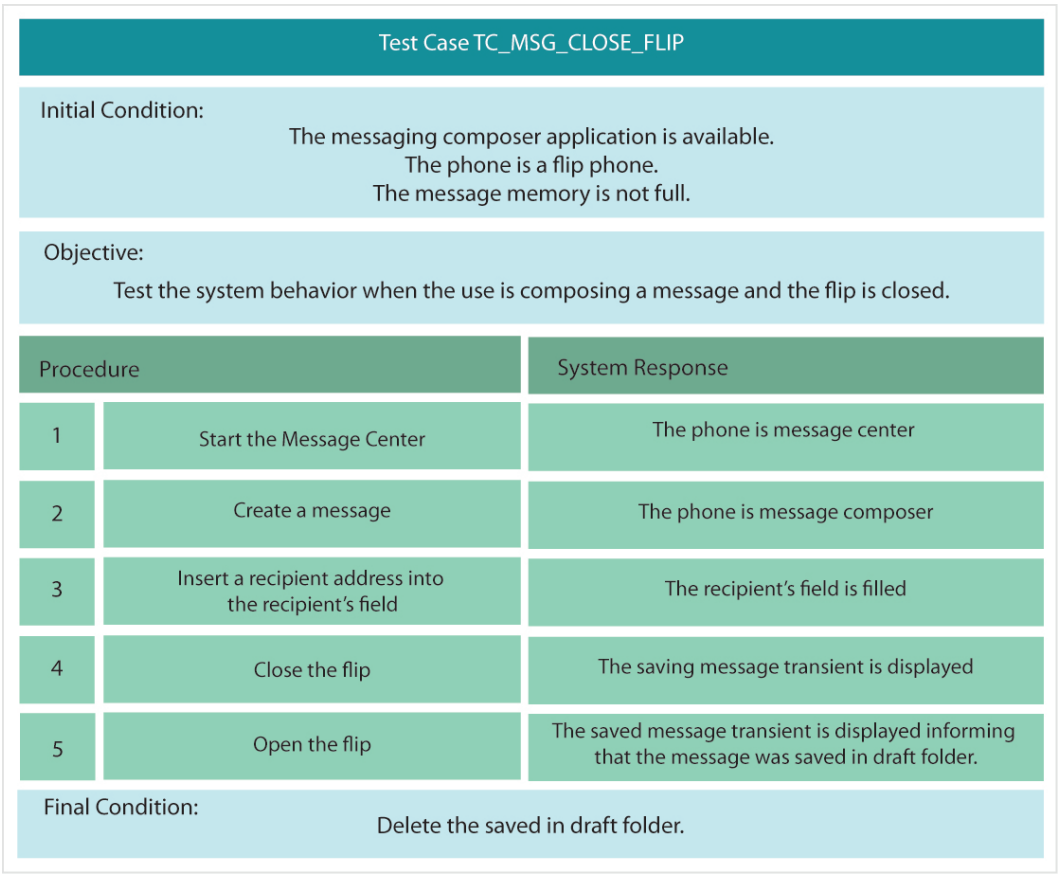
Este seção descreve o impacto de adoção de testes baseados em modelos como processo para automação de teste. Para tanto, serão analisadas algumas abordagens de teste onde será descrito o funcionamento do processo bem como papéis envolvidos e suas respectivas habilidades. Na definição de cada abordagem, utilizamos o termo SUT - *System Under Test* - para denotar o sistema que está sendo testado.

2.1.1 Processo de teste manual

O processo de teste manual se caracteriza pelo processo onde o projetista, baseado nos documentos de requisitos e guiado por um plano de casos de teste, elabora documentos descrevendo casos de teste. Os casos de teste gerados são casos de teste manuais, pois a sua execução é realizada de forma também manual. Para cada caso de teste, o testador segue os passos descritos, interage diretamente com a SUT, compara a saída gerada pela SUT com a saída esperada e grava o veredicto do teste. A descrição dos casos de teste pode ser bastante concisa e de alto nível, muitos dos detalhes de baixo nível a respeito da interação com o sistema sob o teste pode ser deixada para o senso comum de execução do responsável por executar os testes, mais conhecido por testador manual. Temos, portanto, a necessidade de dois papéis - projetista de teste e testador manual - com habilidades diferentes. O projetista de teste precisa ter conhecimentos técnicos sobre a SUT, além de algumas habilidades com estratégias de *design* de teste. Do testador manual pode ser exigido algum conhecimento de como interagir com a SUT, ou pode envolver, simplesmente, seguir os passos do caso de teste e gravar os resultados.

A Figura 2.1 ilustra um exemplo de caso de teste manual. O objetivo do caso de teste é testar o comportamento do celular quando o usuário está escrevendo uma mensagem e o *flip* é fechado. O campo *initial conditions* indica as condições que devem ser satisfeitas antes que seja iniciada a execução do teste. Neste caso, antes de executar os testes, é necessário verificar se o compositor de mensagens está disponível, se o fone é do tipo que possui *flip* e,

por fim, verificar se existe memória disponível para armazenamento de mensagem. Em *final condition*, são executadas ações para resgatar o estado em que se encontrava o fone antes de serem executados os testes.



Test Case TC_MSG_CLOSE_FLIP		
Initial Condition: The messaging composer application is available. The phone is a flip phone. The message memory is not full.		
Objective: Test the system behavior when the use is composing a message and the flip is closed.		
Procedure		System Response
1	Start the Message Center	The phone is message center
2	Create a message	The phone is message composer
3	Insert a recipient address into the recipient's field	The recipient's field is filled
4	Close the flip	The saving message transient is displayed
5	Open the flip	The saved message transient is displayed informing that the message was saved in draft folder.
Final Condition: Delete the saved in draft folder.		

Figura 2.1: Instância de um teste manual

2.1.2 Captura e repetição

Neste processo, os casos de teste continuam sendo projetados manualmente. Os papéis envolvidos continuam sendo os mesmos com uma sutil diferença na interação do testador manual com a SUT. A interação com a SUT ocorre através de uma ferramenta de teste que realiza a "captura e repetição". Esta registra todas as entradas enviadas para o sistema em teste e as saídas resultantes. Depois, quando uma nova versão da SUT deve ser testada, a ferramenta tenta re-executar todos os testes gravados e relatar quais que falharam. Para cada reprise de teste gravado, a ferramenta envia as entradas registradas à SUT e, em seguida, compara os resultados das novas saídas registradas com o resultado gravado da execução

original [35]. Ou seja, a interação do testador manual pode ser reaproveitada.

A partir deste processo é possível automação na execução. Entretanto, a questão fundamental de automatizar a execução do teste está apenas parcialmente resolvida. Isto é devido à extrema sensibilidade desta abordagem a eventuais mudanças na SUT. Por exemplo, uma mudança no *layout* de uma janela ou uma pequena alteração na interface de um processo em uma API pode causar um grande número de testes a serem descartados. Estes têm de ser testados manualmente novamente e gravados para futuras sessões demandando interação do testador manual. Esta abordagem é , entretanto, útil para os testes que simulam a interação do usuário com o sistema sob teste através da GUI.

2.1.3 Processo de teste Baseado em *Script*

Processo de teste baseado na criação de *scripts* de teste. Um *script* de teste é um *script* executável que executa um conjunto de casos de teste. Neste processo estão envolvidas algumas tarefas como:

- Inicializar a SUT;
- Colocar a SUT no contexto exigido;
- Criar valores de entrada para os testes;
- Comparar respostas da SUT com os resultados esperados;
- Atribuir um veredicto (passou/falhou) para cada teste.

Os *scripts* de teste podem ser escritos em linguagem de *scripts* ou em linguagem específica de teste como TTCN-3 (*Testing and Test Control Notation*) [55], ou mesmo linguagem padrão de programação. Portanto, são exigidos dos papéis relacionados ao processo habilidades além de projeto e execução de testes.

Esta abordagem resolve o problema de execução de teste, automatizando-a. A re-execução dos testes consiste em re-executar os *scripts*. No entanto, isto aumenta o problema de manutenção de testes devido não apenas por mudanças de requisitos, mas também quando alguns detalhes de implementação mudam. Uma vez que o tamanho total dos *scripts* de teste pode ser quase tão grande quanto a aplicação sob o teste e os detalhes de uma interface para

a SUT são normalmente repartidos ao longo de muitos testes, a manutenção do teste *scripts* se torna muito dispendiosa.

Scripts de teste são úteis quando interações humanas são inviáveis, por exemplo, enviar várias mensagens para outro telefone, ou inserir cem contatos na lista telefônica. A Figura 2.2 mostra um script que especifica um cenário de teste que verifica se as mensagens são exibidas corretamente na pasta "caixa de saída".

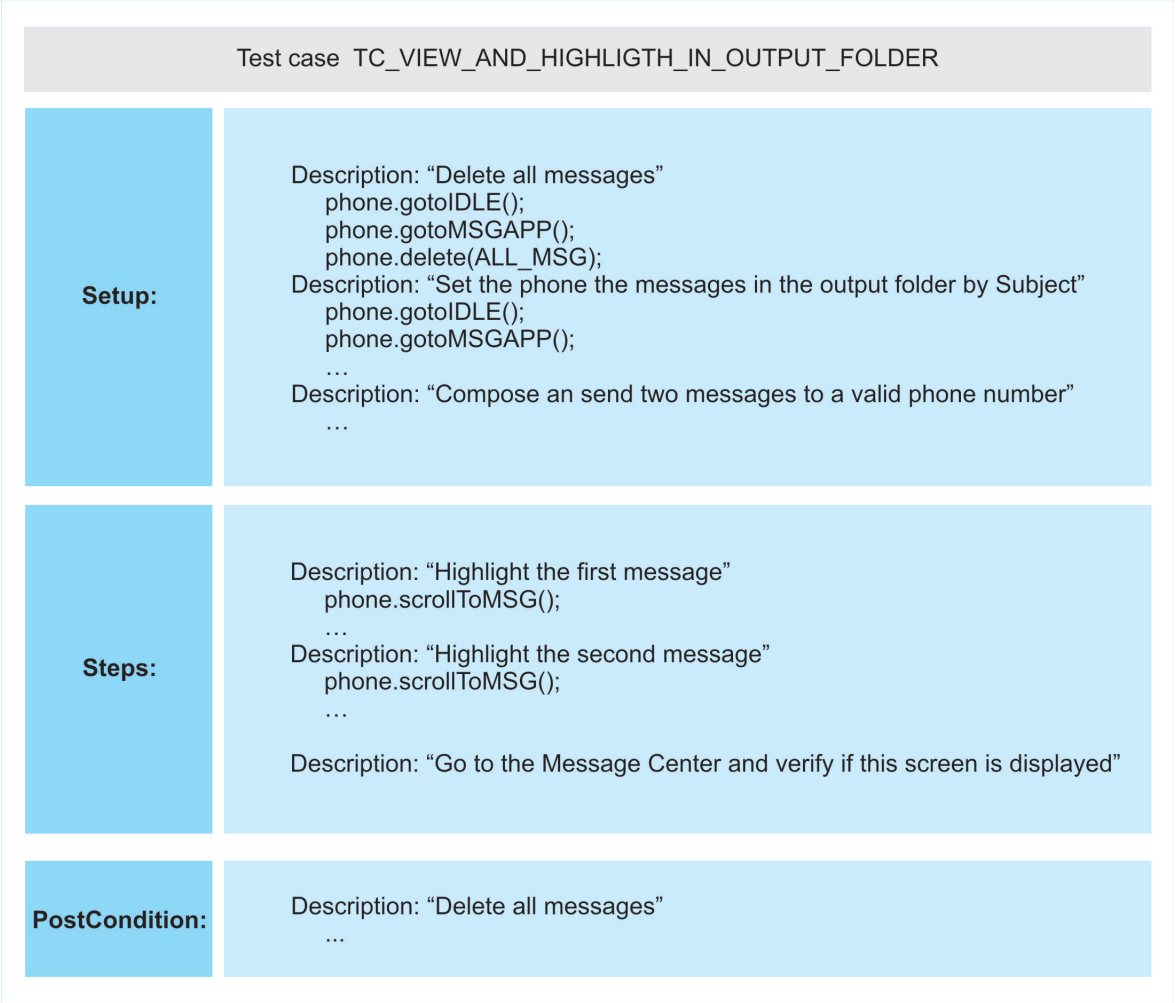


Figura 2.2: Instância de um *script* de teste

Este *script* de teste é formado de três partes: *setup*, *steps* e *pos conditions*. A semântica dos campos se assemelha ao caso de teste manual com uma diferença apenas de nomeclatura para os campos *setup* (*inicial condition*), *pos condition*(*final condition*) e *step* (*test procedure*). Outra diferença se refere ao conjunto de passos (*step*), os resultados esperados (*expected results*) não são mostrados no caso teste. No entanto, etapas de validação são

executadas internamente por um *framework* de automação em teste.

Outro aspecto importante sobre este *script* de teste é que todos os passos do teste são primeiro especificado em um nível mais elevado (description) e, em seguida, os passos são descritos na linguagem do script. A etapa em alto nível é especificada usando linguagem natural (em Inglês), como mostrado pelo texto entre aspas dentro de comandos *description*. A descrição define o objetivo dos passos, enquanto que os comandos de *script* seguintes descrevem as ações automáticas para alcançar esse objetivo.

2.1.4 Processo de teste baseado em palavras-chave

A idéia é expressar os casos de teste de forma mais abstrata possível enquanto mantém-nos precisos o suficiente para serem executados. Cada palavra chave corresponde a fragmento de um *script* de teste. Um ponto forte deste processo é o alto nível de abstração dos casos de teste que são de fácil leitura, mas continuam sendo executáveis. O nível de abstração torna a manutenção dos casos de teste mais simples, pois os casos de teste se tornam adaptados a novas versões da SUT pela simples modificação dos *scripts* associados com poucas palavras chaves. No entanto, os dados de teste e oráculo continuam sendo projetados manualmente, além da cobertura de teste em relação à requisitos também ser realizada de forma manual.

2.1.5 Teste baseado em modelos

Outra abordagem para automação do processo de teste de *software* é a técnica baseada em modelos. Testes Baseados em Modelos ou MBT (*Model-Based Testing*) [23] é uma técnica de teste de *software* onde a geração dos testes ocorre a partir do modelo de *software*. A idéia de MBT é comparar comportamentos de entrada e saída do modelo do sistema com o sistema propriamente dito. Em outras palavras, é o uso do modelo abstrato para testar uma implementação concreta.

O que difere MBT dos outros métodos de automação não é o uso de modelo; pois, como afirma Karner *et al.* [33], todo tipo de teste pode ser considerado *model-based*. Quando se compara diferentes estilos de teste, o que varia é o quão explícito e o tempo de vida do modelo, mas o modelo sempre existe. MBT ocorre quando o modelo é explícito e persistente. O que difere MBT das outras práticas de teste é o nível de automação. A automação

antecipa a execução, chegando ao *design* dos casos de teste. Ou seja, ao invés de escrever os casos de teste dos requisitos manualmente, cria-se um modelo que representa estes casos de teste de forma compacta. Então, as ferramentas de MBT geram testes deste modelo automaticamente.

Existem quatro abordagens principais conhecidas como *model-based testing*[61] :

1. Geração de dados de entrada de teste a partir de um modelo do domínio.
2. Geração de casos de teste de um modelo de comportamento do sistema.
3. Geração de casos de teste com oráculos de um modelo de comportamento do sistema.
4. Geração de *scripts* de testes a partir de testes abstratos.

Na geração de dados de entrada, o modelo é a informação sobre os domínios dos valores de entrada e a geração implica em seleção e combinação de um subconjunto de valores para produzir testes para estes dados de entrada. A segunda abordagem gera seqüência de chamadas à SUT a partir do modelo do ambiente do sistema. Na geração de casos de teste com oráculos, o gerador de teste deve conhecer o suficiente sobre o comportamento esperado da SUT para ser capaz de prever ou controlar os valores de saída da SUT. O último tipo de geração considera a transformação de casos de teste em alto nível obtidos de um modelo de descrição abstrato do sistema em *scripts* que seriam os casos de teste em baixo nível.

Neste trabalho, seguimos a terceira abordagem, ou seja, geramos casos de teste a partir de modelos do comportamento do sistema com informações sobre resultados esperados. O modelo possui informações de entrada do teste e saída do teste, bem como condições que relacionam estas informações.

Em suma, o processo de teste baseado em modelos pode ser caracterizados por cinco etapas principais[61]:

1. Modelar a SUT e seu ambiente;
2. Gerar testes abstratos do modelo;
3. Concretizar os testes para deixá-los executáveis;
4. Executar os testes na SUT e atribuir veredictos;

5. Analisar o resultado dos testes.

O ponto chave da abordagem consiste na primeira etapa. As duas últimas etapas são etapas comuns a qualquer processo de teste, mesmo o teste manual. A concretização dos casos de teste (etapa 3) pode variar deixando MBT mais semelhante à uma técnica dependendo da medida adotada. A execução pode ser realizada de forma manual ou pode gerar *scripts*.

Ainda que seja considerada a quarta geração de automação de teste, a idéia de teste baseado em modelos se remete à década de cinqüenta com estudos realizados por Moore [38] sobre a geração de testes a partir de máquinas de estado finitas. Existe um impacto associado quanto à sua implantação. O maior impacto deste processo é que o modelo atua como o plano de teste e a manutenção dos casos de teste, devido a mudanças de requisitos, se refletem em alterações no modelo. O responsável da equipe de teste pela construção do modelo de teste deverá estar fundamentado em teorias e práticas de modelo que, em muitos casos, foge à normalidade de suas atividades. As habilidades impostas aos papéis da equipe de teste representam, portanto, o maior empecilho na adoção da técnica. Um número maior de barreiras bem como detalhes desta afirmação é apresentado por Rosenblum [50]. A vantagem é que o processo de teste se constitui de uma atividade que ocorre mais cedo dentro do processo de desenvolvimento. Além do mais, por ser uma técnica sistmática, a automação pode levar a uma cobertura mais ampla e efetiva.

Geração de testes

O principal benefício de MBT,e o que o diferencia dos outro métodos, é a geração automática de teste. Geração automática de teste exige uma descrição do sistema e requisitos de teste. A coleção de algoritmos para guiar a seleção de casos de teste varia para cada ferramenta. Uma boa ferramenta suportará vários tipos de critério para conferir o maior controle possível sobre os testes gerados. Os algoritmos podem ser divididos em três categorias principais: critérios de cobertura, metas e técnicas de caminhamento [61].

Critérios de cobertura mostram o quanto uma suíte de teste cobre o modelo. A maioria dos algoritmos de critérios cobertura, bem como os termos utilizados, são derivados das técnicas de testes caixa branca onde o significado está associado a abrangência de código. Em MBT, os critérios de cobertura relacionam o quão bem a suite de teste gerada abrange o modelo. Por isso, em MBT, cobertura total não significa que o código é testado em sua

totalidade. Cobertura total significa apenas que há uma suíte de teste gerada cobrindo o modelo a partir de um aspecto de critério selecionado[61].

Alguns tipos mais comuns de critérios de cobertura são:

- **Cobertura de estado:** mede a eficiência com que uma suíte de teste abrange os estados do modelo. Cobertura total significa que cada estado do modelo é testado pelo menos em um teste da suíte.

- **Cobertura de transições:** representa transições que são abrangidas na suíte de testes. Usando esse critério, é útil saber como a ferramenta gerencia transições bidirecionais. Transições bidirecionais podem ser necessária para testar ambos os sentidos ou apenas uma direção a ser coberta.

- **Análise de valor limite:** gerar testes que estão por perto de limite imposto. Normalmente, cada limite é testado com três valores: abaixo, igual e superior ao valor limite.

- **Rota mais provável:** cobre as rotas cuja probabilidade é maior de ocorrerem numa execução. Esta cobertura requer informações probabilistas no modelo.

Metas são pontos anotados no modelo. Cobrimos estes pontos pela geração passando por eles. São também descritos como requisitos, uma vez que uma meta pode ser usada para determinar requisitos em casos de teste. Estes pontos colocados no modelo dão uma boa rastreabilidade[61].

Técnicas de caminhamento determinam a forma como o gerador de teste caminha através do modelo durante a geração de testes.

2.2 Diagramas de seqüencia

A idéia de utilizar mensagens em seqüência para descrever aspectos do comportamento dos sistemas é antiga, dado que a primeira formalização foi em 1992 com MSC-92 (*Message Sequence Chart*) [17]. Diagramas de Seqüência UML 1.4 são comparáveis aos da MSC-92, enquanto os mais recentes diagramas UML 2.0 estão em conformidade com MSC-2000 [28].

Diagramas de interação em UML são uma especificação de como as mensagens são enviadas entre objetos para executar uma tarefa. Interações são usadas em uma série de

situações diferentes. Elas são usadas para obter uma maior compreensão da comunicação de um cenário ou para um grupo que necessita alcançar um entendimento comum de uma situação. Interações também são usadas durante a fase mais detalhada do *design*, onde um processo preciso de intercomunicação deve ser criado de acordo com protocolos formais. Quando o teste é realizado, o comportamento do sistema pode ser descrito como interações descrevendo cenários [43].

Nesta seção falaremos especificamente do tipo de diagrama de interação conhecido por diagrama de seqüencia. Eles são declarações parciais sobre um sistema (ou uma parte de um sistema) e definem propriedades do sistema, mas não necessariamente todas as propriedades relevantes. Os aspectos mais visíveis de uma interação em UML são as mensagens entre os componentes. A seqüência das mensagens é considerada importante para a compreensão da situação descrita no diagrama. Os dados que transmitem as mensagens também podem ser muito importantes, mas as interações não se centram na manipulação de dados, apesar de os dados poderem ser utilizados para decorar os diagramas [43].

A seqüencia das mensagens é o ponto chave do que é explicado através de uma interação. Os possíveis fluxos de controle em todo o processo são descrito em duas dimensões, a dimensão horizontal mostrando os objetos ativos, e a dimensão vertical mostrando a ordenação no tempo. Esta seqüencia é conhecida por *trace*. Um *trace* é uma seqüência de eventos ordenados no tempo. Um *trace* descreve o histórico da troca de mensagens correspondente à execução passo a passo de um sistema. Um *trace* pode ser parcial ou total.

Um dos principais usos dos diagramas de seqüência está na transição de requisitos expressos como casos de uso para o próximo nível mais formal de requinte do processo. Casos de uso são muitas vezes refinados em um ou mais diagramas de seqüência.

