UNIVERSIDADE ESTADUAL DO CEARÁ

CENTRO DE CIÊNCIAS E TECNOLOGIA

PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

MESTRADO ACADÊMICO EM CIÊNCIA DA COMPUTAÇÃO

AMANDA SOUZA DA SILVA

UMA ABORDAGEM INTERATIVA E SEMI-AUTOMÁTICA PARA A DETECÇÃO E REMOÇÃO DE CENÁRIOS IMPLÍCITOS EM MODELOS COMPORTAMENTAIS USANDO RASTROS DE EXECUÇÃO

FORTALEZA – CEARÁ

2017

AMANDA SOUZA DA SILVA

UMA ABORDAGEM INTERATIVA E SEMI-AUTOMÁTICA PARA A DETECÇÃO E REMOÇÃO DE CENÁRIOS IMPLÍCITOS EM MODELOS COMPORTAMENTAIS USANDO RASTROS DE EXECUÇÃO

Dissertação apresentada ao Curso de Mestrado Acadêmico em Ciência da Computação do Programa de Pós-Graduação em Ciência da Computação do Centro de Ciências e Tecnologia da Universidade Estadual do Ceará, como requisito parcial à obtenção do título de mestre em Ciência da Computação. Área de Concentração: Ciência da Computação

Orientador: Paulo Henrique M. Maia

FORTALEZA – CEARÁ

2017

Dados Internacionais de Catalogação na Publicação

Universidade Estadual do Ceará

Sistema de Bibliotecas

Silva, Amanda Souza da .
Uma Abordagem Interativa e Semi-Automática para a Detecção e Remoção de Cenários Implícitos em Modelos Comportamentais usando Rastros de Execução [recurso eletrônico] / Amanda Souza da Silva. - 2017.
1 CD-ROM: il.; 4 ¾ pol.

CD-ROM contendo o arquivo no formato PDF do trabalho acadêmico com 71 folhas, acondicionado em caixa de DVD Slim (19 x 14 cm x 7 mm).

Dissertação (mestrado acadêmico) - Universidade Estadual do Ceará, Centro de Ciências e Tecnologia, Mestrado Acadêmico em Ciência da Computação, Fortaleza, 2017.
Área de concentração: Engenharia de Software.
Orientação: Prof. Dr. Paulo Henrique Mendes Maia .

1. Modelos de comportamento. 2. Cenários implícitos. I. Título.

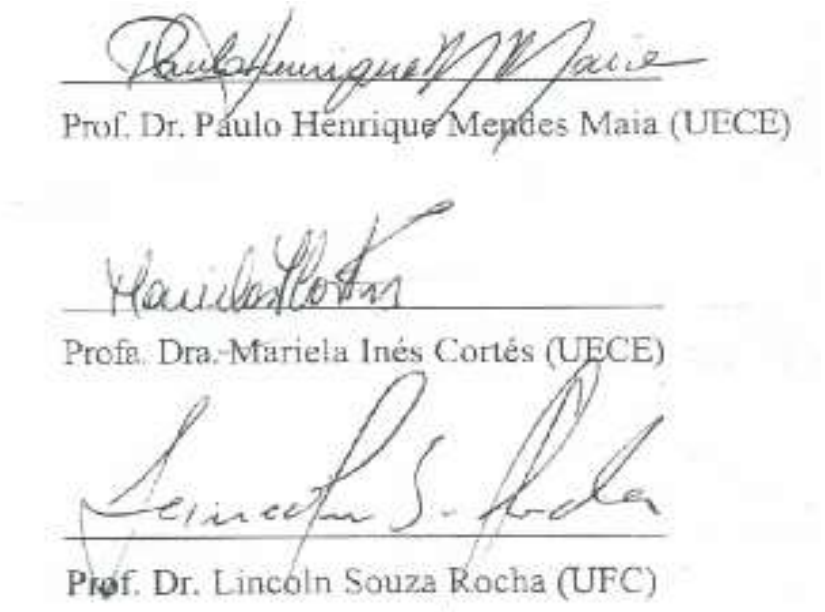
AMANDA SOUZA DA SILVA

UMA ABORDAGEM INTERATIVA E SEMI-AUTOMÁTICA PARA A DETECÇÃO E REMOÇÃO DE CENÁRIOS IMPLÍCITOS EM MODELOS COMPORTAMENTAIS USANDO RASTROS DE EXECUÇÃO

Dissertação apresentada ao Curso de Mestrado Acadêmico em Ciência da Computação do Programa de Pós-Graduação em Ciência da Computação do Centro de Ciências e Tecnologia da Universidade Estadual do Ceará, como requisito parcial à obtenção do título de mestre em Ciência da Computação. Área de Concentração: Ciência da Computação

Aprovada em: 21/ 08/ 2017

BANCA EXAMINADORA



Prof. Dr. Paulo Henrique Mendes Maia (UECE)

Profa. Dra. Mariela Inés Cortés (UECE)

Prof. Dr. Lincoln Souza Rocha (UFC)

A Francisca Almino da Silva (*in memorian*)

AGRADECIMENTOS

Primeiramente a Deus pela sua infinita misericórdia, e por ter me concedido a graça e oportunidade de cursar o mestrado. Agradecer por sempre mostrar a solução para os momentos difíceis. A Ele toda glória e louvor.

Aos meus pais, irmã e noivo, pela paciência, incentivo e apoio em todos os momentos.

Ao meu orientador professor Paulo Henrique Maia por toda instrução, paciência e motivação. Agradecer por todo o acolhimento desde do inicio, e por ser mais que um professor, e sim um verdadeiro amigo. Obrigado pelas constantes demonstração de sabedoria e humildade. O senhor tem toda minha admiração.

Ao grupo GESAD que sempre esteve disponível para ajudar em todos os momentos. Em especial ao Bruno, Yan, Davi e Lucas pelos incansáveis auxílios técnicos.

Aos meus amigos Vanessa, Janaide, Gisele, Marcos, Camila, Marcelo, Rodrigo, entre outros que fiz durante esta fase de minha vida, por todo apoio, horas de estudo e amizade.

Ao Hitalo Nascimento por ser o maior incentivador para que eu pudesse vir cursar o mestrado em Ciências da Computação.

A esta universidade, seu corpo docente, direção e administração que oportunizaram a janela que hoje vislumbro um horizonte superior. À CAPES pelo fomento.

Agradeço a todos os professores por me proporcionar o conhecimento não apenas racional, mas a manifestação do caráter e afetividade da educação no processo de formação profissional, por tanto que se dedicaram a mim, não somente por terem me ensinado, mas por terem me feito aprender. Em especial á professora Mariela pelas indicações, dicas, e correções, que contribuíram para o desenvolvimento deste trabalho.

“Os dois dias mais importantes da sua vida são: o dia em que você nasceu, e o dia em que você descobre o porquê.”

(Mark Twain)

RESUMO

A modelagem comportamental acontece, usualmente, durante a fase inicial de desenvolvimento software e tem a finalidade de representar o comportamento esperado do sistema. Entretanto, sistemas evoluem e essa evolução deve ser refletida no modelo, o que muitas vezes não acontece, ou a fase de modelagem é omitida e o sistema é desenvolvido e evolui sem nenhum modelo que represente seu comportamento. Com base nessa problemática, a extração de modelos surge como uma alternativa para a criação e evolução de modelos a partir de um sistema já implementado. O contexto deste trabalho encontra-se na extração dinâmica de modelos, mais precisamente, na construção de modelos a partir de rastros de execução (*traces*). Apesar de existirem diferentes técnicas de extração, não é possível garantir que o modelo gerado seja isento de comportamentos que não estão de acordo com a implementação do sistema, chamados aqui de cenários implícitos. Este trabalho apresenta uma nova abordagem para a detecção e remoção de cenários implícitos em modelos de comportamento representados como Labelled Transition Systems (LTS). A abordagem é dividida em três fases, compreendendo detecção de cenários implícitos, análise dos mesmos por um usuário especialista no sistema, e refinamento do modelo. A abordagem é implementada na ferramenta de modelagem LoTuS. Através da aplicação em três exemplos de sistemas, foi possível verificar que a abordagem funcionou satisfatoriamente, tanto detectando os cenários implícitos existentes, quanto gerando novos modelos com comportamentos corretos de acordo com o usuário.

Palavras-chave: Modelos de comportamento. Cenários implícitos.