2.2.1 Elementos

O elemento *frame* é utilizado como base para muitos outros elementos no diagrama de UML 2 fornecendo um limite para o diagrama. O elemento *frame* fornece um modo coerente para colocar um rótulo no diagrama, ao mesmo tempo em que proporciona uma fronteira gráfica para o diagrama. Dentro do limite do *frame* existem os elementos que compõem o contexto do diagrama formado por três elementos básicos principais:

- ***Lifeline***: representam instâncias de objetos que participam do diagrama modelado. São representados por uma caixa contendo a classe ou instância específica da classe e possui uma linha pontilhada descendo a partir do centro da caixa.

- **Mensagens:** A notação de uma mensagem é sempre uma seta, mas a natureza da seta e da flecha varia com base no tipo de mensagem, como segue:

  - Uma chamada síncrona aparece como uma linha sólida com uma seta sólida.
  - Uma chamada assíncrona aparece como uma linha sólida com uma meia-flecha.
  - Uma mensagem de resposta aparece como uma linha tracejada com uma seta sólida.
  - Um mensagem "lost"(na qual o remetente é conhecido, mas o receptor não é), tem um pequeno círculo preto ao lado da ponta da flecha.
  - A mensagem "found"(na qual o receptor é conhecido, mas o remetente não é), tem um pequeno círculo preto ao lado do início da seta.

- **Guarda:** são utilizados para controlar o fluxo dos diagramas. A notação é bem simples: uma expressão *booleana* entre colchetes.

2.2.2 Fragmentos combinados

Alguns *traces* podem possuir restrições ou propriedades particulares. Por exemplo, pode existir uma região crítica dentro de uma interação onde um conjunto de chamadas de método deve executar atomicamente, ou um laço que itera sobre uma coleção. UML chama estas partes menores de fragmentos de trace de *interaction fragments*. Estes fragmentos podem ser combinados em um contêiner conhecido por *combined fragment*. Estes contêineres são compostos de um operador e um ou mais *interaction fragments* (operandos). Cada operador tem associado um número de operandos e uma palavra-chave que é colocada no pentágono de um *combined fragment*.

- ***alt:*** representa uma escolha de comportamentos. Cada fragmento tem associado um guarda, o que significa que, no máximo, um dos operandos irá executar.

- ***assert:*** representa uma afirmação: O fragmento deve ocorrer exatamente como especificado.
- ***break:*** representa uma quebra de cenário: O operando é realizada em vez de o restante da interação.
- ***consider:*** define os tipos de mensagens que devem ser considerados.
- ***critical:*** representa uma região crítica: O sistema ignora quaisquer interrupções na região até que tudo tenha acabado na execução.
- ***ignore:*** o fragmento combinado não mostra, pelo menos, um tipo de mensagem, o que significa que estes tipos de mensagens podem ser consideradas insignificantes e são ignorados se aparecerem em um comportamento correspondente na execução.
- ***loop:*** representa um ciclo: o operando do ciclo é repetido o número especificado de vezes.
- ***neg:*** representa uma ou várias mensagens definida(s) como inválida(s).
- ***opt:*** representa uma opção onde um comportamento acontece ou não acontece, com base na avaliação dos guarda associados.
- ***par:*** representa um merge paralelo entre os operandos com restrição de que não seja perdida a ordem em um mesmo operando.
- ***seq:*** representa um seqüencia "weak"entre os comportamentos dos operandos, o que significa o seguinte: [a] A ordenação das mensagens dentro de cada um dos operandos é mantido no resultado; [b] mensagens em diferentes *lifelines* de diferentes operandos podem vir em qualquer ordem, e [c] mensagens sobre o mesmo *lifeline* de diferentes operandos devem ocorrer de forma que uma mensagem dentro do primeiro operando venha antes do segundo operando.
- ***strict:*** representa um seqüencia rigorosa entre os comportamentos dos operandos.

Na Figura 2.3 temos uma representação gráfica de um diagrama de seqüencia com alguns dos elementos já citados.

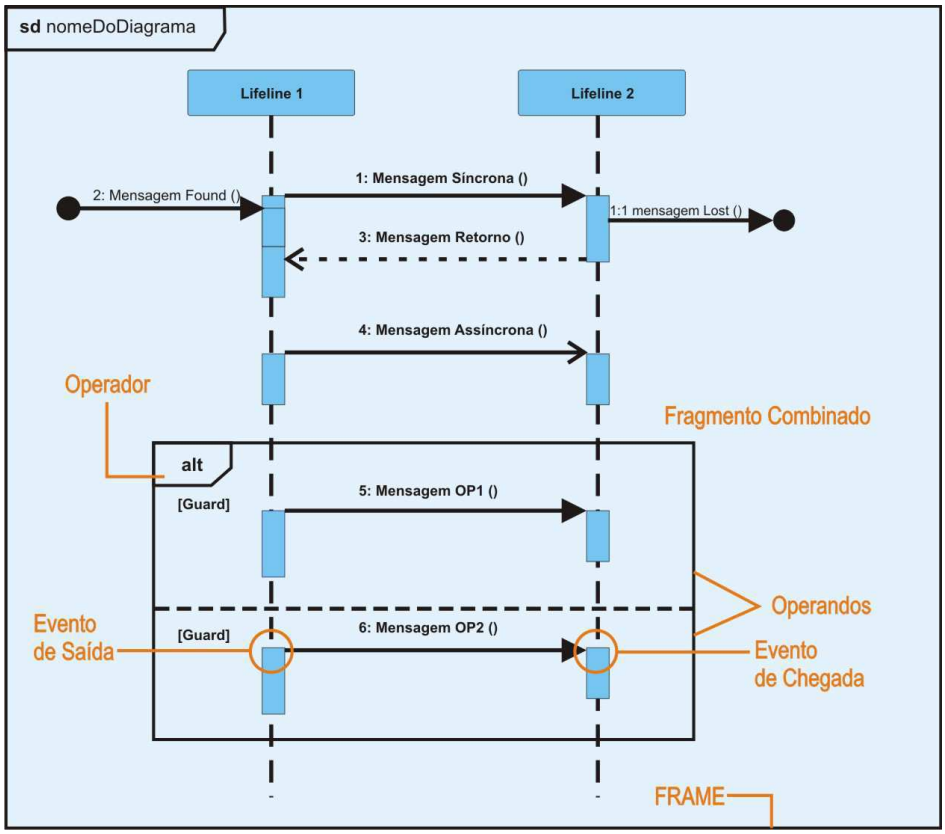


Figura 2.3: Exemplo de diagrama de seqüencia

2.2.3 Modularização de diagramas de seqüencia

A adição de *interaction occurrences* é a mais importante inovação na modelagem de diagramas de interação. Através deste elemento é possível compor diagramas de seqüência simples em diagramas de seqüência complexos [43]. Ou seja, é possível tanto combinar diagramas simples para construir diagramas complexos, como combinar estes diagramas simples para complementar diagramas com detalhes. Significa portanto, que se pode abstrair um comportamento de seqüência em uma unidade conceitual simples. A representação gráfica deste elemento é um quadro delimitador com a palavra reservada *ref* utilizada dentro da caixa de nome no canto superior esquerdo do quadro.

2.3 *Labeled Transition Systems*

Labeled Transition Systems (LTS) [59] é um formalismo que especifica a semântica do comportamento do sistema através do paradigma de notação baseada em transição. LTSs são

representados por grafos onde os estados representam as possíveis configurações do sistema e as arestas representam o movimento entre essas configurações através da ocorrência de ações.

Formalmente, são uma 4-tupla $< Q, A, T, q_0 >$, onde:

- Q é um conjunto finito e não-vazio de estados;
- A é um conjunto finito de ações;
- $T \subseteq Q \times A \times Q$ é a relação de transição;
- $q_0 \in Q$ é o estado inicial.

LTSs são basicamente grafos direcionados com rótulos nas arestas. A Figura 2.4 apresenta um LTS que representa o comportamento de um caso de uso que expressa a funcionalidade de remoção de uma mensagem que está na caixa de entrada de uma *feature* de celular.

2.3.1 LTS Anotado

Um LTS Anotado possui uma diferença sutil do LTS normal, no mais segue basicamente a mesma definição apresentada na seção anterior. A diferença entre um LTS e um LTS Anotado está nos rótulos das transições.

Formalmente, são uma 4-tupla $< Q, R, T, q_0 >$, onde:

- Q é um conjunto finito e não-vazio de estados;
- $R = A \cup N$ é um conjunto finito de rótulos, onde A é um conjunto finito de ações e N é um conjunto finito de anotações;
- $T \subseteq Q \times R \times Q$ é a relação de transição;
- $q_0 \in Q$ é o estado inicial.

A Figura 2.13 apresenta o LTS visto na Figura 2.4 anotado com algumas informações inerentes ao processo teste. Como pode ser observado na Figura 2.13, tais informações são: *Passos*, *Resultados Esperados* e *Condições Iniciais*. Essas anotações inseridas no LTS são

Figura 2.4: Exemplo de LTS

utilizadas para separar as principais informações de um caso de teste: os passos ou ações a serem executados, os resultados esperados e as condições iniciais sob as quais as ações podem ser realizadas.

2.4 Caso de Uso

Um caso de uso descreve o comportamento de um sistema sob várias condições em resposta a um pedido do usuário. O sistema é visto somente pela visão de interface, sem detalhes da implementação interna [18]. A idéia geral é pensar no sistema em termos de sua comunicação com as entidades fora do sistema. Do ponto de vista UML [43], casos de uso

são diagramas simples representando atores e casos de uso e as diferentes relações entre eles.

A simplicidade dos diagramas de casos de uso traz a desvantagem de serem pouco precisos, podendo não definir adequadamente o sistema. Casos de uso são, por isso, utilizados dentro de uma abordagem orientada a cenários onde os diagramas são acompanhados por um conjunto de descrições, também chamadas narrativas [46]. Estas narrativas incluem uma descrição textual dos casos de uso individuais e a extração de cenários. A descrição textual descreve requisitos detalhados, enquanto cenários abordam a necessidade de explorar opções na execução do caso de uso para testar os requisitos e prover um plano de teste em alto nível para fases posterior de desenvolvimento.

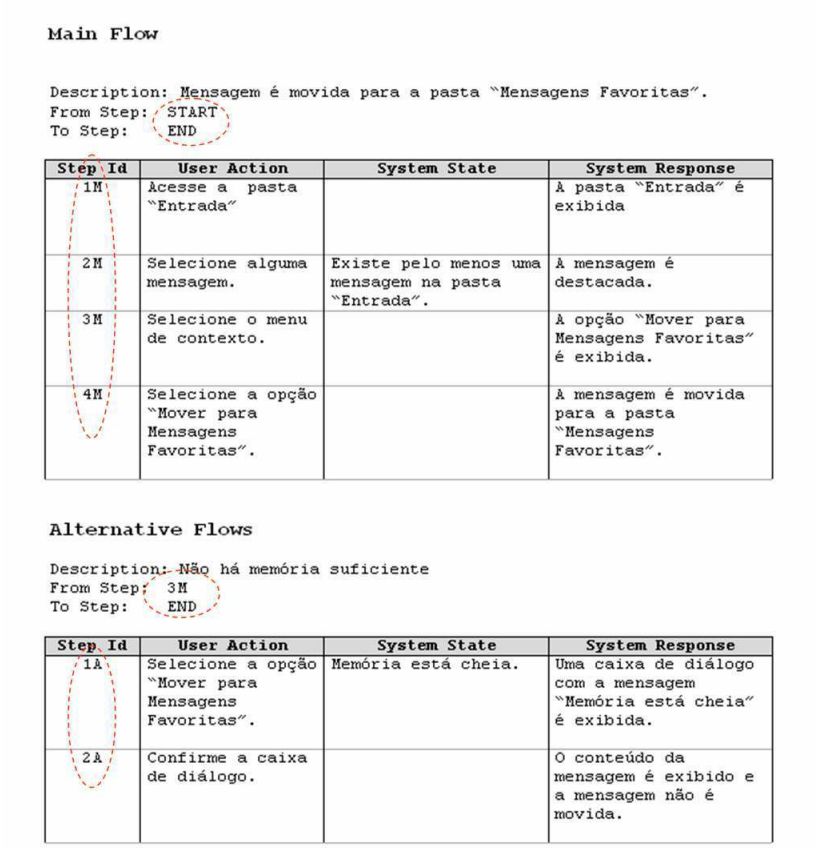
O caso de uso identifica um objetivo principal do sistema, enquanto cenários descrevem as possibilidades de resultados de uma tentativa de conseguir um objetivo de um caso de uso. Um cenário é um caminho lógico único através de um caso de uso. UML mesmo define um cenário como uma instância de um caso de uso. Em outras palavras, o caso de uso define o que poderia acontecer, e um cenário define o que acontecerá sob um determinado conjunto de condições.

Os casos de uso abstraem situações internas e modelam apenas partes do comportamento do sistema visíveis a um ator. Isto se aplica ainda que os casos possam descrever diferentes níveis de detalhe. Caso de uso elaborado em nível de empresa seria descrever interações com outras pessoas ou empresas, enquanto que no nível de componente poderia descrever as operações em uma classe, ou um conjunto de operações realizadas em seqüência. Independentemente do nível, o foco é sobre a finalidade e as interfaces da entidade, e não a execução. As seqüências de ações realizadas por um caso de uso são as interações com os atores, e não os processos internos. Ao definir os casos deste modo, o modelo define um conjunto de requisitos, e não uma solução. Não se deve descrever a forma como o sistema funciona. Descreve-se o que o sistema deve ser capaz de fazer.

A idéia de utilizar casos de uso (ou diagramas de seqüência) para a derivação de testes não é nova. Alguns dos muitos exemplos que poderíamos citar a respeito da geração de casos de teste baseados em cenários de caso de uso são [40; 14; 5; 51]. Estes trabalhos mostram a viabilidade e os ganhos com esta atividade. No nosso contexto, utilizando a ferramenta TaRGeT, cada caso de uso é especificado usando um *template*. O *template*(como pode ser visto na Figura 2.5) é formado por passos (*step*) identificados por um *id*. Um

passo é composto por uma ação do usuário (*user action*) e uma resposta do sistema (*system response*). Uma seqüencia de passos forma um fluxo de execução (*execution flow*). Cada fluxo possui um ponto inicial/final (*from/to step*). Os fluxos de execução são categorizados como principal (*main flow*), aternativo (*alternative flow*) e fluxo de exceção (*exception flow*).

O exemplo da Figura 2.5 corresponde à funcionalidade de mover uma mensagem contida na pasta de "caixa de entrada" para pasta de "mensagens favoritas". A execução é iniciada pelo passo 1M do fluxo principal *main flow*. Como indicado no *from/to step*, a execução segue com 2M, 3M, 4M (para execução com sucesso). Se o sistema estiver no passo 3M e o estado do sistema for "Memória está cheia", o fluxo corrente passa a ser o fluxo alternativo (*alternative flow*).



Main Flow

Description: Mensagem é movida para a pasta "Mensagens Favoritas".
From Step: START
To Step: END

Step Id	User Action	System State	System Response
1M	Acesse a pasta "Entrada"		A pasta "Entrada" é exibida
2M	Selecione alguma mensagem.	Existe pelo menos uma mensagem na pasta "Entrada".	A mensagem é destacada.
3M	Selecione o menu de contexto.		A opção "Mover para Mensagens Favoritas" é exibida.
4M	Selecione a opção "Mover para Mensagens Favoritas".		A mensagem é movida para a pasta "Mensagens Favoritas".

Alternative Flows

Description: Não há memória suficiente
From Step: 3M
To Step: END

Step Id	User Action	System State	System Response
1A	Selecione a opção "Mover para Mensagens Favoritas".	Memória está cheia.	Uma caixa de diálogo com a mensagem "Memória está cheia" é exibida.
2A	Confirme a caixa de diálogo.		O conteúdo da mensagem é exibido e a mensagem não é movida.

Figura 2.5: *Exemplo de template de caso de uso*

Um fluxo possui execução seqüencial. O ponto de partida da execução reside no fluxo principal (*main flow*) iniciado pelo ponto indicado em *from state* que define a origem do

passo inicial. Se o passo inicial não depender de outro fluxo, a palavra START é utilizada significando que o primeiro passo corresponde ao primeiro passo do fluxo em execução. Caso o passo inicial do fluxo depender de outro fluxo, é utilizado o *id* do passo do qual ele depende no *from step*. O término da execução do fluxo é indicado pelo *to step*, que tanto pode ser um passo de um fluxo diferente do corrente indicado pelo *id*, como a palavra END indicando execução do fluxo até o final da seqüencia.

O que define a existência de fluxos de exceção e alternativos é o campo que define o estado do sistema (*system state*). Uma ação do usuário, dependendo do estado do sistema, pode continuar no fluxo corrente ou ser transferido a outro fluxo.

O exemplo utilizado corresponde ao *template* na visão do usuário. Os casos de uso podem ser especificados na visão de componentes. Nesse modelo, o sistema é decomposto em componentes que, concorrentemente processam requisições do usuário e se comunicam entre si. A sintaxe é semelhante ao explicado para visão do usuário e valem as mesmas definições utilizadas. O que muda é que, na visão do componente, é necessário definir o componente que está invocando uma ação, bem como aquele que está fornecendo o serviço.

2.4.1 Relação entre casos de uso

A adição de mais detalhes por vezes revela que a estrutura original do modelo pode não ter sido ideal, ou pode mesmo ter sido incorreta. Um tipo comum de mudança envolve uma divisão de um caso de uso em dois ou mais casos uso, que é necessário quando ocorre de fluxos alternativos serem completamente diferentes em vez de uma variação do fluxo principal, portanto, vão ser melhor expressos como casos separados. Em outras ocasiões, dois casos de uso serão substituídos por um único, por exemplo, quando a descrição detalhada dos fluxos mostra que eles são muito semelhantes e, portanto, é melhor tratá-los como variação de caso de uso concreto. UML define três relações entre os casos de uso:

- **Relação *include*:** no processo de identificar e descrever os casos de uso, pode ocorrer a identificação do mesmo sub-fluxo em dois ou mais casos de uso. Na intenção de assegurar que a parte comum do fluxo permanece idêntica e minimizar a manutenção das descrições, utiliza-se a relação *include*. Nesta relação, um caso de uso se utiliza da execução de outro caso de uso como parte de sua própria execução. O caso de

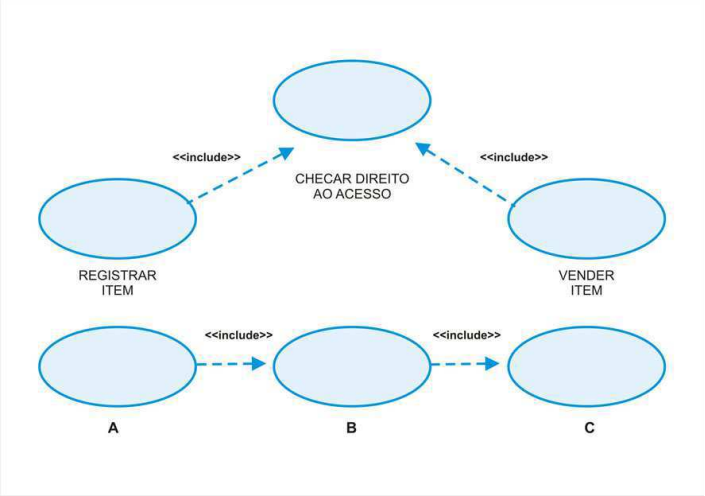


Figura 2.6: Exemplo de relação *include*

uso incluído pode ser caracterizado por concreto quando faz sentido ser executado separadamente ou por abstrato quando ocorre apenas para se tornar um fluxo comum. Outro uso menos comum da relação de *include* verifica-se no caso de separar um fluxo de caso de uso prevendo que ele será utilizado posteriormente por outros casos de uso em outras versões do sistema ou ainda para enfatizar o fluxo. Na Figura 2.6 temos que os casos de uso *Registrar Item* e *Vender Item* incluem o caso de uso *Checar Direito ao Acesso*. A seta indica, graficamente, a relação de dependência: quem inclui o caso de uso não tem funcionamento completo isolado e, portanto, fica do lado oposto à ponta da seta. A relação possui outra característica: a transitividade. No outro exemplo da mesma Figura temos que A inclui B que inclui C, resultando em A inclui C.

- **Relação *extend*** ocorre quando o fluxo de um caso de uso é estendido pelo fluxo de outro caso de uso. A idéia é que sejam adicionadas funcionalidades não existentes em versões anteriores do sistema. A relação pode ser utilizada também quando os fluxos não possuem coerência semântica (um caso de uso que estenda a funcionalidade de *log* para um caso de uso de compras). A extensão adiciona funcionalidade no sentido de que o caso de uso estendido continua tendo uma execução completa e coerente isoladamente. A localização exata na seqüência das ações onde a extensão será inserida é definida pelos chamados pontos de extensão. Um ponto de extensão consiste em um nome e uma condição que determina se ocorrerá a extensão. Na Figura 2.6,

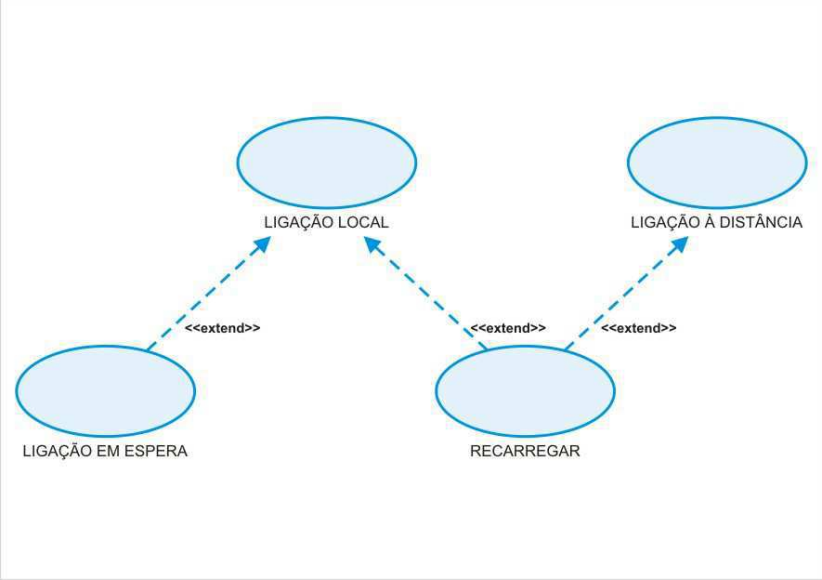


Figura 2.7: Exemplo de relação *extend*

as funcionalidades *Ligação Em Espera* e *Recarregar* adicionam ou estendem a funcionalidade de *Ligação Local*. *Ligação Local* tem funcionalidade completa sem as extensões, por isso a seta mostra que a dependência é da classe que estende e não da que é estendida. Da mesma forma ocorre com *Ligação à Distância* e *Recarregar*. Na Figura 2.8 temos um exemplo com ponto de extensão.

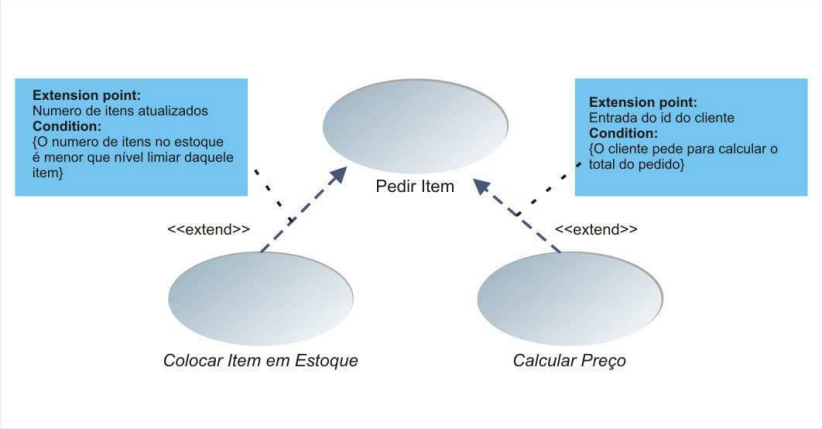


Figura 2.8: Exemplo de relação *extend* com *extension point*

- **Generalização entre casos de uso:** Pode ocorrer de casos de uso possuírem semelhanças quanto ao tipo de uso. Esta propriedade é resolvida pela idéia de generalização, semelhante ao que é feito com as linguagens de programação.

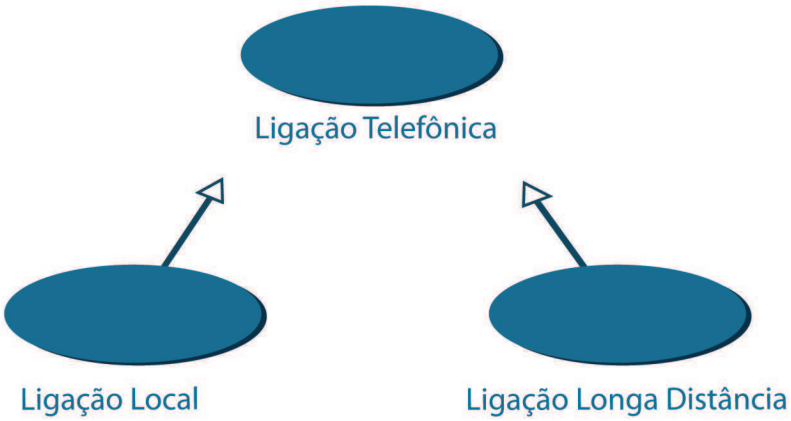


Figura 2.9: Exemplo de relação generalização

2.5 Padrões de teste

Padrões de Teste representam uma forma de reutilização no desenvolvimento de teste em que a essência de soluções e experiências recolhidas nas atividades de teste é extraída e documentada de modo a permitir a sua aplicação em contextos similares que possam surgir no futuro [30]. Os padrões fornecem meios para os testadores se concentrarem mais sobre o que testar e menos sobre a notação utilizada. Assim, esta prática simplifica o processo de desenvolvimento de teste, aumenta o nível de automatização e facilita o entendimento do teste. Embora o processo de passar pelos artefatos de teste existentes tentando identificar padrões para posterior reutilização possa parecer custoso à primeira vista, ele compensa em um longo prazo.

Diferentes áreas do processo de desenvolvimento de teste podem se beneficiar do uso de padrões. Utilizando a classificação de [42], os padrões de teste podem ser classificados através de três dimensões: **a fase de desenvolvimento do teste**, **o objetivo do teste** e **o escopo do teste**. A dimensão da fase de desenvolvimento se refere ao momento no processo de desenvolvimento de teste em que o padrão é usado (divide-se em **requisitos testáveis e especificação**, **definição do propósito de teste**, ***design***, **desenvolvimento**, **avaliação e execução**). A dimensão objetivo diz respeito ao tipo de teste (divide-se em **estático**, **estrutural**, **funcional** e **não-funcional**). E, por fim, a dimensão escopo caracteriza o nível do teste (divide-se em: **unidade**, **integração** e **sistema**).

Com esta classificação, é possível classificar e comparar melhor trabalhos em padrões

de teste. Os padrões de teste de Binder [13], por exemplo, se posicionam como padrões ocorrendo na fase de *design*. Binder introduz uma grande variedade de padrões, assim, toda a dimensão de escopo de teste está coberta. Uma vez que os padrões são destinados para sistemas orientados a objeto, eles fazem suposições e exigem o conhecimento do código. Assim, no que diz respeito à dimensão teste objetivo, esses padrões são classificadas como estruturais.

Padrões de teste podem ainda ser utilizados para descrever padrões de casos de teste para aplicações que possuem comportamento semelhante para determinado tipo de ações. É possível assumir que os passos que são executados para criar uma mensagem, por exemplo, são bastante semelhante em várias aplicações no contexto de celulares.

2.6 Processo de teste em aplicações de celulares

O processo de *design* de casos de teste começa pela especificação dos possíveis cenários de teste. Então, o *designer* de teste modela os vários cenários que são desejáveis para o teste. A depender do sistema em teste, o número de cenários pode ser bastante elevado. Aplicações para celulares são exemplos de contexto onde existem cenários de teste complexos. Freqüentemente, devido a uma variedade extensa de cenários, é decidido pelo *designer* dividir cenários grandes onde cada parte fica responsável pelo teste de uma funcionalidade especifica do SUT. Os cenários divididos são agrupados para compor uma *feature* de teste. Então, por exemplo, todos os cenários de teste relacionados à mensagem, que descrevem ações como enviar, receber mensagem, são agrupados numa *feature* mensagem. Desta maneira, várias *features* de teste podem ser especificadas, como agenda, *email*, multimídia.

Formalmente, uma *feature* descreve uma unidade identificável de funcionalidade e é representada por um conjunto de requisitos individuais[60]. As *features* podem ter requisitos que foram herdados de outras *features*, não sendo seus requisitos exclusivos dentro do contexto da aplicação, mas únicos apenas no contexto da *feature*.

Além das atividades de *design* e especificações, o processo de testes pode conter o processo de inspeção. O processo de inspeção valida se os casos de teste foram especificados corretamente. Basicamente, consiste na análise dos cenários de testes para verificar se os

casos de teste especificados abrangem efetivamente todo o cenário. Outras tarefas realizadas são verificar o padrão da escrita, a coerência da linguagem e outros.

Dentro no nosso contexto, existem duas maneiras de se realizar teste de *feature* que se distinguem pela execução do fluxo de controle do teste. As *features* podem ser testadas de maneira isolada. Ou pode ocorrer o teste simultâneo de diferentes *features*, conhecido por teste de interação. Neste tipo de teste, pode ocorrer tanto cenários onde as funcionalidades de uma feature dependem das funcionalidades de outra feature, como cenários onde há combinações de comportamentos independentes [20].

Figura 2.10: *Features*

Para exemplificar a idéia de *features* dentro do contexto de desenvolvimento de aplicações para celular, tomemos por base a Figura 2.10 onde temos três *features*: **Mensagem**, **Agenda** e **Ligação**. Como pode ser visto, cada feature possui suas funcionalidades que são definidas através de seus requisitos. A *feature* **Mensagem** tem as funcionalidades **enviar**, **receber** e **arquivar** mensagens, a *feature* **Agenda** possui as funcionalidades de **adicionar** e **remover** contatos e a *feature* **Ligação** possui as funcionalidades de **ligar**, **receber ligação** e **encerrar ligação**.

Ocorre dependência de funcionalidade entre *features* quando uma *feature* solicita um serviço ou dados de uma outra *feature*. Por exemplo, ao terminar de digitar uma mensagem, o usuário acessa a lista de contatos da agenda para selecionar um contato e enviar a mensagem. Esse cenário caracteriza uma interação onde a *feature* **Mensagem** depende funcionalmente da *feature* **Agenda**. Se usuário está compondo uma mensagem de texto e uma ligação chega a seu dispositivo, temos uma combinação de cenários independentes entre Mensagem

e Ligação. Nosso trabalho engloba apenas o primeiro caso de interação entre *features*.

A seguir explicaremos com um exemplo sucinto o papel e as dificuldades do testador neste ambiente.

Cenário

Suponha que temos a *feature Enviar Mensagem* com o requisito:

R1 O usuário pode enviar uma mensagem para um contato existente na agenda telefônica.

Ao receber o requisito da *feature*, o *test designer* inicia o processo de escrita de cenários de uso. Analisando os requisitos, ele percebe que serão escritos dois cenários:

- Criar um contato (envolvendo a *feature* Agenda)
- Enviar uma mensagem (envolvendo a *feature* Mensagem)

Temos uma interação entre duas *features*. Para escrever o cenário descrito é necessário um conhecimento do testador em relação à aplicação, mesmo levando em consideração um cenário tão simples. É preciso ter conhecimento suficiente para responder, por exemplo, quais as ações ou os passos necessários para realizar as duas ações principais. Como chegar à opção de menu que adiciona um contato? Quais os passos necessários para escrever uma mensagem e enviar a um contato adicionado? Esse tipo de informação não se encontra no documento de requisitos e para obtê-las, o *designer* de teste precisa ter conhecimento sobre o funcionamento da aplicação ou analisar outros documentos de requisitos, que descrevem outras *features* relacionadas. Estando a informação presente em outros documentos fica mais fácil contornar o problema. Mas e se depender do conhecimento do testador? Como o testador adquiriu tal conhecimento?

Testadores estão acostumados a escrever casos de teste. Esses casos de teste descrevem, de fato, ações que refletem o comportamento do *software*. Além disso, em um processo de teste convencional, os casos de teste são continuamente criados e modificados para se adaptarem aos comportamentos novos ou modificados das SUTs. Testadores estão freqüentemente especificando ou modificando os casos de teste. Assim, observou-se em situações reais de que muitas informações atualizadas sobre o SUT vêm de experiências de testadores. Testadores experientes conhecem bem a *feature* que estão testando, de modo que

acrescentam novas informações quando estão precisando de novos casos de teste ou estão modificando os casos de teste mais antigos.

Uma importante característica das aplicações para celulares é o de seu desenvolvimento se dar de maneira evolutiva. Essa evolução ocorre tanto em termos de requisitos, onde uma *feature* pode evoluir para uma outra *feature* através do acréscimo de requisitos, quanto em termos de implementação, onde o código que representa a *feature* é melhorado a medida que os ciclos de teste vão sendo executados. Exigindo, portanto, um conhecimento anteriormente testado, pois, a evolução do processo de desenvolvimento de *feature* por meio dos requisitos ocasiona vício de não repetição dos requisitos antigos, tornando-os implícitos ou desatualizados [39].

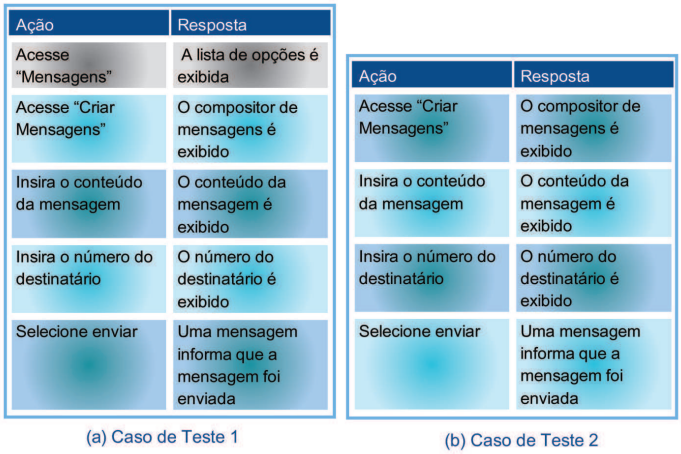


Figura 2.11: *Exemplo de casos de teste de* feature *com redundância sintática*

Ainda que as informações que englobam o cenário em teste estejam completamente contidas no documento de requisitos, existe ainda a necessidade de sutilezas interpretativas por parte *designer* de teste. Existe disparidade entre a linguagem utilizada para escrever o documento de requisitos e a linguagem utilizada para descrever os cenários de teste. Por exemplo, a partir de um requisito que contém *turn off speaker* será escrito um caso de teste que contém a ação *Press Speaker OFF softkey*.

Um caso de teste de *feature* trabalha no nível de requisitos utilizando a linguagem natural para a escrita. Este alto grau de liberdade acarreta em redundância. Dois casos de teste são ditos redundantes sintaticamente se apresentam uma mesma seqüência de passos. Por exemplo, na Figura 2.11, percebe-se que o caso de teste 2 está inserido no caso de teste 1,

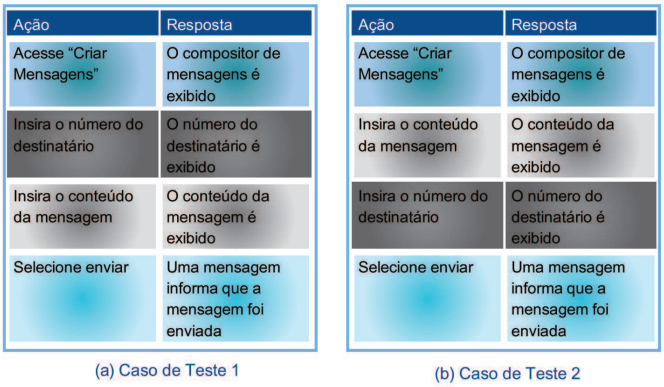


Figura 2.12: *Exemplo de casos de teste de* feature *com redundância semântica*

caracterizando uma redundância sintática. Dois casos de teste podem ainda ser redundantes semanticamente se apresentam uma mesma seqüência de passos ou uma seqüência de passos equivalentes, cuja ordem de alguns passos não caracteriza como diferentes os casos de teste. Na Figura 2.12, são apresentados dois exemplos de casos de teste que testam o envio de mensagens. Apesar de apresentarem uma seqüência de passos diferentes, os casos de teste são redundantes, pois a ordem de preenchimento do corpo da mensagem ou do destinatário não altera o resultado do teste. É preciso ter experiência suficiente da *feature* para visualizar quando dois casos são idênticos, bem como identificar quando a ordem de determinados passos é relevante ou não para o caso de teste.

Além das redundâncias, o mesmo cenário pode variar em termos da linguagem utilizada de acordo com o *test designer* que o escreve, por exemplo, usar a palavra ”Entre”ao invés de ”Acesse”. A diferença pode ainda estar na ordem em que foram construídas as frases.

2.7 Considerações Finais

Nosso intuito com este capítulo foi o de discorrer sobre temas a que fomos expostos de forma teórica e também prática para elaboração dos próximos capítulos contentores de nossa contribuição. Em primeiro lugar, fizemos uma explanação incorporando o processo de geração automática, apesar de estarmos restritos a resultados dentro do campo de Testes Baseados em Modelos. O motivo que nos levou a relatar tal abrangência foi o de estarmos inseridos num contexto amplo de teste de *software* e, portanto, conhecer este processo

Figura 2.13: Exemplo de LTS Anotado

de forma mais ampla; permite-nos entender melhor o contexto existente para ver a melhor maneira de incorporar uma abordagem relativamente nova no âmbito industrial. Os exemplos aqui mostrados resultam da junção entre o que estudamos na literatura e que vivenciamos na prática.

O outro assunto abordado são os diagramas de seqüência. Escolhemos este tipo de diagrama pelas vantagens inerentes a tais diagramas como legibilidade, além de já possuirmos uma linha que trabalha com estes modelos em nosso time. Nosso desafio foi de analisar a versão 2 para estes diagramas de forma que pudéssemos tirar vantagem de uma ou mais *features* para compor nossa solução.

Nosso contexto trata de Testes Baseados em Modelos onde os modelos representam casos de uso. Fizemos uma explanação breve sobre como tratamos casos de uso. Por fim, mostramos dois formalismo internos que utilizamos como representação intermediária de nossos casos de uso (LTS e LTS Anotado) e finalizamos nossa revisão com padrões de teste e como o processo de teste para aplicações de celulares que são o suporte principal do nosso estudo.

Capítulo 3

Padrões de especificação de caso de teste

Este capítulo tem por objetivo explicar o processo de catálogo de padrões no contexto de aplicações para celulares. Em primeiro lugar, será apresentado de forma geral a idéia e motivação da estratégia proposta. Em seguida será apresentada a estratégia de especificação do modelo de teste com auxílio de padrões. Será mostrado como ocorre a adição e atualizações dos padrões. Por fim, teremos um exemplo ilustrativo da estratégia.

3.1 Introdução

Na Seção 2.6 foi descrito, de forma resumida, como ocorre o processo de teste em aplicações para celulares. O processo de teste é feito através de um processo dispendioso de, em primeiro lugar, analisar um conjunto de documentos de requisitos para determinar que testes serão necessários. Os testes são então executados manualmente, revistos e corrigidos exigindo experiência de teste e em telefonia. Este processo é repetido toda vez que os aplicativos são alterados ou atualizados. Como lá descrito, aplicações para celulares são compostas por *features* que se atualizam rapidamente, tornando difícil para o processo de escrito de modelo manual acompanhar a evolução. Além do mais, a pressão do mercado está reduzindo o tempo para a criação de testes e aumentando a necessidade de novas *features*. Sem contar que mudança nos requisitos impõe modificações nos testes existentes diminuindo mais ainda o tempo existente para testes.

Como foi visto na Seção 2.1.5, no processo de testes baseado em modelo, o testador não escreve os casos de teste dos requisitos manualmente. No entanto, ele cria um modelo que

representa estes casos de teste de forma compacta. Logo, os mesmos problemas vistos na Seção 2.6 são refletidos no processo de construção do modelo de teste dentro do processo MBT. Na próxima seção, mostraremos a estratégia que definimos com objetivo de auxiliar na melhoria da elaboração do modelo de teste. O principal problema reside em experiência. Somente com o tempo é possível adquirir tal conhecimento. Nossa estratégia não pretende tornar o testador experiente, mas reutilizar conhecimento de forma explícita.

A estratégia apresentada a seguir, de modo geral, significa acrescentar à entrada de elaboração do modelo, além do documento de requisitos, casos de uso no formato de seqüências de ações pré-definidas as quais definem um padrão de comportamento de uma funcionalidade específica. O primeiro passo na obtenção de padrões de comportamento é a definição de um conjunto de ações reutilizáveis que podem ser usadas para descrever o comportamento de uma aplicação a partir de uma perspectiva de sistema. Por exemplo: *Wait For Call*, *Reject Call*, *Send Message*. Escolhido o conjunto de ações, o próximo passo é preciso definir os passos que compõem estas ações na perspectiva de sistema também. Para tornar estas ações as mais gerais possíveis é necessário haver pontos de variação onde é possível modificar para o caso específico. Alguns destes pontos incluem os dígitos a serem discados e recebidos, o conteúdo de uma mensagem ou mesmo o texto específico de um objeto de interface. Uma vez que as ações foram escolhidas e os seus passos foram descritos, é necessário que exista um método de conectá-los em conjunto para modelar o comportamento da aplicação.

3.2 Estratégia de escrita de modelo de teste

Na tentativa de amenizar estes problemas, adicionamos artefatos auxiliares de entradas à especificação para, juntamente com os requisitos, facilitar o processo de escrita do modelo de casos de uso. Uma visão geral de nossa estratégia pode ser vista na Figura 3.1. O processo segue a metodologia MBT: elaboração do modelo, geração dos casos de teste e execução.

Em primeiro lugar, é elaborado o modelo de teste tendo por artefatos os requisitos e os padrões de fluxo de caso de teste (representados por diagramas de seqüencia). Um padrão pode ser definido como uma *solução* reutilizável para um *problema* que ocorre em um determinado *contexto* de utilização[1]. Como a reutilização das idéias e do conhecimento se torna

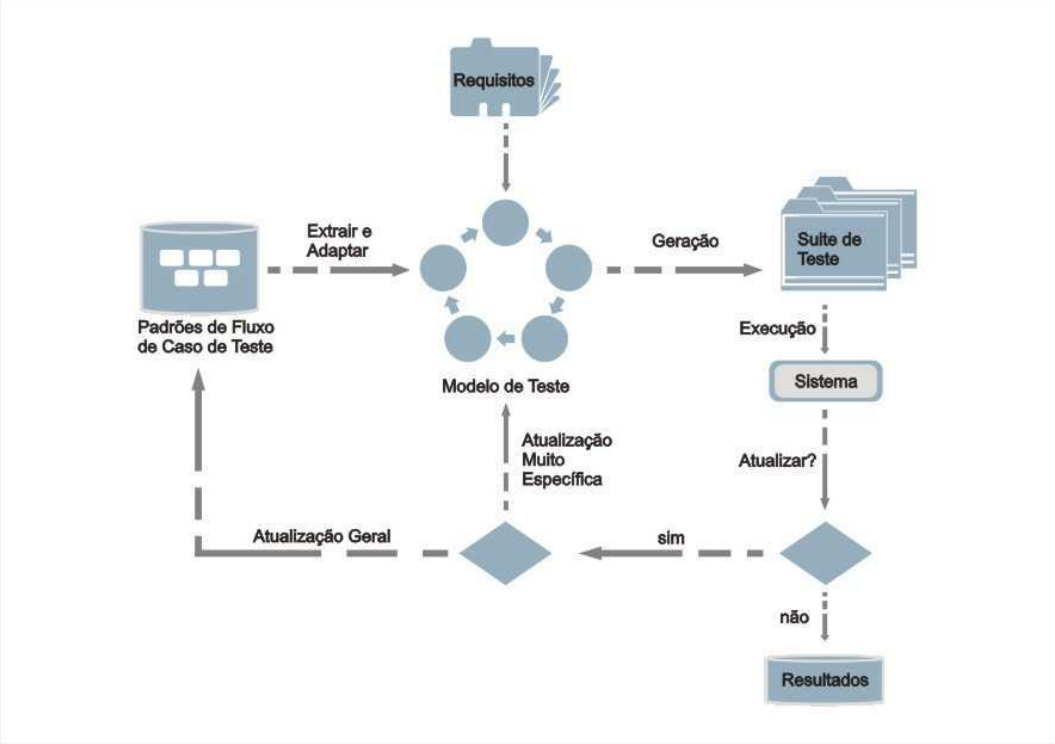


Figura 3.1: Visão Geral da abordagem de estratégia de escrita de modelo de testes com utilização de padrões.

mais e mais fundamental, um padrão pode ser uma forma eficaz de transmitir experiências sobre problemas recorrentes no desenvolvimento de *Software* e no domínio de desenvolvimento de Modelo de *Software*. Um padrão de caso de teste pode ser definido como uma parte do modelo que irá descrever o modelo de caso de uso completo.