ABSTRACT

Behavioral modeling usually occurs during the initial software development phase and is intended to represent the expected behavior of the system. However, systems evolve and this evolution must be reflected in the model, which often does not happen, or the modeling phase is omitted and the system is developed and evolves without any model that represents its behavior. Based on this problem, the extraction of models emerges as an alternative for the creation and evolution of models from an already implemented system. The context of this work lies in the dynamic extraction of models, more precisely in the construction of models from traces of execution (textit traces). Although there are different extraction techniques, it is not possible to guarantee that the generated model is free of behaviors that are not in agreement with the implementation of the system, called here implicit scenarios. This paper presents a new approach for the detection and removal of implicit scenarios in behavior models represented as Labelled Transition Systems (LTS). The approach is divided into three phases, including detection of implicit scenarios, analysis of them by a system expert, and refinement of the model. The approach is implemented in the LoTuS modeling tool. Through the application in three systems examples, it was possible to verify that the approach worked satisfactorily, either detecting the existing implicit scenarios or generating new models with correct behaviors according to the user.

Keywords: Behavior models. Implied scenarios.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE QUADROS

LISTA DE ABREVIATURAS E SIGLAS

FSA	Autômato de Estados Finitos
LTS	Labelled Transition System
LTSE	LTS Extractor
MSCs	Message Sequence Charts
UML	Unified Modeling Language
WBA	Web-Based Business Applications

LISTA DE SÍMBOLOS

Act	Conjunto universal de ações observáveis
τ	Ação interna de um subsistema
S	Conjunto de estados finitos
A	Conjunto de rótulos
Γ	Transições rotuladas entre os estados
π	Estado de erro
ε	Estado final
q	Estado inicial

SUMÁRIO

1 INTRODUÇÃO

O desenvolvimento de um *software* não é considerado uma tarefa trivial, pois envolve diversas ferramentas e processos para a criação de sistemas. Para minimizar essa complexidade, a Engenharia de *Software* apoia o desenvolvimento incluindo técnicas focadas em todos os aspectos da produção de *software*, desde os estágios iniciais da especificação do sistema até sua manutenção, quando o sistema já está sendo usado, não se preocupando apenas com os processos técnicos do desenvolvimento de *software* (SOMMERVILLE, 2011).

Dentre as atividades apoiadas pela Engenharia de *Software*, destaca-se o uso de modelos para facilitar a compreensão dos aspectos estáticos e dinâmicos de um *software*. Modelos são artefatos utilizados para capturar e representar o conhecimento adquirido durante o processo de desenvolvimento de *software* (MAIA *et al.*, 2016). Eles ajudam analistas a entender problemas complexos e projetar suas soluções potenciais por meio de abstração. Portanto, sistemas de *software* podem se beneficiar muito do uso de modelos e técnicas de modelagem (SELIC, 2003).

Modelagem de comportamento é uma técnica que permite desenvolvedores descrever abstratamente e raciocinar sobre o comportamento desejado de um sistema de *software* (UCHITEL; KRAMER; MAGEE, 2004a). Por sua simplicidade, o custo do desenvolvimento de modelos é considerado pequeno quando comparado ao custo de construir um sistema por completo. Além disso, a modelagem traz como benefício, a facilidade de entendimento sobre como o *software* deve se comportar em tempo de execução e a possibilidade de antever possíveis falhas em tempo de projeto (MAGEE; KRAMER, 2006). Conforme Pimentel *et al.* (2014), a visão comportamental fornece a verificação do cumprimento de tarefas que satisfazem os requisitos do *software*.

A modelagem comportamental acontece, usualmente, durante a fase de análise e projeto de *software*, e sua implementação deve refletir o comportamento especificado. Contudo, por muitas vezes a fase de modelagem é omitida e o sistema é desenvolvido e não possui nenhum modelo que represente seu comportamento. Nesse contexto, a extração de modelos (*model extraction*) (HOLZMANN; SMITH, 1999) surge como uma alternativa para a criação e evolução de modelos a partir de um sistema já implementado pois, à medida que o *software* muda, basta gerar novamente os modelos a partir da nova versão do sistema. Segundo Duarte (2013), as abordagens de extração de modelos de comportamento podem ser agrupadas em três categorias, de acordo com o tipo de informação utilizada para construir o modelo: (i) *estática*, na qual o modelo é construído a partir de informações advindas diretamente do código-fonte,

sem a necessidade de execução do sistema; (ii) *dinâmica*, que requer a execução do código para construir modelos através da inferência do comportamento a partir de rastros de execução (*traces*); e (iii) *híbrida*, que usa informações tanto estáticas quanto dinâmicas para a geração do modelo.