Com os artefatos em mãos, o *designer* vai realizar a atividade de, alternadamente, fazer a leitura do documento de requisitos e verificar se existe algum padrão catalogado com ações que correspondam a algum cenário dos requisitos ou parte dele. Caso haja padrão correspondente, é preciso adaptá-lo com modificações necessárias ao contexto específico. Segue-se, então, com a geração dos casos de teste e posterior execução no sistema. Ao final do processo (fase de análise dos resultados), é preciso considerar possíveis atualizações na suite de testes. Idealmente, em metodologias baseadas em modelo, para geração de teste a modificação é realizada no modelo seguida da re-geração dos casos de teste. No nosso caso, deve-se verificar se a atualização é muito específica e, portanto, deve ser modificada no modelo gerado ou, caso contrário, deve ser modificada no padrão associado e em seguida atualizada no modelo de teste.

O repositório de padrões e as atividades relacionadas à aplicação de padrões no modelo de teste na nossa abordagem serão descritos a seguir.

Repositório

O repositório de padrões foi construído como um sistema de recuperação de informação [3] onde uma base de índices é criada a partir de itens selecionados, podendo ser posteriormente consultada pelos usuários usando palavras-chaves ou frases completas. Basicamente, temos como etapas principais (1) aquisição dos documentos que irão compor a base; (2) preparação dos documentos; (3) indexação dos documentos; (4) busca e (5) ordenação dos documentos recuperados. O principal objetivo de um sistema de recuperação de informação é facilitar o acesso a itens de informação relevantes às necessidades de informação do usuário.

No nosso caso, os documentos são os diagramas de seqüência representando um cenário abstrato de um comportamento padrão. Mais especificamente, os documentos são as mensagens que são trocas entre os lifelines. A entrada inicial é o diagrama salvo com todas as informações no formato XML. O arquivo XML contém todas as informações necessárias para recompor o modelo. Então, um *parser* extrai as informações relevantes do diagrama, ou seja, as mensagens. No final, os itens da base de índice são formados pelas mensagens do diagrama, seus respectivos nomes, mais o nome do diretório associado (aquisição dos documentos que irão compor a base).

As outras etapas foram assistidas pela API *Lucene*. Através da utilização do *Lucene* é possível indexar e buscar qualquer tipo de item que possa ser representado por meio de atributos em formato de texto. O *Lucene* oferece classes que realizam as etapas de: preparação dos documentos; indexação; busca; e ordenação dos resultados. A forma como ocorre a comunicação entre uma aplicação e o *Lucene* está resumido na Figura 3.2.

A preparação dos documentos (etapa 2) consiste em transformar as informações relevantes que foram extraídas no formato de documento da API *Lucene*. Os documentos do *Lucene* são a unidade de indexação e busca. Um documento é um conjunto de campos. Cada campo tem um nome e um valor textual. Um campo pode ser armazenado com o documento, caso em que é devolvido com *hits* de pesquisa sobre o documento. No nosso caso cada documento possui três campos: **diretório**, **mensagem** e **nome do arquivo**. Respectivamente, significam o diretório (*feature*) correspondente, as mensagens do diagrama e o nome

do arquivo XML onde o diagrama foi salvo.

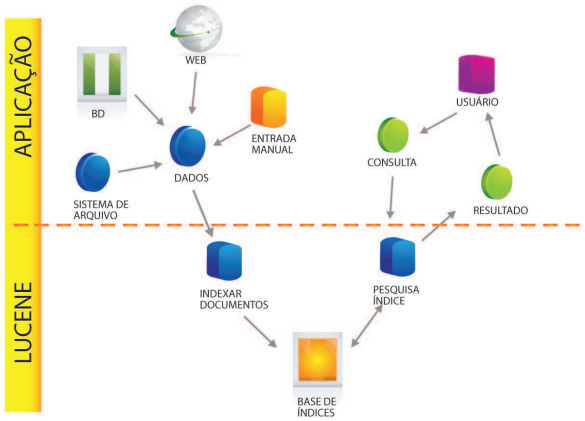


Figura 3.2: Integração do *Lucene* com as aplicações.

No processo de indexação dos documentos é primeiramente realizada uma análise pelo *Analyzer* do *Lucene*. O Analisador é responsável por tratar o texto que será indexado e por tratar o texto das consultas também. O processamento realizado por ele consiste em dividir o texto em palavras-chave, excluir as *stopwords* e realizar *stemming*. Depois é criado o repositório de índices.

A consulta ou recuperação de informação é realizada através de palavras ou frases e como resultado temos um conjunto ordenado de documentos que são relevantes para a consulta. As frases e as palavras-chave são montadas em uma consulta booleana com o operador booleano ”OU”de forma que possam ser recuperados itens que contenham pelo menos uma das palavras informadas na consulta. Como estamos interessados em procurar algum modelo que seja compatível com a aplicação específica que estamos testando, o domínio de palavras é bastante restrito.

A seguir definiremos as operações básicas realizadas sobre o repositório.

3.2.1 Utilização

A utilização ocorre pela busca no sistema de um modelo padrão que seja correspondente ao modelo de teste que se está construindo. Caso a busca seja realizada com sucesso, o usuário deverá incorporar o padrão e por fim adaptá-lo. Temos, portanto, três atividades:

- **Consulta:** a consulta é realizada através da sintaxe de consulta utilizada pela API *Lucene*. A busca é simples e feita por uma ou mais palavras chaves, e pode ser feita com ou sem filtro de diretório. O usuário pode consultar por ”NomeDoDiretório:consulta”, onde a resposta são todos os modelos daquele diretório que possuem em alguma de suas mensagens o texto da consulta, ou por ”consulta”onde a resposta são todos os modelos de todos os diretórios que possuem em alguma de suas mensagens o texto da consulta.

- **Incorporação:** a incorporação é feita através da utilização de *interaction use*, uma funcionalidade que acompanha os diagramas de seqüência da versão 2 de UML, cuja função é encapsular um diagrama em outro através da utilização de uma quadro com palavra *ref* no canto esquerdo e com o nome do diagrama referenciado no meio (Figura 3.3).

- **Adaptação:** os pontos variáveis são colocados em letra maiúscula nas mensagens e a adaptação consiste em modificar estas variáveis para o contexto específico.

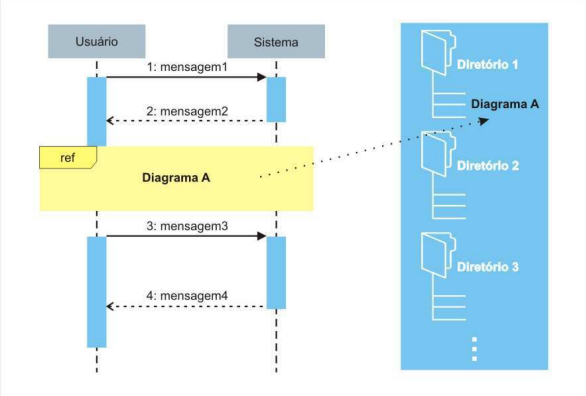


Figura 3.3: Exemplo de utilização

3.2.2 Adição de padrões

A adição de padrões ao catálogo acontece resumidamente em duas etapas descritas na Figura 3.4. A primeira etapa ocorre pela escrita do modelo com ações padrão em Diagrama de Seqüência UML 2.0. Em seguida é enviado o modelo para o diretório[1] correspondente no

[1]No nosso caso a divisão dos diretórios está em conformidade com a divisão das *features*

repositório de padrões.

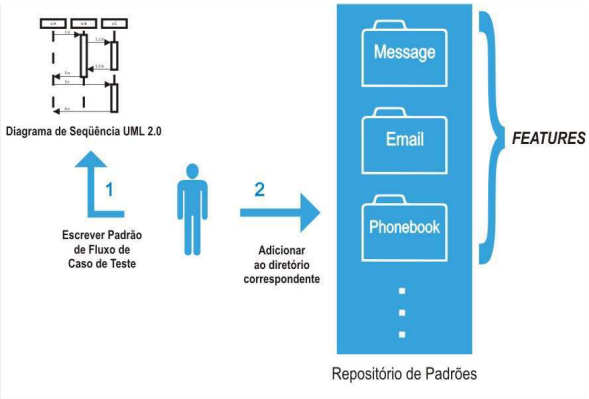


Figura 3.4: Adição de novos padrões

O processamento da entrada ocorre da forma descrita na Figura 3.5. Os passos a serem seguidos são:

1. O usuário escreve o diagrama de seqüência e salva no formato *EXtensible Markup Language*(XML). A ferramenta que utilizamos como suporte à escrita foi o *MagicDraw UML*[2].

2. O usuário envia, então, o arquivo XML e o diretório associado ao padrão para o repositório de padrões.

3. A ferramenta utiliza dois módulos para processar a entrada:

   - *XML Parser* onde é processado o arquivo XML para extrair as informações relevantes do modelo. Este módulo foi desenvolvido pela utilização da API *Digester* da *Apache*.[3]

   - *Index Engine* em que se utiliza a API *Lucene*[4] para fazer indexação de

[2]http://www.magicdraw.com/

[3]http://commons.apache.org/digester/. Basicamente, o pacote *Digester* permite configurar um módulo de mapeamento de um arquivo XML para um objeto Java através da execução de determinadas ações (regras) quando um padrão particular de elementos aninhados XML é reconhecido. Um rico conjunto de regras predefinidas está disponível para o uso, mas é possível criar as próprias regras.

[4]http://lucene.apache.org/. O *Apache Lucene*, ou simplesmente *Lucene*, é um *software* de busca e uma API de indexação de documentos, escrito na linguagem de programação Java. É um *software* de código aberto da *Apache Software Foundation* licenciado através da licença *Apache*.

informações extraídas do XML pelo módulo anterior tendo por referência o diretório escolhido pelo usuário.

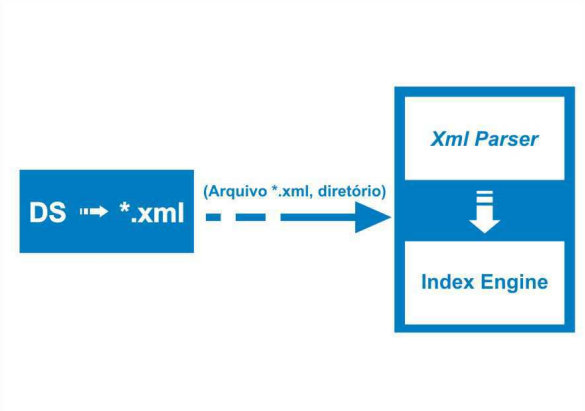


Figura 3.5: Processamento da Entrada

3.2.3 Atualização

É possível que o *test designer* ao criar testes para um aparelho novo, não possuindo o conhecimento suficiente do comportamento da aplicação, suponha algumas ações erradas a serem realizadas pelo usuário na intenção de cobrir o requisito. Este engano será percebido durante a execução, onde será necessária a modificação do caso de teste. Um percentual alto de casos de teste modificados pode ser um indicativo de que há problemas na especificação dos requisitos ou que o time que define o modelo está com dificuldades de entender os requisitos ou definir o modelo na notação adotada[22].

Um risco inerente a este processo de modificação de casos de teste é que a informação correta e atualizada do requisito esteja refletida em casos de teste específicos o que acarreta no espalhamento de soluções. Na nossa estratégia, esta atualização ocorre de duas maneiras:

- Se a atualização se mostra muito específica da aplicação que se está testando, a modificação ocorre normalmente como definido pelo processo MBT: no modelo de entrada.

- Se atualização é de tal forma que aparenta ser de caráter geral, a modificação deve ser realizada no modelo no catálogo de padrões no diretório correspondente e deverá, portanto, ser reenviado para ser reprocessado.

Assim, é preciso atualizar o modelo que envolve aquela funcionalidade e fazer a re-geração dos casos de teste.

3.3 Exemplo

O exemplo será ilustrando o cenário descrito anteriormente (Seção 2.6). Especificamente, temos os seguintes cenários:

- Colocar o contato de nome João e de telefone 3333 na agenda do telefone.
- Enviar uma mensagem de conteúdo "oi"para o contato recém-criado.

Os dois padrões associados estão descritos na Figura 3.7 e na Figura 3.6 respectivamente. O padrão de criar contato faz um fluxo padrão de atividades com as variações escritas em letra maiúscula. O padrão descrito pela funcionalidade de escrever mensagem segue o padrão para mensagens curtas, que são mensagens puramente textuais, e também possui os pontos variáveis descritos pela utilização do maiúsculo.

Os passos seguidos para compor o modelo são:

- Referenciar os dois padrões no modelo pela utilização de *"interaction use"*.
- Adaptar as variáveis do modelo padrão ao contexto específico.
- Compor as mensagens restantes do modelo.

O modelo resultante fica como ilustrado na Figura 3.9. As respectivas adaptações estão representadas na Figura 3.10 para a criação do contato João e na Figura 3.8 para a criação da mensagem simples de texto com conteúdo "oi". Perceba que algumas variáveis continuam sem alterações por opção do *designer* de teste de simplificar o modelo, por exemplo.

3.4 Conclusão

Um dos fatores que entra como vantagem inerente ao processo de testes baseados em modelo é o menor impacto de mudanças devido às mudanças nos requisitos. Na técnica, a mudança é refletida apenas no modelo ao invés de em todos os casos de teste que comporta o requisito.

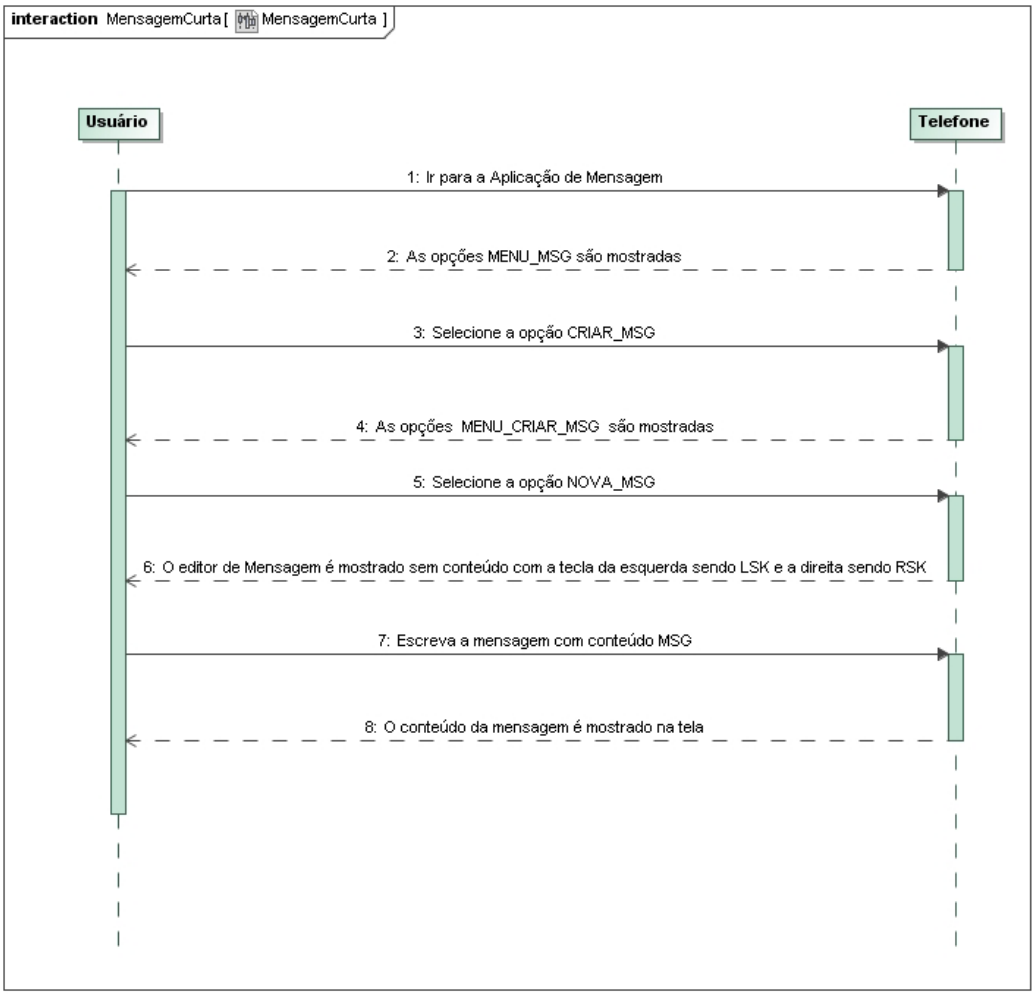


Figura 3.6: Padrão *Criar mensagem simples de texto*

No entanto, as abordagens não explicam como atualizar os seus modelos, qual é o esforço e o custo, ou a viabilidade de executar esta tarefa. Eles geralmente consideram que um novo modelo está pronto para re-gerar casos de teste. O que acontece se o modelo se torna imenso e complexo. Esta estratégia de modelagem com suporte de padrões, de certa forma, auxilia a modularização do modelo de teste. Implicando, portanto, numa melhoria do processo evolutivo do modelo.

É importante destacar, no entanto, que a manutenção do repositório fica sob a responsabilidade do testador. A formalização do padrão bem como a definição do que deve ser variável ou não dentro do contexto é realizado também pelo conhecimento do testador. Por isso, a etapa de atualização pode em primeiro momento ser bastante utilizada até que se tenha

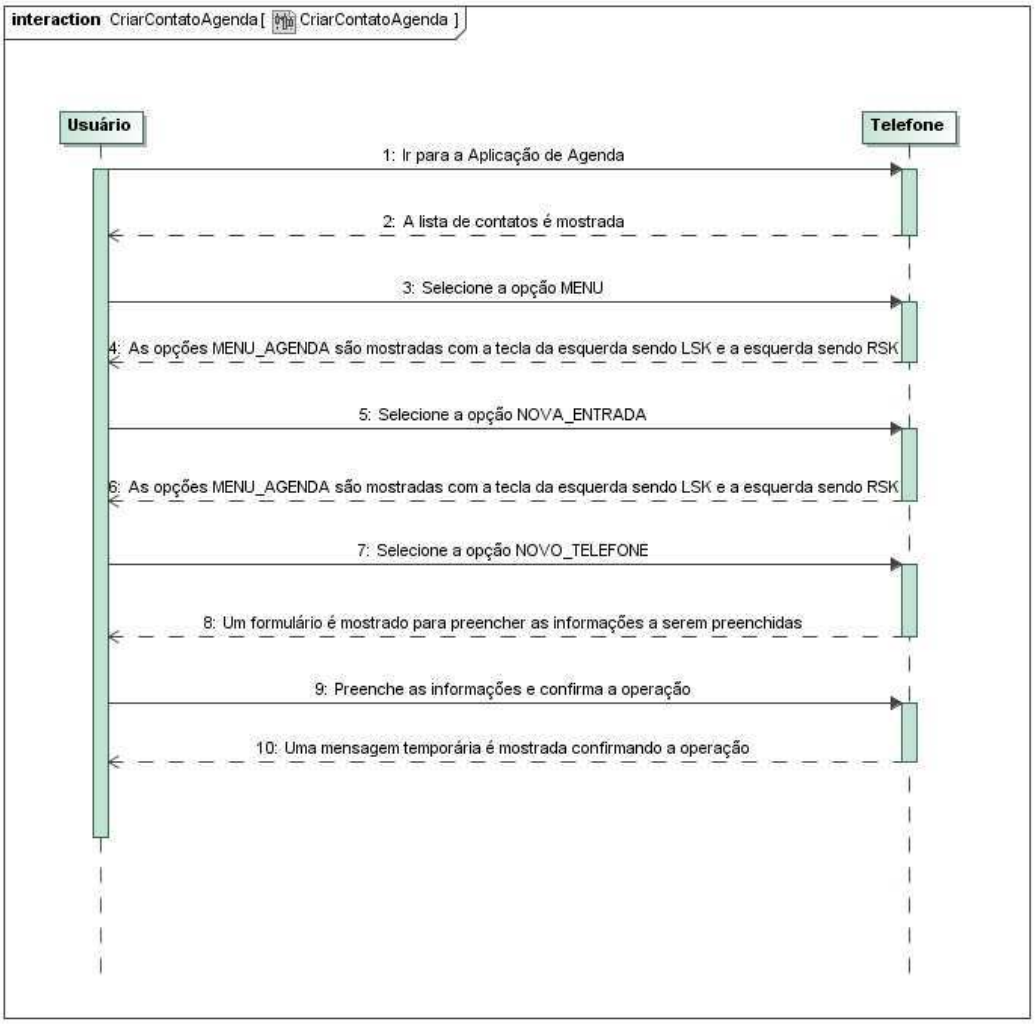


Figura 3.7: Padrão *Adicionar um novo contato para a Agenda*

experiência o suficiente para produzir bons modelos de padrão.

Além de simplificar a manutenção dos modelos, a utilização de padrões resultou na uniformidade da escrita dos casos de uso. O testador que conduziu a escrita do modelo se utilizando do repositório de padrões produziu um documento com escrita padronizada tanto na seqüência em que foram escritas as sentenças como na utilização de vocabulário restrito de verbos (ver Capítulo 5).

Uma das limitações decorrentes da utilização da API *Lucene* para manipulação da base de dados é atualização da base de dados, pois o *Lucene* não permite atualizações e para salvar as modificações realizadas, é necessário recriar a base de dados. Além disso, o *Lucene* apenas indexa informações que estão explícitas no documento. Assim, não é possível, por

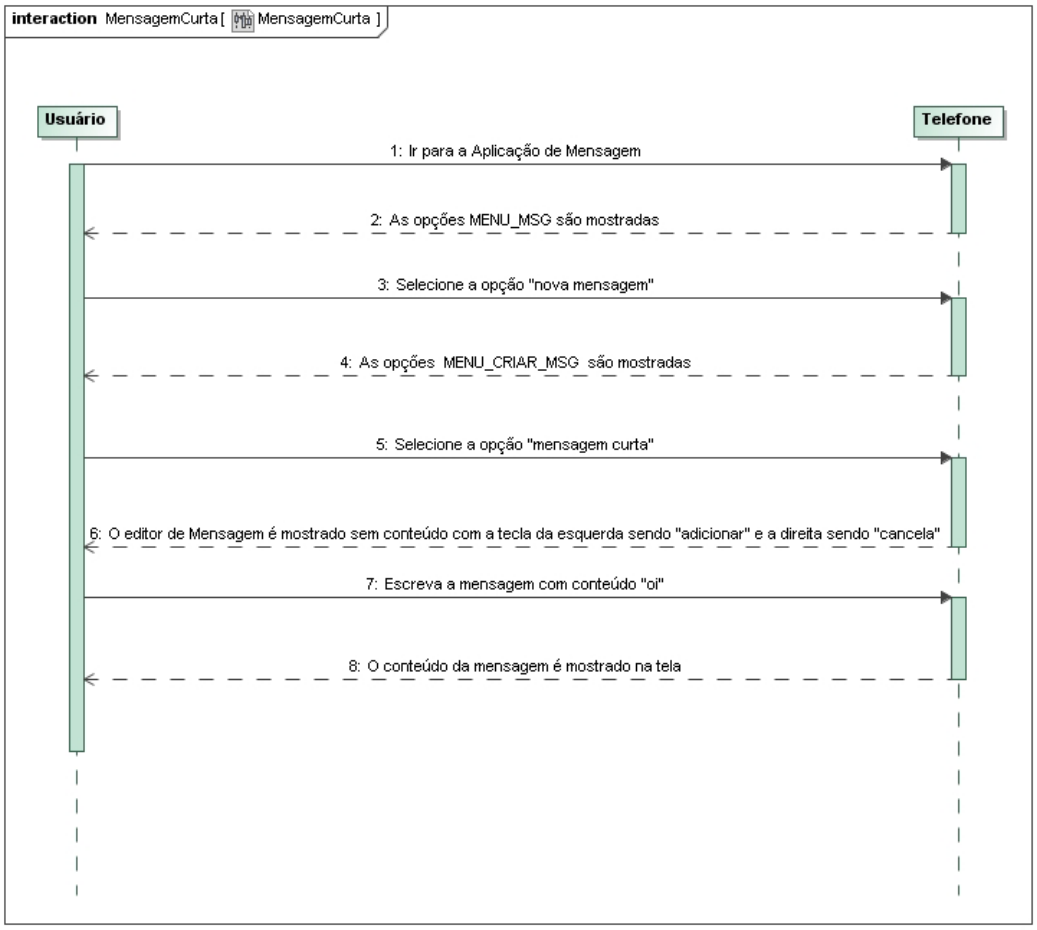


Figura 3.8: Padrão instanciado *Criar mensagem simples de texto*

exemplo, considerar a data de inclusão do padrão na base como atributo de indexação , a menos que esta data apareça explicitamente no padrão. Entretanto, através da utilização do *Lucene* é possível indexar e buscar qualquer tipo de item que possa ser representado por meio de atributos em formato de texto. O que permite que a estratégia possa ser utilizada em outros contextos onde seja possível tornar os requisitos domínio um padrão descrito em uma representação textual.

No tocante a requisitos de domínio, trabalhos com idéia de semelhante são os testes de família de produto. Estes podem ser divididos em testes do domínio, ou seja, testes genéricos ao domínio da família de produtos, e testes de aplicação, ou seja, testes para um produto específico de um determinado cliente. Testes de sistema do domínio compreendem principalmente a produção de artefatos de teste reutilizáveis. Testes de sistema de aplicação

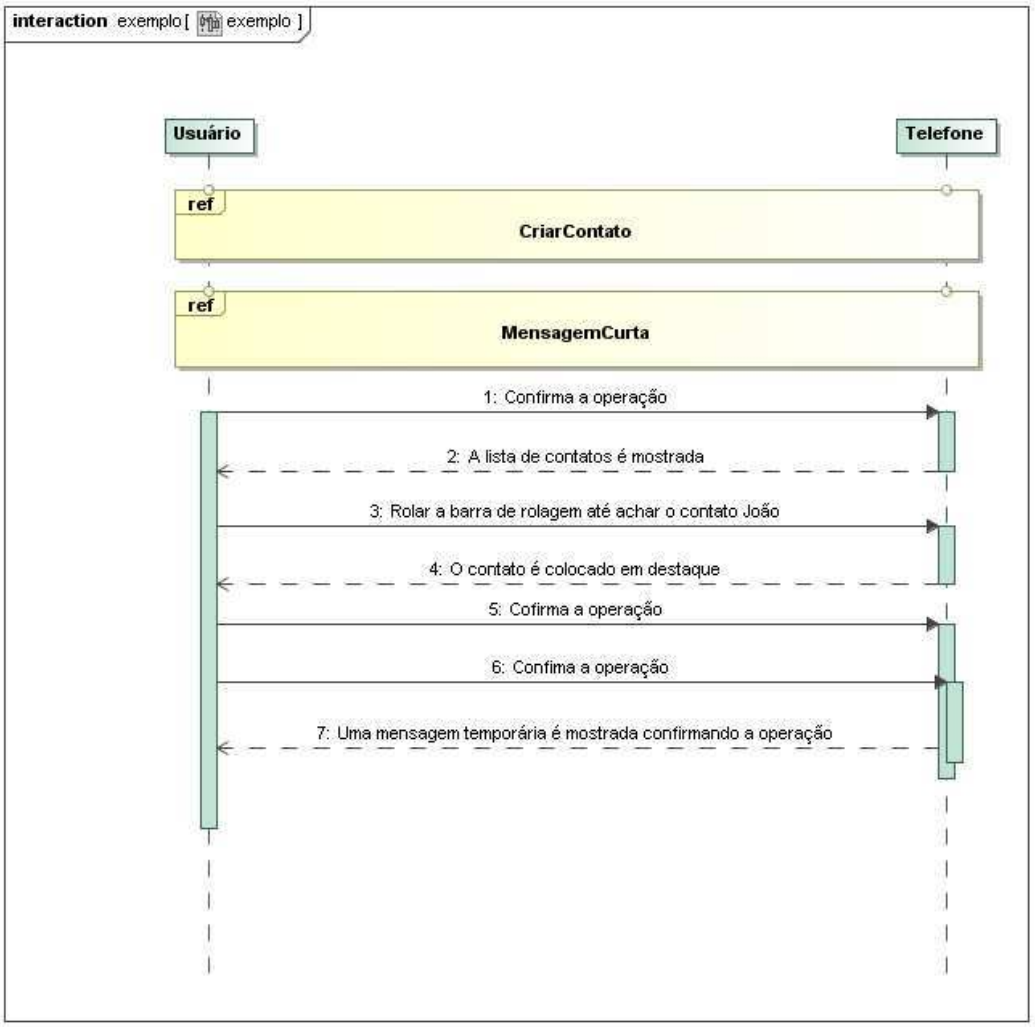


Figura 3.9: Exemplo

compreendem a reutilização e adaptação desses artefatos e o desenvolvimento de novos devido a requisitos específicos do cliente. Compartilhamos desta linha de trabalho pela idéia de reusabilidade. Nebut *et al* [41] adicionaram uma técnica à ferramenta TGV em que os casos teste para cada um dos diferentes produtos de uma linha de produtos são gerados a partir dos mesmos requisitos funcionais utilizando padrões de comportamento. Por outro lado, Kamsties *et al.* [32] propuseram uma abordagem para derivar casos de teste de sistema do domínio e testes de aplicação através da utilização de casos de uso que contêm variabilidades. Por fim, Bertolino e Gnesi [10] propõem uma metodologia para gerenciar o processo de teste de linhas de produtos baseado no método de categoria de partição. O ponto onde nos diferenciamos das abordagens acima descritas é que propomos pequenos modelos de

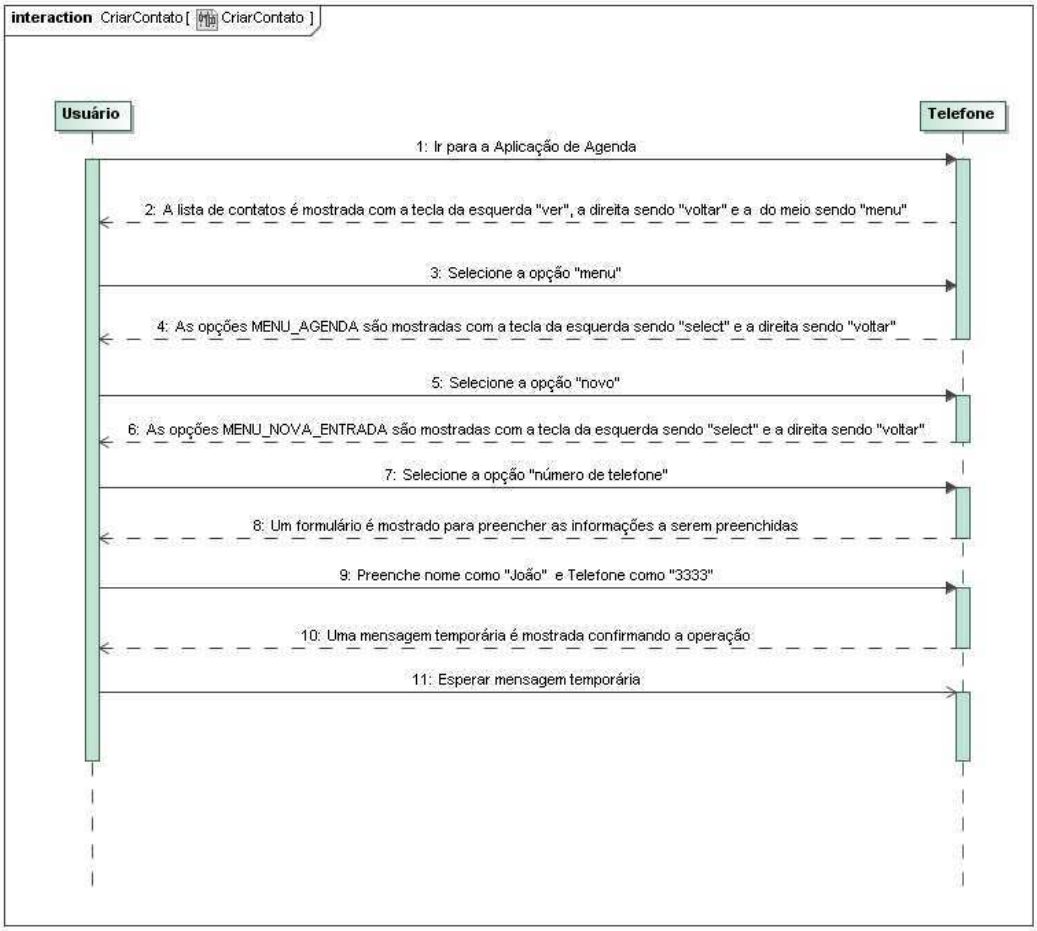


Figura 3.10: Padrão instanciado *Adicionar um novo contato para a Agenda*

uso comum enquanto que nas abordagens de linha de produto é construído um modelo geral contendo tudo que é similar entre os produtos e dele são retirados apenas o que é específico.

Capítulo 4

Procedimento de Testes Baseados em Modelos

No capítulo anterior, descrevemos uma estratégia de utilização de padrões baseados em fluxos de casos de teste como ferramenta de auxílio ao processo de escrita do modelo de teste. Descrevemos a composição do modelo abstraindo o processo de geração de casos de teste. Apresentaremos aqui o procedimento de Testes Baseados em Modelos de forma completa. Adicionaremos à ferramenta LTS-BT, a qual possui procedimento para geração e seleção de casos de teste, o modelo de teste resultante. Nesta ferramenta, o modelo de caso de uso é transformado em uma representação formal (LTS Anotado). O procedimento sistemático da ferramenta utiliza o LTS Anotado como estrutura interna para os algoritmos de geração e seleção de casos de teste. Em primeiro lugar, iremos descrever brevemente LTS-BT. Em seguida, explicaremos a estratégia que utilizamos para descrever casos de uso, que está dividida em duas seções: na primeira seção explicaremos nossa abordagem sintática para cobrir a semântica de relações existentes entre casos de uso, na segunda aproveitamos nossa abordagem de padrões de casos de teste explicada no capítulo anterior para descrever a composição de cenários detentores de pouco ou muito detalhe da aplicação.

4.1 Geração e Seleção de Casos de Teste com LTS-BT

LTS-BT (Labelled Transition System-Based Testing) [16] é uma ferramenta para geração e seleção de casos de teste funcionais inicialmente proposta para o contexto de aplicações para

celulares. LTB-BT foi desenvolvida para dar suporte ao teste de *feature*, onde os casos de teste são gerados e selecionados a partir de um LTS Anotado. As anotações utilizadas na ferramenta são: **Passos**, **Condições Iniciais** e **Resultados Esperados** . Estas informações são equivalentes, respectivamente, aos campos "*User Action*", "*System State*"e "*System Response*"do *template* apresentado na Seção 2.4. Seja $L =< Q, R, T, q_0 >$ um LTS Anotado, temos (ver Seção 2.3.1):

- Q é um conjunto finito e não-vazio de estados;
- $R = A \cup N$ é um conjunto finito de rótulos, onde A é o conjunto finito de informações ("*User Action*", "*System State*"e "*System Response*"). N = {"Passos","Condições Iniciais", "Resultados Esperados"} é o conjunto finito de anotações;
- $T \subseteq Q \times R \times Q$ é a relação de transição;
- $q_0 \in Q$ é o estado inicial.

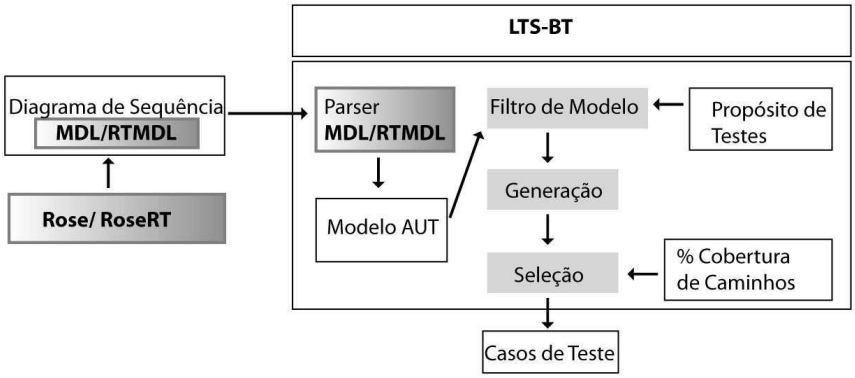


Figura 4.1: *Arquitetura da LTS-BT*

A arquitetura da ferramenta pode ser vista na Figura 4.1. Como pode ser percebido, as funcionalidades foram divididas em módulos.

- *Parser MDL/RTMDL* [1]: gera um grafo de fluxo dos diagramas de seqüência.
- *Filtro de Modelo*: utiliza o propósito de teste juntamente com o modelo para gerar novo modelo que seja um subconjunto do original guiado pelo propósito.

[1]MDL/RTMDL é o padrão para diagramas de seqüência UML

- *Geração*: obtém os casos de teste do modelo LTS.

- *Seleção*: reduz o número de casos de teste em situações onde cobertura total é inviável e é pretendido cobrir o modelo tanto quanto possível. Para esta seleção, o usuário deve passar uma percentagem de cobertura como um critério de seleção.

Resumindo, a ferramenta LTS-BT recebe como entrada uma especificação do sistema, que pode ser tanto um LTS Anotado no formato TGF (*Trivial Graph Format*)[2] quanto um conjunto de diagramas de seqüencia; um propósito de teste e uma porcentagem de cobertura de caminhos. Na Figura 4.2 temos um exemplo de entrada utilizada pela ferramenta LTS-BT na qual o usuário liga para o número de telefone embutido em uma mensagem de texto recebida, quando está visualizando a mensagem. Neste caso de uso, são possíveis dois cenários: um onde existe apenas um número embutido na mensagem e outro onde existe mais de um número embutido. O LTS resultante pode ser visualizado na Figura 4.3.

Quanto à geração, cada caminho no LTS (partindo de um nó inicial até um nó final) é um caso de teste, e o processo de identificação de todos os caminhos é realizado através de um algoritmo de busca em profundidade no modelo (no exemplo, teríamos dois casos de teste). Vista a impossibilidade de execução exaustiva dos casos de teste gerados, o propósito de teste e a similaridade possuem papel seletivo no sentido de eleger para execução apenas os casos de teste a que foi dada prioridade. O propósito de teste possui notação gráfica bem simples e é definido através de uma seqüência de transições que resulta em uma especificação de um modelo. Basicamente é um caminho seqüencial de ações que se deseja cobrir no modelo (termina com a transição ”Aceitar”), ou que se deseja excluir do modelo (termina com a transição ”Rejeitar”). Por exemplo, para testar o cenário onde o usuário escolhe uma mensagem que possui apenas um número embutido, é suficiente escolher o propósito ”*;Message contains only 1 Phone Number;*;Aceitar”. A porcentagem de cobertura de caminhos é definida pelo grau de similaridade entre dois casos de teste, analisando de forma estrutural o modelo, onde o caso de teste que tem maior grau é eliminado.

[2]http://www.yworks.com/products/yfiles/doc/developers-guide/tgf.html

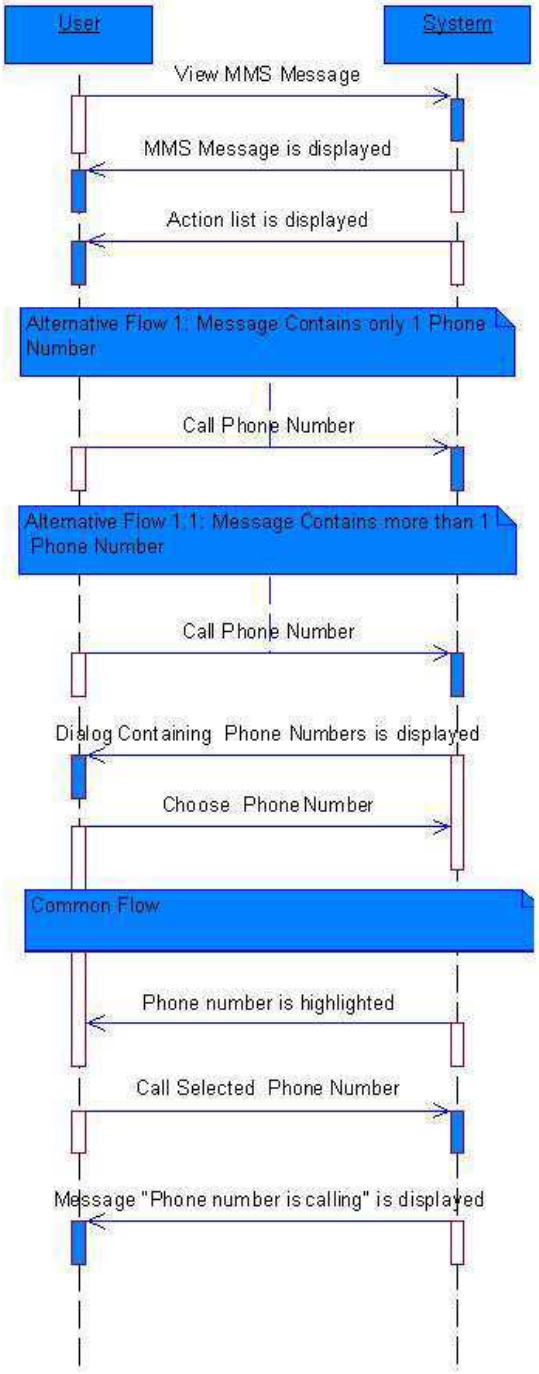


Figura 4.2: Ligar para o número de telefone embutido de uma mensagem.

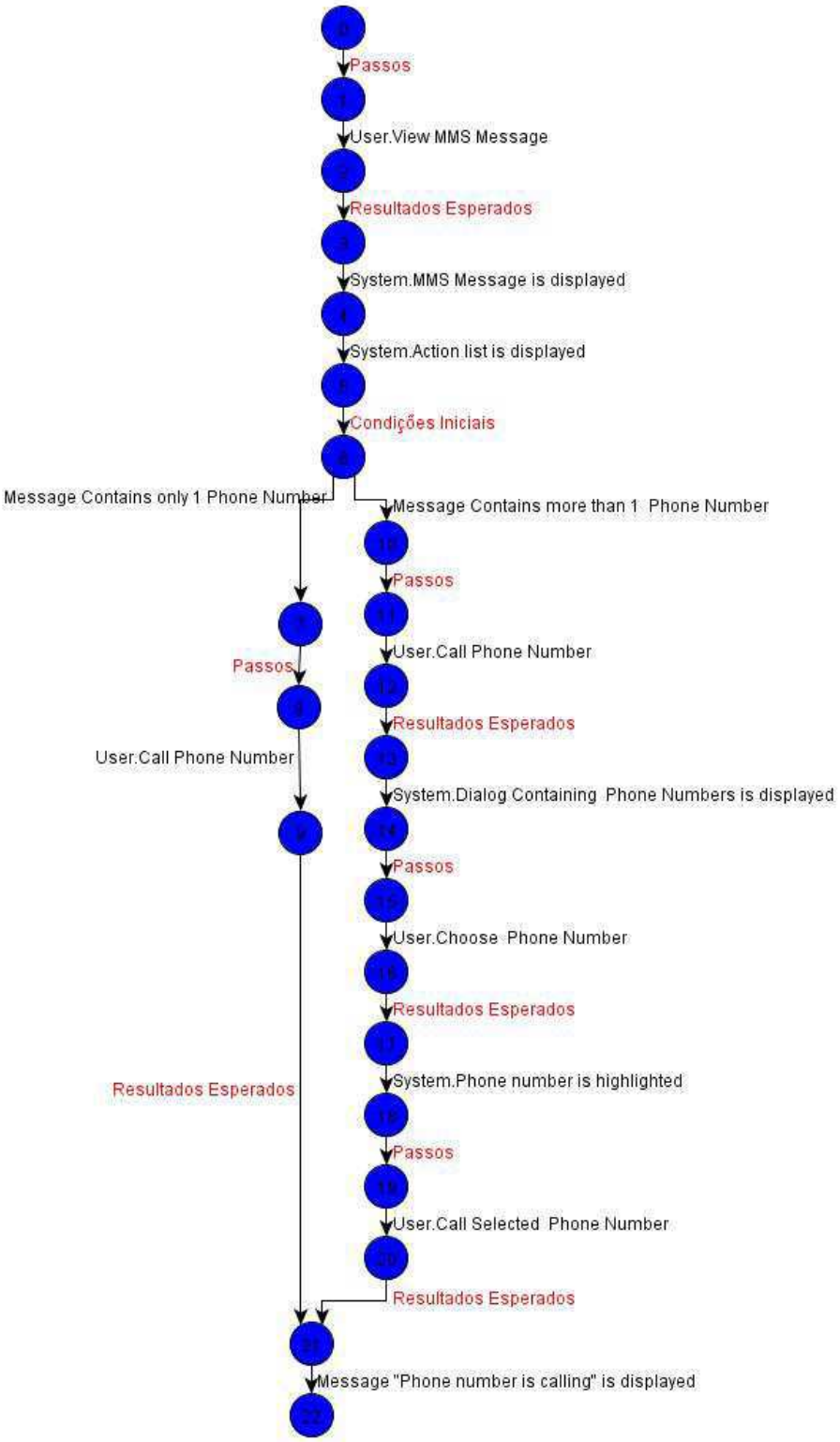


Figura 4.3: LTS resultante da Figura 4.2

4.2 Relações entre casos de uso

Em LTS-BT, cada diagrama de seqüencia representa um caso de uso completo, onde a variação entre os cenários ocorre através da verificação dos guardas ocorridos entre os fluxos alternativos, e a separação ocorre através de notas de comentários. Contudo, a relação entre casos de uso não aparece. Nesta seção, explicaremos a sintaxe que adotamos para a semântica das relações existentes entre os casos de uso, explicadas na Seção 2.4.1. A relação entre cenários ocorre da mesma forma, com a diferença que estamos lidando com a versão 2 do *framework* UML e, para tanto, utilizamos fragmentos combinados.