O contexto deste trabalho encontra-se na extração dinâmica de modelos, mais precisamente, na construção de modelos a partir de rastros de execução (*traces*). Os *traces* contribuem para a identificação dos possíveis cenários de execução de um dado sistema, representando uma visão parcial do comportamento global do sistema. Um cenário representa a interação entre sistema, *software* e os usuários (SOUZA; MENDONÇA, 2007), e são considerados como descrições evolutivas de situações em um determinado ambiente (LEITE, 1993). Abordagens baseadas em *traces* permitem a filtragem das informações irrelevantes e acrescentam ao modelo dados específicos, por exemplo, sobre entrada e saída que não constam claramente no sistema. Elas possuem a vantagem de analisar o comportamento do sistema sem o acesso ao código fonte, através da inferência dos rastros de execução gravados por monitoramento (DURY; HALLAL; PETRENKO, 2009), pois nem sempre o código-fonte está disponível para uma instrumentação.

Um requisito essencial para qualquer técnica de extração é o fato de que o modelo resultante seja uma representação real do comportamento do sistema (DUARTE, 2013). O desafio consiste em generalizar um comportamento global do sistema a partir de um conjunto de amostras de comportamentos individuais, representados por execuções separadas. Devido a isso, independente do algoritmo de geração de modelos utilizado, as abordagens de extração dinâmica sofrem do problema da incompletude (*incompleteness*), pois é difícil saber quantos e que comportamentos devem ser observados para permitir a inferência do comportamento completo do sistema sob análise (DUARTE, 2013). Consequentemente, o modelo gerado pode conter comportamentos que não estão presentes nos *traces* do sistema, denominados neste trabalho de cenários implícitos (UCHITEL; KRAMER; MAGEE, 2001).

Cenários implícitos podem ser positivos ou negativos. O primeiro caso acontece quando os comportamentos diferentes existem no sistema real, entretanto não estavam representados nas amostras dos *traces* do sistema utilizados para gerar o modelo. Esse problema ocorre, em algumas circunstâncias, devido à arquitetura especificada do sistema não refletir a arquitetura que foi implementada. Isto significa que, se a arquitetura do sistema implementada não fornece componentes com uma visão local bastante rica do que está acontecendo em um nível de sistema, os componentes não podem ser capazes de representar o comportamento do sistema a que se destina (SOUZA; MENDONÇA, 2007). Em outra circunstância, a falha identificada

não representa erro, entretanto pode ser vista como situações inesperadas devido à incompletude de alguma especificação (MUCCINI, 2003).

Por outro lado, cenários negativos representam comportamentos indesejados (GIANNAKOPOULOU, 1999), os quais não existem (ou não deviam existir) no sistema real e, portanto, não devem estar contidos no modelo comportamental. Cenário negativo é definido como uma situação não aceitável que proporciona desvios do comportamento esperado (SOUZA; MENDONÇA, 2007), ocasionando incorreções no modelo comportamental. Um cenário negativo pode comprometer todo o desempenho de um sistema, pois implementar um *software* com base em modelo errado faz com que o sistema perca toda a sua confiabilidade. Esse erros na modelagem podem ser gerados, dentre outro motivos, por imprecisão no algoritmo de inferência do modelo, que não faz uma implementação coerente com o que é pretendido, ou por erros na implementação do sistema, que ocasiona a produção de tais cenários em tempo de execução.

Para minimizar esse problema, algumas abordagens têm sido propostas, sejam elas baseadas em inferência de gramática (ANGLUIN, 1987; LORENZOLI; MARIANI; PEZZè, 2008), análise estatística (COOK; WOLF, 1998), técnicas de reengenharia (SOUZA; MENDONÇA, 2007), abordagens que baseiam-se em UML (CASTEJÓN; BRÆK, 2006), ou a partir do uso de *Message Sequence Charts* Uchitel, Kramer e Magee (2001). Contudo, mesmo assim, não é possível garantir a geração de um modelo sem cenários implícitos. Além disso, nem todas essas abordagens permitem refinar o modelo de modo a remover os cenários implícitos negativos detectados ou, quando o fazem, não levam em consideração a expertise do usuário, o que garantiria uma maior confiabilidade do modelo resultante.