É importante destacar que a notação utilizada pelo template (Figura 2.5) é utilizada de forma genérica para as quatro notações que serão apresentadas aqui e serve tanto para relacionar fluxos entre cenários como para relacionar casos de uso. No entanto, a adição desta divisão explícita de relacionamento tem por objetivo tornar tanto a escrita como a leitura do modelo mais clara visualmente.

4.2.1 Relação de extensão

Uma extensão de um caso de uso, como visto na Seção 2.4, é um caso de uso que introduz um caminho alternativo não especificado no caso de uso base. Uma forma de representar extensões é através de pontos de extensão constituídos de um nome e uma referência do ponto onde ele deve ser inserido no caso de uso base. Uma possibilidade de representá-los é por meio de uma tupla [2]: $< after, before >$. Por exemplo, uma tupla $< 4, 9 >$ significa que o local onde o caso de uso será inserido no caso de uso base é após a mensagem 4 e antes da mensagem 9. Usando tuplas para representar pontos de extensões, podemos considerar os seguintes tipos de extensões [2]:

- *Same point*: $< after = before >$. A extensão quebra a seqüência principal, mas retorna ao mesmo ponto.
- *After that*: $after < before$. Ocorre uma quebra na seqüência normal.
- *Before that*: $after > before$. Ocorre uma quebra na seqüência e retorna a um passo anterior.

- *No return*: $< before = noreturn >$. Ocorre uma quebra na seqüência e não há retorno.

Representaremos estas relações em nosso modelo através da utilização do elemento ”comentário”com as informações representadas de modo geral, pela Figura 4.4(a) onde temos três elementos:

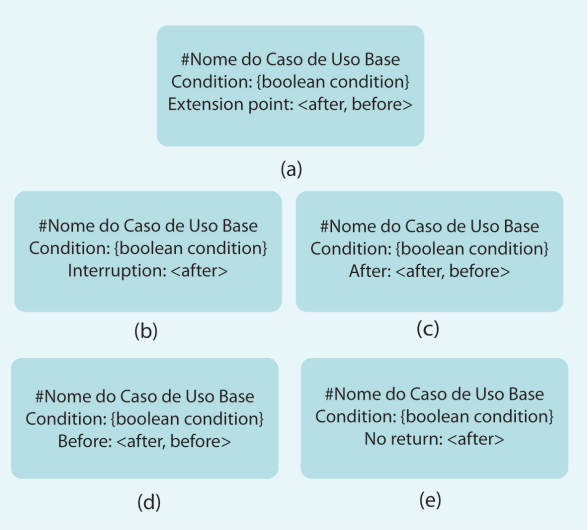


Figura 4.4: Relação entre casos de uso (pontos de extensão).

- ***Nome do Caso de Uso Base***: nome do diagrama onde será incorporada a extensão.
- ***Condition***: indica a condição para que o caso de uso seja incorporado.
- ***Extension point***: onde indicamos a localização onde ocorrerá a extensão. Utilizamos 4 nomenclaturas representativas dos tipos acima: *Interruption*(b), *After*(c), *Before*(d) e *No return*(e) para representar, respectivamente, *Same point*, *After that*, *Before that*, *No return*.

Nas Figuras 4.5 e 4.6 temos um exemplo de uso do ponto de extensão do tipo *Before*. Temos uma operação onde o Usuário usa o Sistema (Telefone) para ler a Caixa de Entrada da aplicação Mensagem e uma funcionalidade adicional é utilizada para restringir o acesso somente aos autorizados pela utilização do *Personal Identification Number* (PIN). O LTS resultante pode ser visto na Figura 4.7. No grafo resultante, observamos a possibilidade de dois caminhos: um onde não é necessário colocar o PIN (ou ele já é considerado válido), o

caminho do caso de uso base; e outro onde é necessário informar o PIN, caminho do caso de uso base estendido. Um caso de teste é um caminho no modelo LTS partindo do vértice inicial e terminando em um vértice final. Temos, portanto, dois casos de testes resultantes (Figura 4.8).

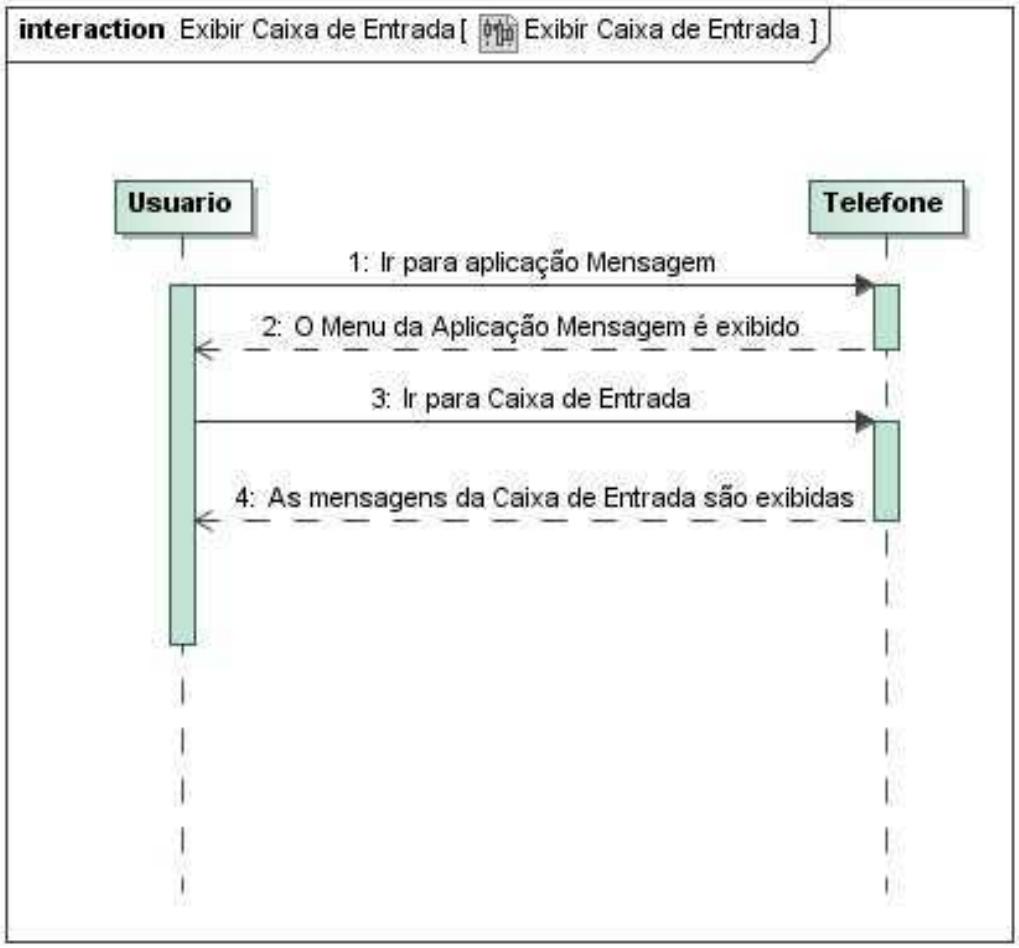


Figura 4.5: Caso de Uso: Exibir Caixa de Entrada.

De modo geral, temos que o comportamento do ponto de extensão para a tupla$<$ $after, before >$ ocorre da seguinte forma:

- *after*: indica que será inserida uma seta que vem daquele ponto do caso de uso base no início do caso de uso de extensão.
- *before*: indica que será inserida uma seta partindo da última mensagem do caso de uso de extensão até o ponto indicado (no caso de uso base).

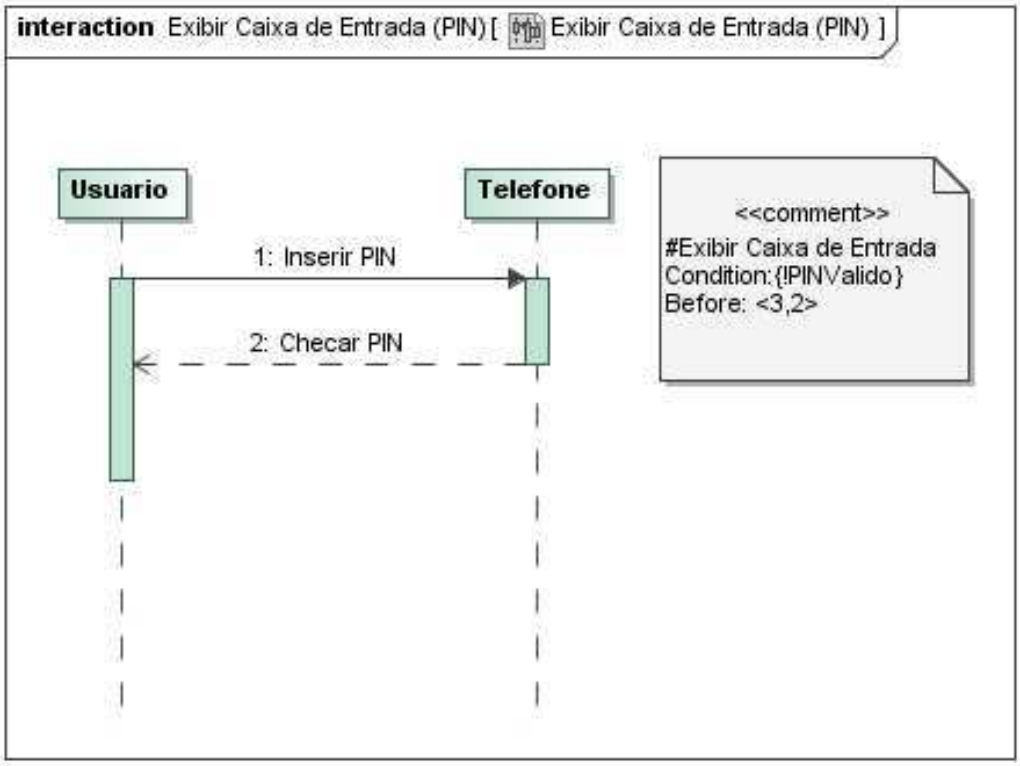


Figura 4.6: Caso de Uso: Inserir PIN.

Same point, representado por *Interruption*, representa o comportamento das interações dinâmicas entre *features* que ocorre quando quando uma determinada *feature* ou serviço de rede, que está em segundo plano, interrompe a execução de uma *feature* que está em primeiro plano, ou seja, a execução das *features* é alternada [20]. E, finalmente, *No return* termina a execução no final do caso de uso de extensão (ou seja, só utilizam *after*) e são, normalmente, utilizados para representar uma condição de exceção onde, devido à uma condição anormal, muda-se o fluxo de execução normal e termina a operação de forma diferente da prevista.

O procedimento utilizado para incorporar relações entre casos de uso é o seguinte: o LTS de cada caso de uso (correspondente a um diagrama de seqüência) é gerado separadamente e depois esses modelos são unidos. A união ocorre da seguinte forma:

- No ponto onde ocorre a interceptação no caso de uso base é criada uma transição "Condições Iniciais"(transição 7-8 da Figura 4.7), dado que o ponto de extensão está associado a uma condição. Então, a partir do vértice destino desta transição, são cri-

adas duas transições: uma com a condição associada ao ponto de extensão (transição 8-12 da Figura 4.7), de onde segue o fluxo associado ao caso de uso de extensão e, outra com a negação da condição (transição 8-9 da Figura 4.7), de onde segue o fluxo do caso de uso base.

- Caso exista retorno (Relação *After* e *Before*), colocar transição para o vértice de origem da transição que corresponde ao ponto definido como retorno.

- Rearranjar a numeração dos vértices.

4.2.2 Relação de inclusão

Outra relação existente entre os casos de uso é a inclusão (Seção 2.4). Um caso de uso incluído é uma parte do comportamento de um caso de uso base e, portanto, o caso de uso base não tem funcionalidade completa sem o caso de uso a ser incluído. A relação *include* apenas fragmenta o caso de uso. Representamos esta relação utilizando-se da funcionalidade *"interaction use"*(Seção 2.2).

4.3 Casos de uso em dois níveis de detalhes

Nesta seção utilizaremos o processo explicado no Capítulo 3 para estender o processo de geração de casos de teste da ferramenta LTS-BT para atender a dois tipos de perfis: testador experiente na aplicação em teste, onde a adição de detalhes nos casos de teste se torna desnecessária, e o testador inexperiente. Utilizamos os padrões extraídos para compor cenários em dois níveis de abstração. Para diferenciar de uma simples inclusão, colocaremos um símbolo de referência (#) na mensagem acima para identificar que logo abaixo teremos a mesma mensagem com mais detalhes.

Utilizando o exemplo da Seção 3.3 (com algumas modificações), temos o modelo representado pela Figura 4.9. As duas primeiras mensagens representam ações que são detalhadas pelos diagramas referenciados no *"interaction use"* logo abaixo delas. Se a opção de geração for sem detalhes, teremos que o algoritmo de geração de seleção casos de teste atuará na instância do LTS representado pela Figura 4.9. Caso contrário, teremos a estrutura completa com todas as ações dos diagramas "embutidos"substituindo as mensagens com #.

4.4 Conclusão

A abordagem utilizada para descrever e processar o modelo de teste foi incorporado ao processo sistemático da ferramenta LTS-BT, cujo procedimento foi estendido. Basicamente, adicionamos um *parser* para extrair as informações e modificamos algumas operações que manipulam o LTS. A apresentação dos casos de testes bem, como os procedimentos de seleção de casos de teste, continuam os mesmos, visto que a notação resultante manteve a estrutura do LTS Anotado utilizado pela ferramenta.

Como dito anteriormente, nosso modelo teste é elaborado de forma manual pela escrita de cenários utilizando a linguagem natural e, portanto, a inspeção realizada ocorre pela leitura do modelo de forma não-automática, geralmente, por uma pessoa diferente da que elaborou o modelo. A notação utilizada em 2.4 é uma forma consistente e prática para relacionar os componentes do modelo porque trata as relações entre cenários e entre casos de uso da mesma maneira tornando mais simples a manipulação automática . Porém, com esta notação, fica difícil verificar propriedades e relações entre os casos de uso se isto for realizado por um humano. A notação aqui proposta para definir relações entre casos de uso torna a legibilidade dos documentos muito melhor e ocasiona mais agilidade na fase de inspeção. Atentando para o fato de que um modelo pode ser fruto de várias pessoas, para continuar o trabalho deixado por outro, é necessário entendê-lo bem e a notação tem papel facilitador neste sentido. Resumindo, não adicionamos semântica às relações, apenas tornamos a notação mais explícita.

Em [34], Katara *et al* utiliza-se da restrição de domínio para propor uma metodologia em torno de uma linguagem de modelagem de domínio específico consistindo das chamadas palavras de ação e palavras-chave, que descrevem ações do usuário, com um elevado nível de abstração, bem como suas implementações de nível inferior, respectivamente. Em[51], Roubtsov e Heck utilizaram uma abordagem baseada em caso de uso para derivação de testes reproduzidos em escala industrial através da utilização de especificação de artefatos de teste em três níveis. Ao mais alto nível, cenários de teste são utilizados para validar casos de uso de sistema. Os níveis mais baixos são apresentados pelos scripts de teste e casos de teste. Relacionamos-nos a estes trabalhos pela utilização de artefatos contentores de pouco e muito detalhes.

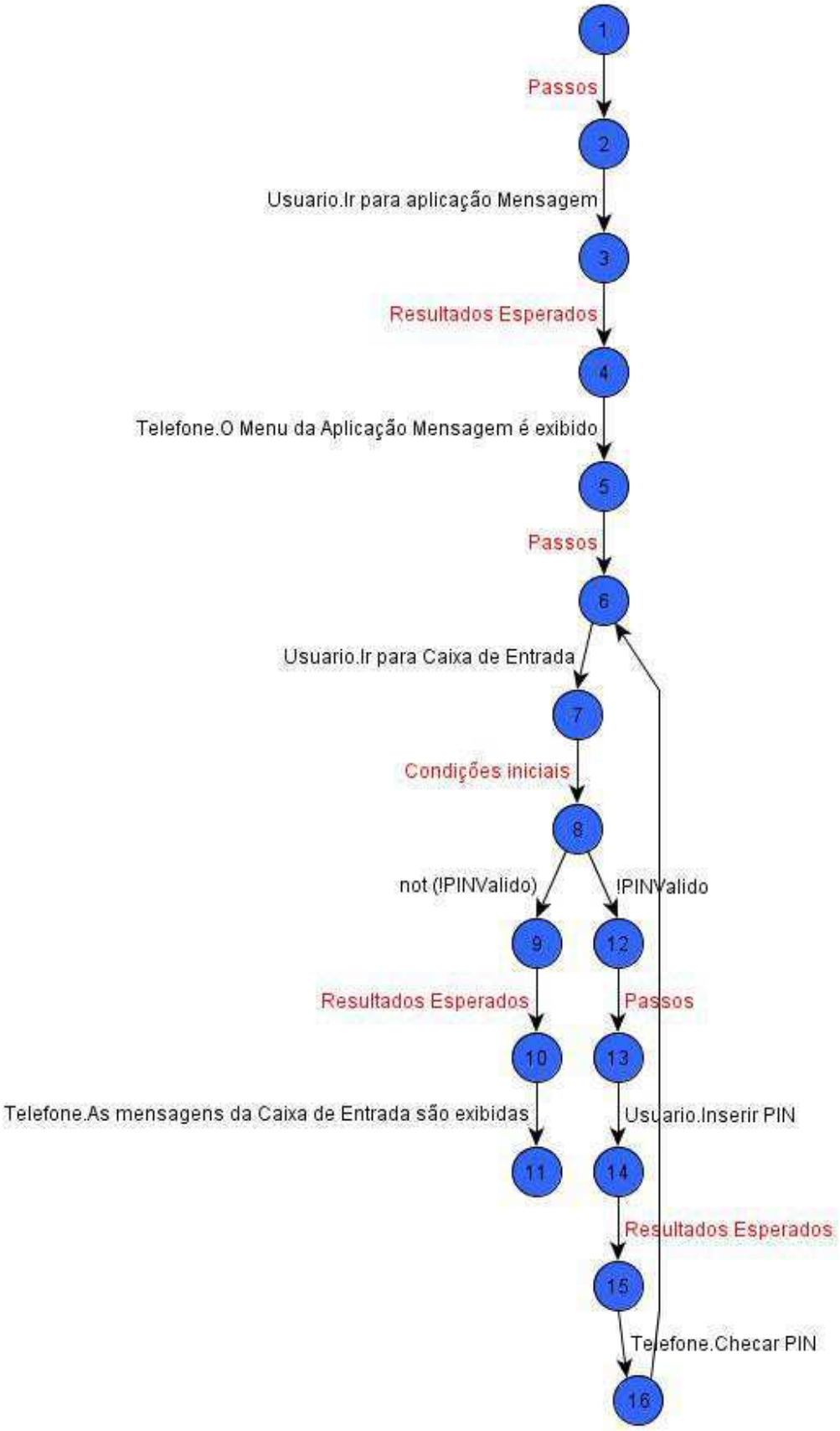


Figura 4.7: LTS resultante das Figuras 4.5 e 4.6

```
conditions: not (!PINValido)

 | Steps                             | Expected Results                                        |
  -------------------------------------------------------------------------------------------
 | Usuario.Ir para aplicação Mensagem | Telefone.O Menu da Aplicação mensagem é exibido         |
  -------------------------------------------------------------------------------------------
 | Usuario.Ir para Caixa de Entrada   | Telefone.As mensagens da Caixa de Entrada são exibidas |
  -------------------------------------------------------------------------------------------

conditions: (!PINValido)  (Loop Condition)
            not (!PINValido)

 | Steps                             | Expected Results                                        |
  -------------------------------------------------------------------------------------------
 | Usuario.Ir para aplicação Mensagem | Telefone.O Menu da Aplicação mensagem é exibido         |
  -------------------------------------------------------------------------------------------
 | Usuario.Ir para Caixa de Entrada   |                                                         |
  -------------------------------------------------------------------------------------------
 |                                   loop                                                      |
 | Usuario.Inserir PIN               | Telefone.Checar PIN                                     |
 |                               end of loop                                                   |
 |                                   | Telefone.As mensagens da Caixa de Entrada são exibidas |
  -------------------------------------------------------------------------------------------
```