Com base nessa problemática, este trabalho apresenta uma abordagem interativa e semi-automática para a detecção e refinamento de modelos de comportamentos, representados como Labelled Transition Systems (LTS), usando rastros de execução do sistema. A abordagem é dividida em três fases: (i) detecção de cenários implícitos, que compara os *traces* do sistema fornecidos como entrada com *traces* gerados a partir do modelo inicial, gerados utilizando um algoritmo comum na área de testes baseados em modelos; (ii) análise, que utiliza do conceito de *man-in-the-loop* e introduz o usuário especialista no sistema como elemento tomador de decisão sobre os cenários, e (iii) remoção, que remove os cenários escolhidos pelo usuário, gerando um novo modelo refinado. Esse processo acontece iterativamente, até que nenhum cenário negativo seja mais detectado.

1.1 OBJETIVOS

1.1.1 Objetivo Geral

Este trabalho propõe o desenvolvimento de uma nova abordagem para apoiar a detecção e remoção de cenários implícitos em modelos construídos a partir dos *traces* do sistema.

1.1.2 Objetivos Específicos

- Propor um mecanismo de detecção de cenários implícitos em modelos comportamentais.
- Implementar uma técnica voltada para remover de cenários negativos, em que o usuário terá disponível uma interface voltada para facilitar as inferências sobre os rastros identificados.
- Disponibilizar uma ferramenta que implemente a abordagem proposta.
- Realizar exemplos de uso para mostrar a aplicabilidade da abordagem proposta.

1.2 ESTRUTURA DO TRABALHO

Este trabalho encontra-se estruturado da seguinte forma:

O capítulo 2 apresenta a fundamentação teórica contendo um estudo aprofundado na literatura. Inicialmente é apresentada a definição de modelo comportamental. Em seguida os conceitos relativos aos *Labelled Transition System* (LTS) são especificados. São apresentadas também as definições sobre Cenários Implícitos e Extração de Modelos, abordando os principais conceitos, diferenças e fundamentos. Por fim, encontra-se uma seção que explana sobre a ferramenta LoTuS expondo suas particularidades.

O Capítulo 3 traz os principais trabalhos relacionados às ideias centrais desta pesquisa. São apresentados trabalhos que abordam a extração de modelos a partir do código. Logo após, são apresentadas pesquisas relativas ao contexto da síntese de modelos a partir de *traces*. Posteriormente é mencionado alguns dos principais trabalhos referentes a detecção de uso de cenários implícitos. E para finalizar encontra-se uma comparação entre os principais trabalhos relacionados com a presente abordagem.

O Capítulo 4 apresenta a abordagem proposta por este trabalho para detectar e remover cenários negativos em modelo comportamental. Na primeira seção é mostrada uma visão geral do funcionamento da técnica, mostrando todas as etapas necessárias para conduzir o trabalho. Na segunda é exposto o Suporte ferramental da técnica. A terceira apresenta-se a

forma de como é realizado a a detecção dos cenários implícitos. Em seguida é mostrado toda análise por parte do usuário, sobre os rastros detectados. Ao final é explanado a respeito de toda a técnica do refinamento, abordando o funcionamento e metodologia usada.

No Capítulo 5 encontram-se os exemplos de uso da abordagem proposta. Inicialmente, apresentam-se um modelo comportamental de sistema de vendas, em seguida é realizada inferências sobre um aplicativo de serviço médico. E por fim, é apresentado um estudo de caso que real que envolve um sistema de vendas *web*.

Finalmente no Capítulo 6 são apresentadas as considerações finais do trabalho, as limitações da abordagem, bem como as perspectivas de evoluções da atual pesquisa através de possíveis trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo é realizado um estudo mais aprofundado sobre cada conteúdo abordado nesta proposta. Inicialmente é apresentada a definição de modelo comportamental. Em seguida os conceitos relativos aos *Labelled Transition System* (LTS) são especificados. São apresentadas também as definições sobre cenários implícitos e, por fim, apresenta-se a extração de modelos, abordando os principais conceitos e categorias.