Figura 4.8: Casos de testes resultantes da Figura 4.7

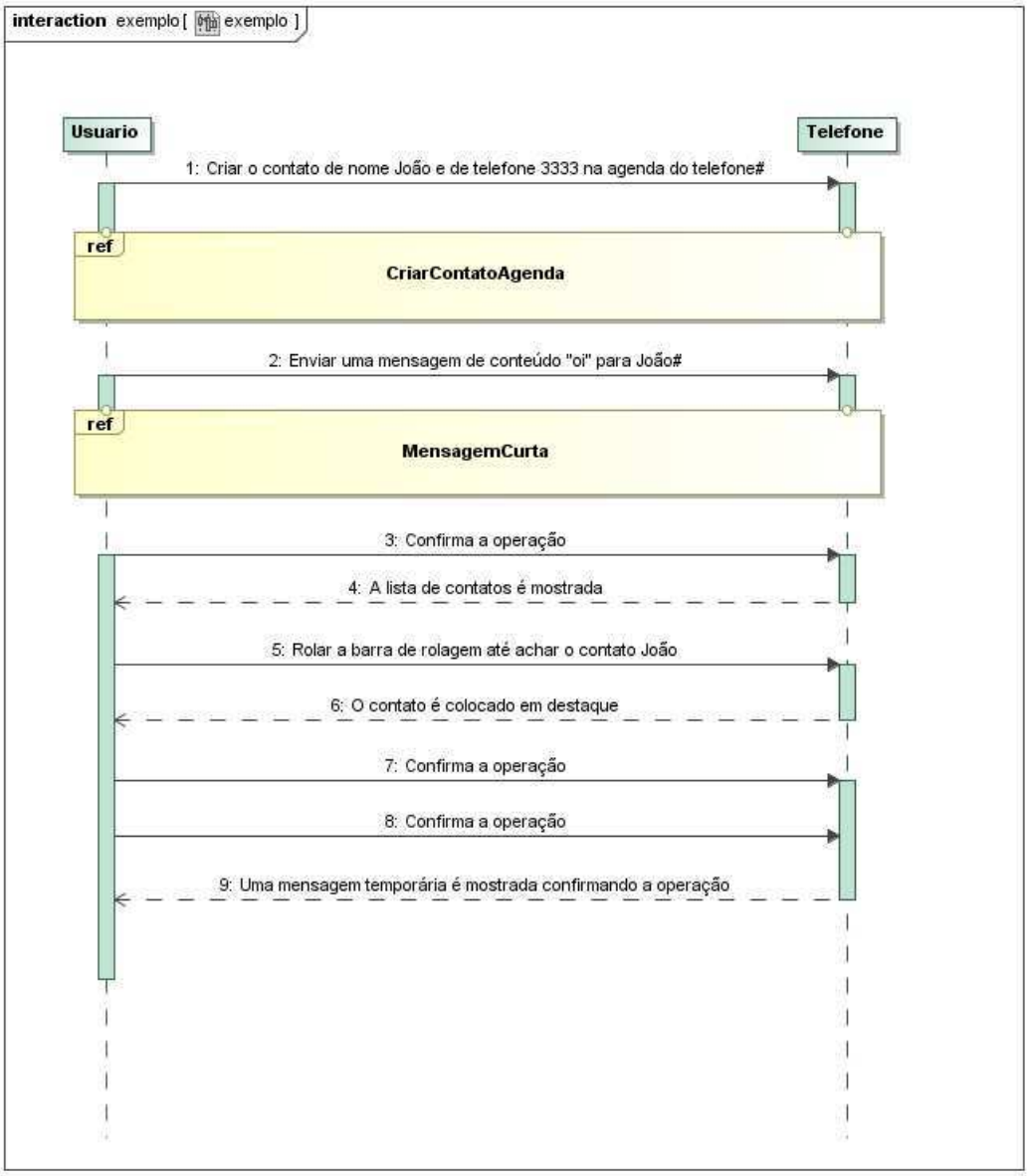


Figura 4.9: Exemplo

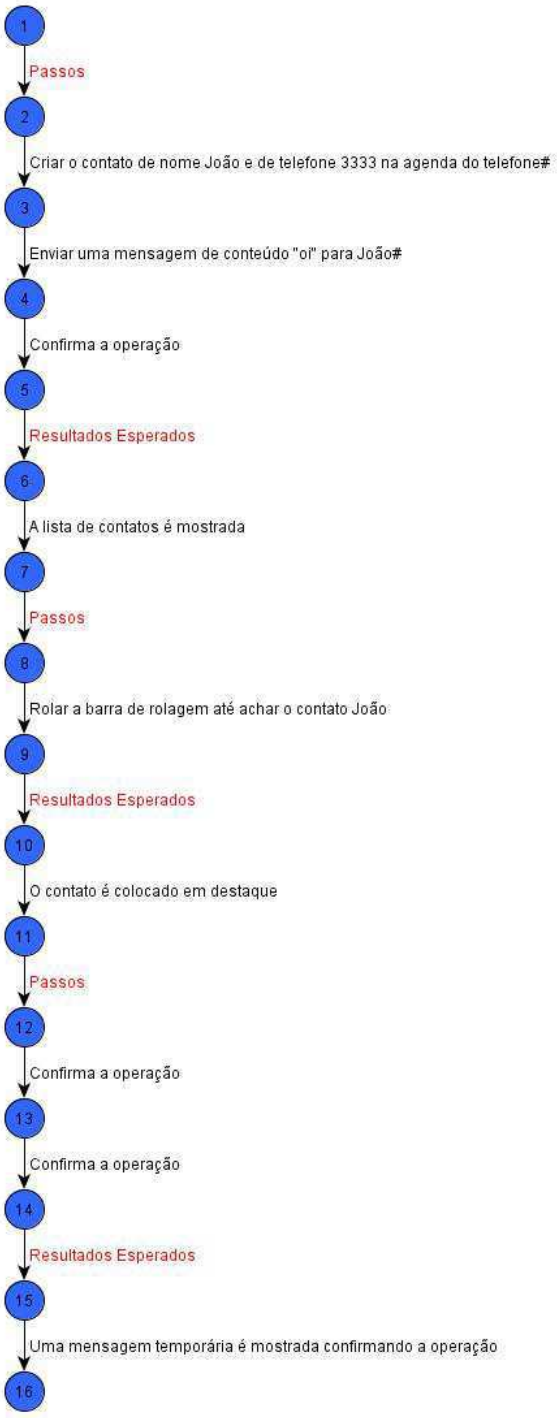


Figura 4.10: LTS resultante da Figura 4.9 sem detalhes

Capítulo 5

Estudo de Caso

As etapas que constituem o processo de teste baseado em modelos são, em geral, quatro: *(a) elaboração do modelo teste*, *(b) geração e seleção dos casos de teste*, (*c) execução dos casos de teste* e *(d) análise dos resultados*. Estamos diretamente relacionados à etapa *(a)* e, portanto, construiremos métricas para que seja possível avaliar o procedimento que propomos relacionado a esta etapa. No entanto, a partir da etapa *(a)* decorrem as outras atividades. Como estamos enquadrados no tipo de execução manual, e considerando que *(b)* se constitui de uma atividade automática, iremos relacionar métricas da combinação entre *(a)*, *(c)* e *(d)*.

Neste capítulo, iremos mostrar um conjunto de métricas e procedimentos para a realização de um estudo de caso com intuito de avaliarmos nossa estratégia de escrita de modelo de teste. Ao final, analisaremos o resultado da execução do estudo de caso.

5.1 Métricas e Procedimento

5.1.1 MÉTRICAS

As métricas que utilizamos são baseadas no paradigma *Goal Question Metric* (GQM) [6], cuja abordagem define um mecanismo para definição e análise de medição de software. Basicamente, este paradigma engloba a definição de um objetivo (*Goal*) relacionado ao objeto em estudo. Em seguida, são levantas questões (*Question*) que caracterizam o objetivo, de forma operacional, com respeito a algum critério de qualidade escolhido. Definidos os questionamentos, o próximo passo é definir um conjunto de dados (*Metric*) que responda às questões

de forma quantitativa. Resumindo, a aplicação do paradigma envolve cinco etapas:

1. Definição dos objetivos
2. Elaboração de questões que caracterizam o objetivo
3. Especificação de métricas que respondem as questões levantadas
4. Desenvolvimento de mecanismos para a coleta de dados
5. Coleta, Validação e Análise dos Dados

OBJETIVO

Seguindo as etapas propostas, iremos definir o objetivo. O objetivo sintetiza o que se deseja avaliar, ou seja, define o propósito do estudo. De acordo com o paradigma GQM, a definição de um objetivo inicia-se com a escolha do objeto de estudo e, em seguida, com a resposta da questão: Por que esse objeto precisa ser estudado? Nosso objetivo será caracterizado através de cinco aspectos [21]:

- **O objeto:** exprime o principal alvo do estudo (o processo ou produto que será analisado).
- **O propósito:** indica como o objeto será analisado.
- **O foco de qualidade:** expressa as propriedades particulares do objeto que será analisado no decorrer do estudo, tais como custos, confiabilidade, etc.
- **O ponto de vista:** manifesta informações sobre o grupo de pessoas que vão interpretar os dados.
- **O ambiente:** expressa o contexto em que o estudo será realizado.

Utilizando esses cinco aspectos, caracterizamos nosso objetivo como ilustrado na Figura 5.1. Em outras palavras, estamos analisando o processo de escrita de modelos de teste no contexto do processo de Teste Baseados em Modelos com intenção de melhorar o processo através de modelos mais completos, os quais proporcionam mais facilidade de escrita e execução.

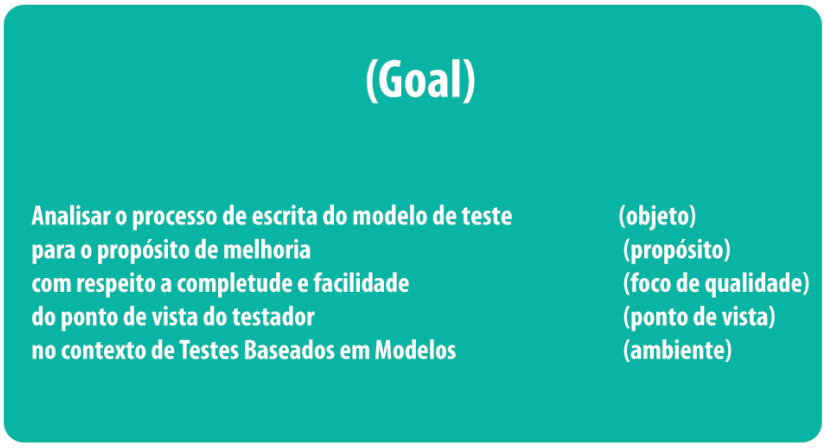


Figura 5.1: Goal

Em seguida, refinaremos o objetivo através de uma estrutura conhecida por GQM *Abstraction Sheet* [21]. Esta estrutura é utilizada como uma visão abstrata do plano de GQM que ajuda a revelar a dependência entre as questões que caracterizam o objetivo. Consiste em quatro quadrantes com as seguintes informações:

- **Focos de qualidade:** este quadrante captura a informação que define os focos de qualidade. A informação destina-se a captar a intuição do ponto de vista sobre os focos de qualidade que os transformá em uma definição operacional.

- **Hipótese de linha de base:** este quadrante documenta dados concretos sobre os focos de qualidade. As declarações registradas aqui têm por finalidade capturar a experiência acerca do estado das coisas no inicio da medição.

- **Fatores de variações:** este quadrante capta os fatores que causam impacto sobre os focos de qualidade em um contexto particular.

- **Impacto nas hipóteses de linha de base:** as relações entre os fatores de variações e os focos de qualidade são registradas aqui concretamente.

Seguindo a linha de raciocínio da *GQM Abstraction Sheet*, nosso objetivo foi refinado e o resultado pode ver visualizado na Figura 5.2. O foco de qualidade **facilidade** diz respeito ao esforço de utilização da técnica, e pode ser medido pelo tempo requerido na escrita do modelo de teste e pelo tempo decorrido durante a execução. Já o foco de qualidade **completude** diz respeito a quão completo está o modelo de teste, e pode impactar no número de casos

de testes bloqueados. As hipóteses de linha de base deveriam informar valores para os focos de qualidade, no entanto estes valores variam por fatores como tamanho da *feature*, e serão coletados a partir de um grupo que realizará o experimento sem utilizar nossa estratégia de escrita de modelo de teste com padrões. Os fatores de variações estão diretamente ligados à experiência do testador. Por fim, relacionamos os fatores de variações aos focos de qualidade no quarto quandrante para que possamos guiar o experimento considerando tais fatores.

Figura 5.2: *GQM Abstraction Sheet*

QUESTÕES

A segunda etapa é a elaboração de questões, que consiste na parte operacional da elaboração do modelo. Significa levantar questões que possam atingir o objetivo proposto. O nosso objetivo é avaliar o processo de escrita do modelo de teste dentro do contexto de Testes Baseados em Modelos. Uma questão que pode ser levantada é o esforço. **Qual esforço do processo adotado para escrever o modelo de teste?** Outro questionamento diz respeito a eficiência do processo adotado: **qual a eficiência do processo?** Por fim, sabemos que no contexto de Testes Baseados em Modelos a geração se constitui de uma atividade de caráter automático. Entretanto, a execução pode se caracterizar como procedimento manual (o nosso contexto é um exemplo). Outra questão, portanto, seria a relação do processo de escrita do modelo de teste com os casos de testes resultantes: **qual impacto causado à execução dos casos de teste?**

MÉTRICAS

A especificação das métricas tem por objetivo quantificar as questões quem foram abordadas. O esforço para escrever o modelo de teste pode ser quantificado pelo tempo gasto para exercer a atividade. Em geral, temos o tempo do entendimento dos requisitos mais o tempo com a escrita propriamente dita. Representamos a soma destas atividades por T_{esc}. A eficiência, visto que estamos pensando somente em termos do modelo, pode ser medida pela porcentagem de casos de testes não bloqueados da suite de teste gerada. Ou seja, para N casos de teste, qual a porcentagem de casos de teste não bloqueados, dado N_b o número de casos de testes bloqueados ? O impacto na execução pode ser visto pelo esforço exigido pela atividade, e tal esforço pode ser medido também pelo tempo requerido pela tarefa de executar os casos de teste. O tipo de execução manual envolve o tempo para o entendimento dos casos de teste e o tempo para execução no sistema (T_{exe})[1]. A Figura 5.3 ilustra nosso modelo GQM completo.

5.1.2 PROCEDIMENTOS

O procedimento adotado corresponde às etapas restantes (4 e 5) de aplicação do paradigma. A aplicação do modelo de medição consiste em coletar os dados necessários para o cálculo das métricas, bem como analisar os resultados obtidos. Estamos interessados em avaliar nossa estratégia de escrita de casos de uso com utilização de padrões de casos de teste. E

[1] Questões como configuração do sistema estão considerados em conjunto com o tempo de execução.

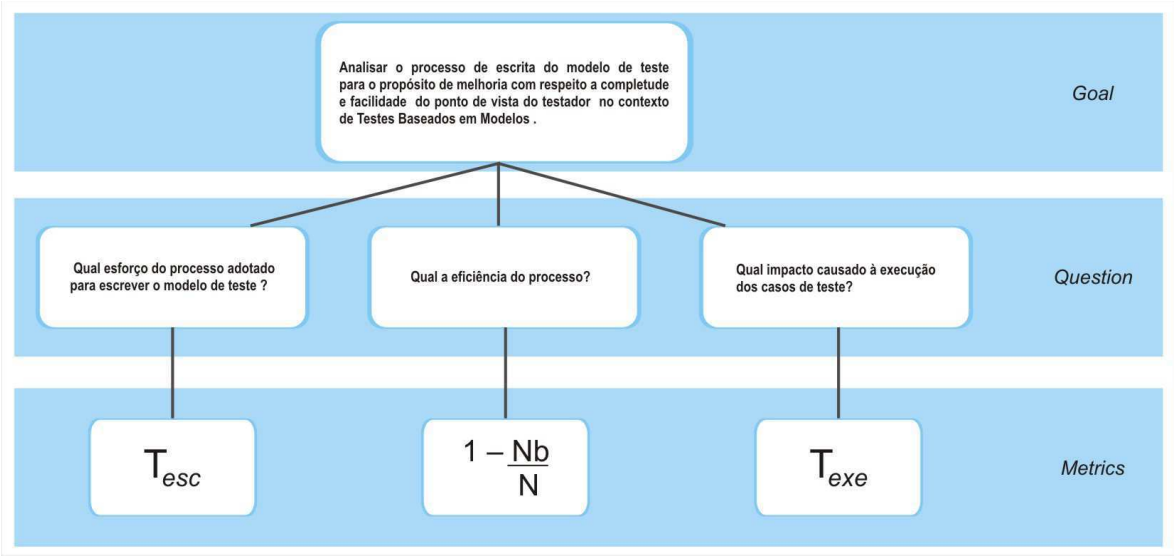


Figura 5.3: Modelo GQM

como estamos interessados em agregar valor ao procedimento padrão do contexto MBT, faremos análise comparativa através de um estudo de caso aplicado tanto a nossa técnica como a já existente. Para tanto, as atividades necessárias à coleta dos dados são:

- **Construção do modelo:** consiste em mapear os requisitos funcionais para uma notação formal ou semi-formal que representará o comportamento funcional da *feature*. No caso, a notação utilizada são diagramas de seqüência do *framework* UML versão 2.0. Envolve as seguintes tarefas, e corresponde à métrica T_{esc}:
  - Metodologia sem padrões
    - Leitura e entendimento dos requisitos.
    - Escrita do modelo de casos de uso.
  - Metodologia com padrões
    - Leitura e entendimento dos requisitos.
    - Escrita do modelo de casos de uso.
    - Busca pelo padrão.
    - Adaptação do padrão ao contexto específico.
- **Execução dos casos de teste:** Consiste na execução dos casos de teste gerados a partir

do modelo construído na etapa anterior. Define a variável T_{exe}. Possui atividades idênticas para as dois processos:

- Leitura e entendimento dos casos de teste.
- Execução dos casos de teste.
- Análise do resultado.

A coleta dos dados deve ocorrer durante as atividades listadas. A métrica de eficiência pode ser coletada pela tarefa **Análise do resultado** da atividade **Execução dos casos de teste**, onde o testador, ao finalizar a execução do caso de teste, associa um resultado: *passou*, *falhou* ou *bloqueado*, que caracteriza o teste que, por alguma condição não satisfeita ao longo do teste, não teve sua execução concluída, como por exemplo algum passo que o testador não sabia como executar.

5.2 Ambiente e execução

O estudo foi conduzido dentro do ambiente Motorola por quatro pessoas selecionadas levando em consideração os fatores de variações (Figura 5.2). A Tabela 5.1 resume o perfil de cada pessoa.

Tabela 5.1: *Perfis*

	design1	*design2*	*testador1*	*testador2*
Conhecimento em UML	sim	sim	-	-
Conhecimento da *feature*	não	não	não	não
Conhecimento na escrita de casos de uso	sim	sim	-	-
Conhecimento em execução de casos de teste	-	-	sim	sim

O experimento foi conduzido da seguinte forma: uma pessoa que possuía conhecimento no tipo da *feature* utilizada catalogou algumas ações padrões. Em seguida, foram entregues artefatos a dois *designers* de teste para que escrevessem o modelo de teste correspondente à *feature*. O *designer1* utilizou apenas o documento de requisitos e o *designer2* utilizou os requisitos mais os padrões que foram catalogados. Os modelos produzidos foram executados

pelo *testador1* (modelo produzido pelo *designer2*) e pelo *testador2* (modelo produzido pelo *designer1*). Ao término, obtivemos os dados resumidos nas Tabelas 5.2, 5.3 e 5.4.

Tabela 5.2: *Dados coletados do designers*

Designer	Número de casos de teste	Tempo para elaborar o modelo(min)
designer1 (sem padrões)	6	725
designer2 (com padrões)	7	1230

Tabela 5.3: *Dados coletados do testador1 (com padrões)*

	Tempo gasto na execução do caso de teste (seg)	Resultado
Caso de teste 01	32	*passou*
Caso de teste 02	40	*passou*
Caso de teste 03	28	*passou*
Caso de teste 04	59	*passou*
Caso de teste 05	44	*passou*
Caso de teste 06	75	*passou*
Caso de teste 07	30	*passou*

Obtivemos, portanto os seguintes resultados para as métricas propostas: T_{esc}(*desiner1*) = 725, T_{esc}(*designer2*) = 1230, T_{exe}(testador1) = 308, T_{exe}(testador2) = 379, $1 - \frac{N_b}{N}$(design1) = 1, $1 - \frac{N_b}{N}$(design1) = 0,84.

5.3 Análise dos resultados

Os resultados obtidos podem ser analisados por várias perspectivas. Vai depender do objetivo e da situação mais favorável ao estudo em questão. Pelo que podemos observar das métricas fornecidas pelo estudo de caso, não houve técnica que tenha sido melhor em todos os dados. Enquanto o tempo gasto para escrever o modelo pela técnica sem padrão foi $\approx 40\%$ menor do que a técnica com padrões, houve uma perda de $\approx 20\%$ do tempo na etapa de execução em relação à abordagem com padrões. Se considerarmos o fator eficiência, cuja métrica define uma porcentagem de casos de testes não bloqueados, temos um ganho de $\approx 16\%$ com

Tabela 5.4: *Dados coletados do testador2 (sem padrões)*

	Tempo gasto na execução do caso de teste (seg)	Resultado
Caso de teste 01	79	*passou*
Caso de teste 02	60	*passou*
Caso de teste 03	45	*passou*
Caso de teste 04	110	*bloqueado*
Caso de teste 05	50	*passou*
Caso de teste 06	35	*falhou*

a técnica que se utiliza de padrões, o que nos leva a crer que a técnica com suporte de padrões requer um esforço inicial maior, no entanto este esforço pode ser recompensado nas etapas subseqüentes.

É importante ressaltar que o *designer1* realizou a escrita do modelo de caso de uso com suporte de padrões pela primeira vez, o que pode ter impactado no tempo. É necessária a realização de outros estudos de caso, com mais dados, para verificar como se comportam as métricas e de estudos de caso considerando como fator de variação o conhecimento na utilização de padrões.