2.1 MODELOS COMPORTAMENTAIS

Modelos comportamentais representam o comportamento dinâmico do sistema, incluindo os aspectos relevantes e observáveis para descrever o comportamento do *software*. A modelagem mostra o que ocorre quando o sistema responde a um estímulo no seu ambiente (SOMMERVILLE, 2011). Especificações comportamentais desempenham um papel fundamental no *design*, validação, verificação, teste e manutenção de sistemas (YANG *et al.*, 2006). Os modelos comportamentais podem ajudar os engenheiros a entender os objetos complexos e suas interações (LORENZOLI; MARIANI; PEZZè, 2008), bem como a detectar falhas e outras anomalias (HANGAL; LAM, 2002) (YANG *et al.*, 2006).

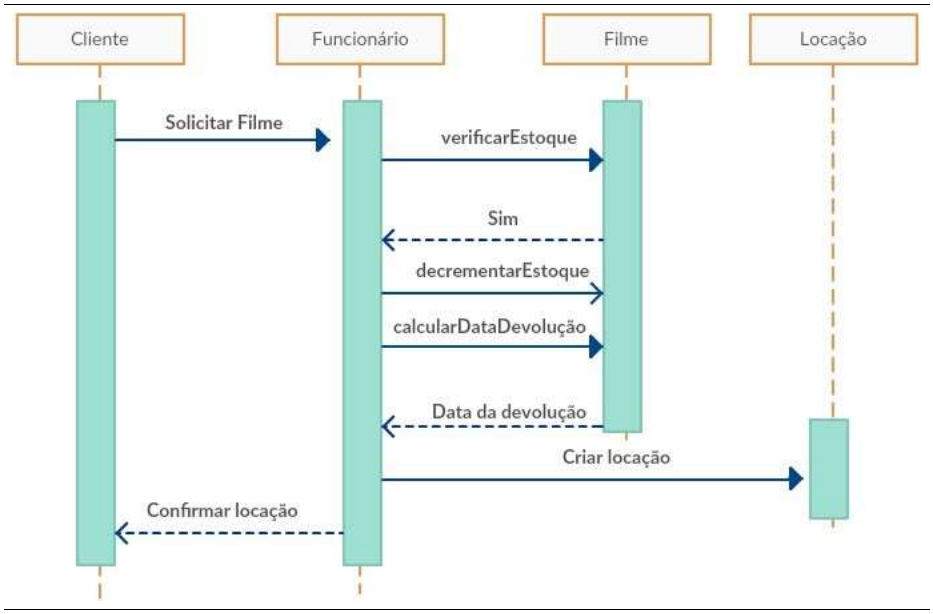
Modelos de comportamento descrevem as abstrações internas de um sistema, definindo suas concepções, com a finalidade de associa-lo adequadamente a um ambiente (CLARKE; WING, 1996). Dentre as aplicações da modelagem está a descrição dos requisitos e a possibilidade de fornecer uma ampla gama de técnicas de validação, tais como, simulações e técnicas baseadas em cenários (UCHITEL; BRUNET; CHECHIK, 2007). Os modelos baseados em especificações definidas nas fases iniciais do processo de desenvolvimento do *software* fornecem todas as possíveis condutas do sistema, uma vez implementado, e torna possível antever falhas no projeto (MAGEE; KRAMER, 2006). Outra possibilidade é a geração de modelos comportamentais após a implementação do sistema, técnica conhecida como extração de modelo (HOLZMANN; SMITH, 1999), na qual este trabalho se baseia.

Modelos de comportamento podem ser representados por notações ou linguagens semi-formais ou formais. Como linguagem semi-formal pode-se citar a UML (Unified Modeling Language), que possui notações voltadas para a descrição de sistemas computacionais (BOOCH; RUMBAUGH; JACOBSON, 2006) e fornece padrões e recursos para o desenvolvimento do projeto de *softwares*. A modelagem UML é baseada em dois conjuntos de representações: estrutural e comportamental (SYSTEMS, 2007) (MEDVIDOVIC *et al.*, 2002).