Outra questão que deveria ter sido levada em consideração é a similaridade com respeito às funcionalidades entre os casos de teste gerados considerando as duas técnicas. No estudo de caso realizado, os casos de teste de um a cinco (em negrito) nas duas tabelas possuem equivalência entre si. O caso de teste 1 da Tabela 5.3 é equivalente ao caso de teste 1 da Tabela 5.4 e, assim, sucessivamente, até o quinto caso de teste. Para estabelecer mais critérios, deveríamos considerar apenas os cinco primeiros casos de teste de cada tabela. Entretanto, no exemplo especifico, teríamos resultados similares considerando que cortaríamos o maior tempo de execução da abordagem mais rápida na soma geral (caso de teste 6 da Tabela 5.3), e o caso de teste bloqueado continuaria participando da métrica de eficiência (caso de teste 4 da Tabela 5.4).

5.4 Conclusão

Neste capítulo apresentamos um modelo de medição com questionamentos e métricas utilizando o *framework* GQM no intuito de avaliar o processo de escrita do modelo de teste dentro do contexto de MBT. Como forma de prover valores às métricas sugeridas, um estudo de caso foi realizado dentro de um ambiente real onde foi possível definir em que etapas seriam possíveis a coleta de métricas. A idéia é que a coleta de métricas seja minimamente intrusiva dentro do processo de teste para que seja possível a coleta constante e com isso possuirmos histórico de métricas. A vantagem deste modelo é que ele foi ajustado às etapas de forma a não causar esforço em sua utilização. A desvantagem é que tivemos o esforço de controlar os passos e, na prática, isto se torna inviável. Um passo que pode ser dado é acoplar uma forma automática de solucionar a montagem do modelo de medição.

Realizar estudo de caso dentro do ambiente de execução real é um grande desafio. É preciso não interferir muito na execução normal das atividades e, ao mesmo tempo, coletar valores significativos. O momento precisa ser bem definido e as pessoas devem estar dispostas e/ou disponíveis para colaborar com o experimento. Geralmente, não há possibilidade de repetição e o planejamento é crucial. Como conseqüência, o experimento pode acabar com escopo reduzido e com poucos integrantes colaborando . O que, entretanto, não diminui o seu valor experimental. Obtivemos alguns resultados que não foram diretamente relacionados às métricas coletadas no estudo de caso realizado:

- A utilização de padrões resultou na uniformidade da escrita dos casos de uso. O testador que conduziu a escrita do modelo se utilizando do repositório de padrões produziu um documento com escrita padronizada tanto na seqüência em que foram escritas as sentenças como na utilização de vocabulário restrito de verbos.

- Ações que devem ser repetidas no mesmo caso de uso só precisam ser escritas uma única vez e referenciadas onde necessário. Por exemplo, deletar uma imagem associada a um contato possui as seguintes operações (considerando a execução manual):

  - **Ir até o contato, ver a imagem.**
  - Deletar a imagem.
  - **Ir até o contato , ver a imagem (para verificar se foi deletada).**

- A possibilidade de escrever mais cenários é maior se existem também padrões de *features* associadas.

- A utilização de notação explícita dos pontos de extensão auxilia na agilidade da inspeção do modelo produzido, pois fica mais explícita a relação existente entre os casos de uso

- A utilização de padrões instigou o *designer*, mesmo com perfil de pouca experiência, a investigar a possibilidade de outros padrões no contexto em teste.

Nossas conclusões a respeito do comparativo entre as duas técnicas utilizadas notificou um resultado que precisa ser mais investigado pela execução de outros estudos de casos, bem como utilização de outros modelos de teste. No entanto, estamos apenas nos primeiros passos desta investigação.

Capítulo 6

Conclusão

Este trabalho apresentou uma estratégia para escrita de modelo de teste dentro do contexto de Testes Baseados em Modelos. Um dos caminhos propostos foi uma abordagem para auxiliar o processo de especificação de casos de uso a partir do uso de padrões de comportamento que detalham ações abstratas padrão para um domínio de aplicações. Outro caminho adotado foi a utilização de uma nomenclatura para representar as relações existentes entre casos de uso. Adotamos também uma estratégia de seleção onde é possível selecionar casos de teste com detalhes para atender a perfis distintos. A integração deste trabalho à ferramenta LTS-BT possibilitou a utilização de um ambiente completo com suporte à geração e seleção de casos de teste. No entanto, por restrições de tempo e recurso disponível, utilizamos apenas a geração completa com todos os casos de teste extraídos do modelo de teste, pois a execução ocorreu em apenas um ciclo de execução.

Conduzimos nosso estudo de caso em um ambiente industrial, o que nos possibilitou uma análise real dos fatos. Algumas restrições impediram a condução controlada, e deveremos conduzir outros estudos de caso a fim de analisarmos melhor as métricas propostas. No entanto, obtivemos alguns resultados que não foram diretamente relacionados às métricas coletadas no estudo de caso realizado:

- A utilização de padrões resultou na uniformidade da escrita dos casos de uso. O testador que conduziu a escrita do modelo se utilizando do repositório de padrões produziu um documento com escrita padronizada tanto na seqüência em que foram escritas as sentenças como na utilização de vocabulário restrito de verbos.

- Ações que devem ser repetidas no mesmo caso de uso só precisam ser escritas uma única vez e referenciadas onde necessário. Por exemplo, deletar uma imagem associada a um contato possui as seguintes operações (considerando a execução manual):
  - **Ir até o contato, ver a imagem.**
  - Deletar a imagem.
  - **Ir até o contato , ver a imagem (para verificar se foi deletada).**
- A possibilidade de escrever mais cenários é maior se existem também padrões de *features* associadas.
- A utilização de notação explícita dos pontos de extensão auxilia na agilidade da inspeção do modelo produzido, pois fica mais explícita a relação existente entre os casos de uso
- A utilização de padrões instigou o *designer*, mesmo com perfil de pouca experiência, a investigar a possibilidade de outros padrões no contexto em teste.

6.1 Trabalhos relacionados

Diferentes abordagens têm sido propostas para derivação testes a partir de casos de uso. Ryser e Glinz propuseram um método para Scenario-Based Validation and Test of Software (SCENT) [53]. Nesta abordagem, os casos de uso são formalizados em máquinas de estado. Estas máquinas de estado são então anotadas com informação, tais como pré-condições, dados e requisitos não-funcionais. Finalmente, casos de teste são gerados pelos caminhos das máquinas de estado. A abordagem SCENT apresenta gráficos de dependência para capturar dependências entre casos de uso e gerar casos de teste de sistema. Em [14], Briand e Labiche apresentam uma metodologia para testes de sistemas baseados em modelos UML, tais como modelos de caso de uso, diagramas de interação, diagramas de classe e OCL. Os casos de uso são refinados como diagramas de interação e diagramas de atividade são utilizadas para capturar dependências de casos de uso. Em [40; 25] máquinas de estado são derivadas de casos de uso. Uma diferença fundamental entre a nossa abordagem e as abordagens citadas é que propomos automatizados de geração de cenários de teste a partir dos casos

de uso original em forma de texto. Nosso objetivo é obter testes de nível de sistema, tanto quanto possível de modelos de requisitos. Outra diferença é que nós não usamos um modelo diferente para descrever as relações seqüenciais entre os casos de uso, as relações já estão embutidas como notação nos casos de uso.

No tocante a requisitos de domínio, trabalhos com idéia de semelhante são os testes de família de produto. Estes podem ser divididos em testes do domínio, ou seja, testes genéricos ao domínio da família de produtos, e testes de aplicação, ou seja, testes para um produto específico de um determinado cliente. Testes de sistema do domínio compreendem principalmente a produção de artefatos de teste reutilizáveis. Testes de sistema de aplicação compreendem a reutilização e adaptação desses artefatos e o desenvolvimento de novos devido a requisitos específicos do cliente. Compartilhamos desta linha de trabalho pela idéia de reusabilidade. Nebut *et al* [41] adicionaram uma técnica à ferramenta TGV em que os casos teste para cada um dos diferentes produtos de uma linha de produtos são gerados a partir dos mesmos requisitos funcionais utilizando padrões de comportamento. Por outro lado, Kamsties *et al.* [32] propuseram uma abordagem para derivar casos de teste de sistema do domínio e testes de aplicação através da utilização de casos de uso que contêm variabilidades. Por fim, Bertolino e Gnesi [10] propõem uma metodologia para gerenciar o processo de teste de linhas de produtos baseado no método de categoria de partição. O ponto onde nos diferenciamos das abordagens acima descritas é que propomos pequenos modelos de uso comum enquanto que nas abordagens de linha de produto é construído um modelo geral contendo tudo que é similar entre os produtos e dele são retirados apenas o que é específico.

Em [34], Katara *et al* utiliza-se da restrição de domínio para propor uma metodologia em torno de uma linguagem de modelagem de domínio específico consistindo das chamadas palavras de ação e palavras-chave, que descrevem ações do usuário, com um elevado nível de abstração, bem como suas implementações de nível inferior, respectivamente. Em[51], Roubtsov e Heck utilizaram uma abordagem baseada em caso de uso para derivação de testes reproduzidos em escala industrial através da utilização de especificação de artefatos de teste em três níveis. Ao mais alto nível, cenários de teste são utilizados para validar casos de uso de sistema. Os níveis mais baixos são apresentados pelos scripts de teste e casos de teste. Relacionamos-nos a estes trabalhos pela utilização de artefatos contentores de pouco e muito detalhes.

Temos abordado estas questões de um ângulo ligeiramente diferente. Em um domínio restrito como o nosso, a modelagem de teste pode ser atribuída a alguns especialistas internos ou a terceiros. Então, a tarefa principal do testador experiente no domínio é o de transformar os casos de uso em seqüências de ações pré-definidas as quais definem um padrão de comportamento de uma funcionalidade específica. Os padrões correspondem a conceitos familiares para testadores de um domínio particular.

6.2 Trabalhos futuros

O trabalho aqui proposto consiste apenas em um passo inicial da estratégia proposta para escrita de modelo de teste no contexto de Testes Baseados em Modelos. Como prosseguimento ao que aqui foi proposto, sugerimos:

- **Realização de estudo de caso de contexto mais amplo** - o estudo de caso realizado precisa ser conduzido num contexto mais amplo onde estejam presentes mais padrões no repositório. E as métricas devem coletadas e armazenadas para que se tenha um histórico que sirva de linha de base para estimativa de tempo.

- **Realização de estudo de caso variando os fatores de variação** - no estudo realizado utilizamos perfis que fossem iguais quanto aos fatores de variação. Outra abordagem seria variar os perfis quanto aos fatores, no entanto necessitaria de mais tempo e talvez outra técnica para avaliar os resultados.

- **Análise de outras métricas**- é preciso avaliar os padrões quanto a sua reusabilidade. Métricas como freqüência de reuso, eficiência de reuso e esforço de adaptação pode ser utilizados para verificar o quão relevantes são os padrões catalogados e como podem ser melhorados.

- **Combinação com outras técnicas** - combinação da escrita de padrões em conjunto com um processamento de linguagem natural controlada [57].

- **Identificação Automática de padrões** - dado um conjunto de casos de uso, fornecer uma forma automática de indentificar padrões de caso de teste.

- **Raciocínio Baseado em Casos** - estudar a possibilidade de resolver o problema de inclusões/atualizações, além de ter mais liberdade na escolha dos atributos de indexação, utilizando o arcabouço de Raciocínio Baseado em Casos(RBC) da Inteligência Artificial Simbólica.

Apesar de ter sido experimentada dentro do contexto de aplicações para celulares, a estratégia de forma geral é independente deste contexto e pode ser aplicada em outros contextos.

Bibliografia

[1] Steve Adolph and Paul Bramble. Patterns for effective use cases, 2001.

[2] Jesús Manuel Almendros-Jiménez and Luis Iribarne. Describing use-case relationships with sequence diagrams. *Comput. J*, 50(1):116–128, 2007.

[3] Ricardo Baeza-Yates and Berthier Ribeiro-Neto. *Modern Information Retrieval*. Addison-Wesley, 1999.

[4] F. Basanieri, A. Bertolino, G. Lombardi, G. Nucera, E. Marchetti, and A. Ribolini. Cow_Suite: A UML-based tool for test-suite planning and derivation. *ERCIM News*, 58:30–32, 2004.

[5] Francesca Basanieri, Antonia Bertolino, and Eda Marchetti. The Cow_Suite approach to planning and deriving test suites in UML projects. *Lecture Notes in Computer Science*, 2460, 2002.

[6] Victor R. Basili. Software modeling and measurement: The goal/question/metric paradigm. Technical report, September 1992.

[7] Boris Beizer. *Black-Box Testing: Techniques for Functional Testing of Software and Systems*. John Wiley & Sons, 1999.

[8] Eddy Bernard, Bruno Legeard, Xavier Luck, and Fabien Peureux. Generation of test sequences from formal specifications: GSM 11-11 standard case study. *Softw, Pract. Exper*, 34(10):915–948, 2004.

[9] Antonia Bertolino. Software testing research: Achievements, challenges,dreams. In *FOSE '07: 2007 Future of Software Engineering*, pages 85–103, Washington, DC, USA, 2007. IEEE Computer Society.

[10] Antonia Bertolino and Stefania Gnesi. Use case-based testing of product lines. In *ESEC/SIGSOFT FSE*, pages 355–358. ACM, 2003.

[11] Antonia Bertolino and Stefania Gnesi. Use case-based testing of product lines. In *ESEC/SIGSOFT FSE*, pages 355–358. ACM, 2003.

[12] Antonia Bertolino, Eda Marchetti, and Henry Muccini. Introducing a reasonably complete and coherent approach for model-based testing. *Electr. Notes Theor. Comput. Sci*, 116:85–97, 2005.

[13] Robert V. Binder. *Testing object-oriented systems: models, patterns, and tools*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1999.

[14] Lionel C. Briand and Yvan Labiche. A UML-based approach to system testing. *Software and System Modeling*, 1(1):10–42, 2002.

[15] Gustavo Cabral and Augusto Sampaio. Formal specification generation from requirement documents. *Electron. Notes Theor. Comput. Sci.*, 195:171–188, 2008.

[16] Emanuela G. Cartaxo, Wilkerson L. Andrade, Francisco G. Oliveira Neto, and Patrícia D. L. Machado. Lts-bt: a tool to generate and select functional test cases for embedded systems. In *SAC '08: Proceedings of the 2008 ACM symposium on Applied computing*, pages 1540–1544, New York, NY, USA, 2008. ACM.

[17] CCITT. CCITT recommendation Z.120: Message sequence chart (MSC92). Technical report, CCITT, Geneva, 1992.

[18] Alistair Cockburn. *Writing Effective Use Cases*. Addison-Wesley Longman, 2000.

[19] S. Dalal, A. Jain, C. Lott, G. Patton, N. Karunanith, J. M. Leaton, and B. M. Horowitz. Model-based testing in practice. In *Proceedings of the 21st International Conference on Software Engineering*, pages 285–294. ACM Press, May 1999.

[20] Wilkerson de Lucena Andrade. Geração de casos de teste de interação para aplicações de celulares. Master's thesis, Universidade Federal de Campina Grande, 2007.

[21] Christiane Differding, Barbara Hoisl, and Christopher M. Lott. Technology package for the Goal Question Metric Paradigm. Technischer Bericht 281-96, Fachbereich Informatik, Universität Kaiserslautern, 67653 Kaiserslautern, April 1996.

[22] Laísa Helena Oliveira do Nascimento. Abordagens para avaliação experimental de teste baseado em modelos de aplicações reativas. Master's thesis, Universidade Federal de Campina Grande, 2008.

[23] I. K. El-Far and J. A. Whittaker. Model-based software testing. *Encyclopedia on Software Engineering*, 2001.

[24] E. Farchi, A. Hartman, and S. S. Pinter. Using a model-based test generator to test for standard conformance. *IBM Systems Journal*, 41(1):89–110, ???? 2002.

[25] Peter Fröhlich and Johannes Link. Automated test case generation from dynamic models. In E. Bertino, editor, *Proceedings of ECOOP 2000*, volume 1850 of *LNCS*, pages 472–491. Springer, 2000.

[26] Object Management Group. *OMG Unified Modeling Language, Version 1.4*. OMG, http://www.omg.com/uml/, 2001.

[27] ISO/IEC. Information technology. open systems interconnection. conformance testing methodology and framework., 1994-1997. International ISO/IEC multipart standard No.9646.

[28] ITU-TS. ITU-TS recommendation Z.120: Message sequence chart 1996 (MSC96). Technical report, ITU-TS, Geneva, 1996.

[29] Antti Jääskeläinen. A domain-specific tool for creation and management of test models, January 2008. M.Sc. Thesis.

[30] Ina Schieferdecker Zhen Ru Dai1 Justyna Zander-Nowicka1, Abel Marrero Pérez. Test design patterns for embedded systems. Potsdam, Germany, September 2007. 10th International Conference on Quality Engineering in Software Technology.

[31] Ibrahim K. and El-Far. Enjoying the perks of model-based testing. Software Testing, Analysis, and Review Conference, October/November 2001.

[32] Erik Kamsties, Klaus Pohl, Sacha Reis, and Andreas Reuys. Testing variabilities in use case models. In Frank van der Linden, editor, *Proceedings of the Fifth International Workshop on Product Family Engineering (PFE-5)*, LNCS 3014, Siena, Italy, 2003. Springer Verlag.

[33] Cem Kaner, James Bach, and Bret Pettichord. *Lessons Learned in Software Testing*. John Wiley & Sons, Inc., New York, NY, USA, 2001.

[34] Mika Katara, Antti Kervinen, Mika Maunumaa, Tuula Pääkkönen, and Mikko Satama. Towards deploying model-based testing with a domain-specific modeling approach. In Phil McMinn, editor, *TAIC PART*, pages 81–89. IEEE Computer Society, 2006.

[35] Edward Kit and Tyler Wilkins. Software testing in the real world methodology. 2004.

[36] Bjorn Regnell Lund, Michael Andersson, and Johan Bergstrand. A hierarchical use case model with graphical representation. In *ECBS '96: Proceedings of the IEEE Symposium and Workshop on Engineering of Computer Based Systems*, page 270, Washington, DC, USA, 1996. IEEE Computer Society.

[37] Mika Maunumaa Tuula Pääkkönen Mika Katara, Antti Kervinen and Antti Jääskeläinen. Can i have some model-based gui tests please providing a model-based testing service through a web interface. In *Proceedings of the second annual Conference of the Association for Software Testing (CAST 2007)*, July 2007.

[38] E. F. Moore. Gedanken experiments on sequential machines. In *Automata Studies*, pages 129–153, Princeton, 1956.

[39] Laisa H. O. Nascimento and Patricia D. L. Machado. An experimental evaluation of approaches to feature testing in the mobile phone applications domain. In *DOSTA '07: Workshop on Domain specific approaches to software test automation*, pages 27–33, New York, NY, USA, 2007. ACM.

[40] C. Nebut, F. Fleurey, Y. Le Traon, and J. M. Jezequel. Automatic test generation: A use case driven approach. 2006.

[41] Clémentine Nebut, Simon Pickin, Yves Le Traon, and Jean-Marc Jézéquel. Reusable test requirements for UML-modeled product lines. pages 51–56, September 2002.

[42] Helmut Wolfram Neukirchen. *Languages, Tools and Patterns for the Specification of Distributed Real-Time Tests*. PhD thesis, University of Goettingen, 2004.

[43] Object Management Group, Framingham, Massachusetts. *UML 2.0 Superstructure Specification*, October 2004.

[44] ObjectMentor. Junit test patterns, 2004. http://www.junit.org/news/article/patterns/index.htm.

[45] Erika Mir Olimpiew and Hassan Gomaa. Model-based testing for applications derived from software product lines. *ACM SIGSOFT Software Engineering Notes*, 30(4):1–7, 2005.

[46] Tom Pender. *UML Bible*. John Wiley & Sons, Inc., 2003.

[47] A. Pretschner, W. Prenninger, S. Wagner, C. Kühnel, M. Baumgartner, B. Sostawa, R. Zölch, and T. Stauner. One evaluation of model-based testing and its automation. In *ICSE '05: Proceedings of the 27th international conference on Software engineering*, pages 392–401, New York, NY, USA, 2005. ACM.

[48] Olli-Pekka Puolitaival. Adapting model-based testing to agile context, 2008.

[49] Harry Robinson. Obstacles and opportunities for model-based testing in an industrial software environment. In *1st European Conference on Model Driven Software Engineering*, pages 118–127, December 2003.

[50] David S. Rosenblum. Formal methods and testing: why the state-of-the art is not the state-of-the practice. 1996.

[51] Serguei A. Roubtsov and Petra Heck. Use case-based acceptance testing of a large industrial system: Approach and experience report. In Phil McMinn, editor, *TAIC PART*, pages 211–220. IEEE Computer Society, 2006.

[52] Bernhard Rumpe. Agile test-based modeling. In Hamid R. Arabnia and Hassan Reza, editors, *Software Engineering Research and Practice*, pages 10–15. CSREA Press, 2006.

[53] Johannes Ryser and Martin Glinz. A scenario-based approach to validating and testing software systems using statecharts. In *Software and Systems Engineering and Their Applications (ICSSEA'99)*, December 1999.

[54] Mikko Satama. Event capturing tool for model-based gui test automation, September 2006. M.Sc. Thesis.

[55] Ina Schieferdecker and Juergen Grossmann. Testing hybrid control systems with TTCN-3: an overview on continuous TTCN-3. 2008.

[56] Avik Sinha and Carol Smidts. HOTTest: A model-based test design technique for enhanced testing of domain-specific applications. *ACM Transactions on Software Engineering and Methodology*, 15(3):242–278, July 2006.

[57] Dante Gama Torres. Specnl: Uma ferramenta para gerar descrições em linguagem natural a partir de especificações de casos de teste. Master's thesis, Universidade Federal de Pernambuco, 2008.

[58] J. Tretmans. Testing labelled transition systems with inputs and outputs. Memoranda Informatica 95-26, University of Twente, 1995.

[59] Jan Tretmans. Test generation with inputs, outputs, and quiescence. In Tiziana Margaria and Bernhard Steffen, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, volume 1055 of *Lecture Notes in Computer Science*, pages 127–146. Springer-Verlag, 1996.

[60] C. Reid Turner, Alfonso Fuggetta, Luigi Lavazza, and Alexander L. Wolf. Feature engineering. In *Proceedings of the 9th International Workshop on Software Specification and Design*, pages 162–164, 1998.

[61] Mark Utting and Bruno Legeard. *Practical Model-Based Testing: A Tools Approach*. Morgan-Kaufmann, 2006.

[62] Mark Utting, Alexander Pretschner, and Bruno Legeard. A taxonomy of model-based testing. Technical report, Department of Computer Science, The University of Waikato (New Zealand), 2006.