A modelagem estrutural descreve a arquitetura estática do sistema através da utilização de diagramas de classes, objetos, interfaces, desenvolvimento, pacotes e componentes (VITAL; VITAL, 2015). Os diagramas comportamentais da UML podem capturar as interações das execuções de um sistema ao longo do tempo. Portanto, permitem visualizar, especificar e documentar os aspectos dinâmicos de um sistema, por exemplo, o fluxo de mensagens ao longo do tempo e a movimentação física de componentes em uma rede. Exemplos de diagramas dinâmicos são o digrama de casos de uso, sequência, máquina de estados, comunicação, atividades, visão geral de interação e temporização (GROUP *et al.*, 2012).

Como exemplo de diagrama de comportamental da UML, o diagrama de sequência (Figura 1) mostra as interações entre um cliente que deseja alugar filme e o funcionário de uma locadora. Primeiramente, o cliente faz a solicitação do filme ao funcionário. A seguir, o funcionário verifica se o filme solicitado tem no estoque, retorna ao cliente que o filme esta disponível, e calcula a data de devolução. Após o termino dos devidos procedimentos de locação, o funcionário confirma com o cliente o aluguel do filme.

Figura 1 – Exemplo de diagrama de sequência



Fonte – Elaborado pelo autor

Apesar dos diagramas comportamentais UML serem úteis para descrever o comportamento do *software*, eles carecem de formalização que permita a verificação de propriedades, aliado a uma ferramenta automatizada que dê suporte à análise desejada. Especificações formais ajudam a detectar e resolver ambiguidades, omissões e inconsistências, uma vez que o formalismo disponibiliza uma precisão matemática na análise de programas que fornecem a

possibilidade de certificar a corretude do *software*, algo que o teste e a revisão informal não conseguem. O objetivo maior dos métodos formais é aumentar a qualidade do sistema ao tornar os requisitos, projeto e implementação mais explícitos e, portanto, fazendo com que os defeitos sejam facilmente detectados (GEER, 2011).

A modelagem formal de comportamento é geralmente descrita, dentre outros formalismos, conforme máquinas de estados finitos, que fornecem uma teoria matemática concisa para apoiar a análise e verificação das propriedades de sistemas e possui uma semântica intuitiva do comportamento do *software* que permite a visualização, hierarquia e abstração (CHAKI *et al.*, 2004).

Os Sistemas de Transição Rotuladas (*Labelled Transition Systems* - LTS) são um formalismo frequentemente utilizado para modelar comportamento de *software*, sendo sua estrutura composta por um conjunto de estados e transições rotuladas que indicam que a mudança de estado se deve pela ocorrência da ação que rotula a transição entre eles (UCHITEL; KRA- MER; MAGEE, 2004a). Neste trabalho é utilizado o LTS como formalismo de modelagem de comportamento do sistema, o qual será melhor descrito a seguir.

2.2 LABELLED TRANSITIONS SYSTEMS

LTSs (KELLER, 1976) são adequados para fazer a modelagem e o raciocínio do comportamento do sistema (UCHITEL; BRUNET; CHECHIK, 2007), onde cada modelo refere-se a um componente do sistema que interage com os demais componentes através de ações compartilhadas. O modelo LTS pode representar uma função em qualquer nível de abstração, ou seja, desde o diagrama de contexto, que retrata todo o sistema como uma única função, até o nível mais baixo, em que as funções são atômicas, e não podem mais ser decompostas em outras funções elementares (DURY; HALLAL; PETRENKO, 2009). O sistema de transição é usado com a finalidade de modelar o comportamento do processo de comunicação em um programa concorrente (MAIA, 2011).

Definição LTS: De acordo com Keller(1976), um LTS é definido da seguinte forma: seja *Estados* o conjunto universal de estados, *Act* o conjunto de ações observáveis, e $Act_\tau = Act \cup \{\tau\}$, onde τ é usado para denotar uma ação interna para um subsistema, e, portanto, não observável pelo seu ambiente. Um LTS é formalmente uma estrutura $P = (S, A, \Gamma, q)$, em que:

- $S \subseteq$ *Estados* é um conjunto de estados finitos;
- $A = \alpha(P) \cup \{\tau\}$, quando $A = \alpha(P) \subseteq Act$, é o conjunto de rótulos que denota a comunica-

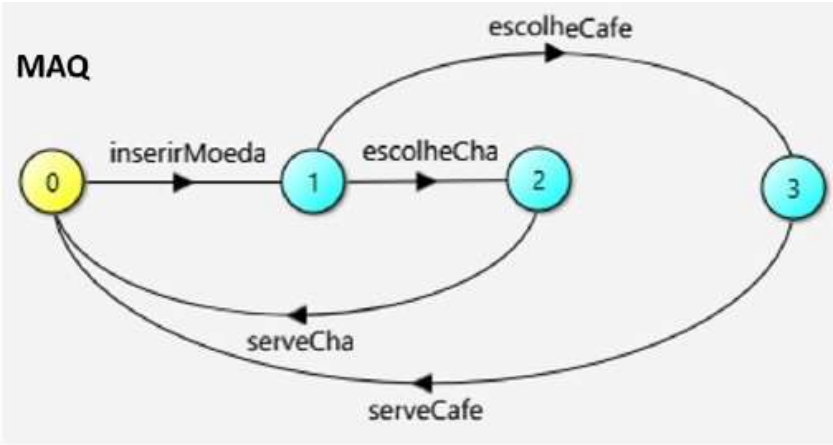
ção do alfabeto de P;

- $\Gamma \subseteq (S \cup \{\pi, \varepsilon\} \times A \times S)$, define as transições rotuladas entre os estados, onde nenhuma transição é originada a partir do *estado de erro* π ou o *estado final* ε;
- $q \in S$ é o estado inicial.

Para exemplificar melhor, considere o seguinte modelo LTS de uma máquina de venda de bebidas MAQ (Figura 2):

`MAQ=({0,1,2,3},{inserirMoeda, escolheCha, escolheCafé, serveCha, serveCafe}, {(0,inserirMoeda,1), (1,escolheCha,2), (1,escolheCafe,3), (2,serveCha,0), (3,serveCafe,0)},0).`

Figura 2 – LTS de uma máquina de venda de bebidas



Fonte – Adaptado de (Maia, 2011)

A Figura 2 apresenta um LTS que representa o comportamento de uma máquina de venda de bebidas. A partir do estado inicial 0, o sistema aguarda a inserção de uma moeda. Quando a moeda é inserida, o modelo passa para o estado 1, indicando que o usuário pode escolher entre as opções chá ou café. Ao realizar as ações chá e café, o modelo vai para o estado 2 e 3, respectivamente, o que apresenta a preferência do usuário entre as bebidas. De ambos os estados, a bebida correta é servida e o modelo retorna ao seu estado inicial, significando que a máquina de venda bebidas está pronta para aceitar uma outra moeda.

Uma execução de um LTS é uma sequência de estados e rótulos obtidos a partir do estado inicial do LTS. Por exemplo, a sequência <0, inserirMoeda, 1, escolheCha, 2, serveCha> representa uma execução do LTS da Figura 2.

Um caminho (*path*) é a sequência de estados resultantes da exclusão dos rótulos de uma execução. Por exemplo, considerando o modelo da Figura 2, a partir da execução: <

0, inserirMoeda, 1, escolheCha, 2, serveCha, 0>, tem-se o seguinte caminho: <inserirMoeda, escolheCha, serveCha> e <0,1,2,0>.

Um rastro (*trace*) é determinado como uma sequência observável de ações definidas por um caminho que se origina no estado inicial *q*, ou seja, é uma sequência finita de ações observáveis que rotulam as transições que são feitas a partir de seu estado inicial (BIANCULLI; GIANNAKOPOULOU; PASAREANU, 2011). Um rastro representa um possível comportamento do sistema. Por exemplo considerando a execução <0, inserirMoeda, 1, escolheCha, 2, serveCha, 0>, o trace correspondente é <inserirMoeda, escolheCha, serveCha>.

2.3 CENÁRIOS IMPLÍCITOS

Cenários, em geral, podem ser considerados como descrições evolutivas de situações em um determinado ambiente (LEITE, 1993), compostos por um conjunto ordenado de interações entre seus participantes. Um cenário descreve como um ou mais componentes do sistema e o usuários simultaneamente interagem de forma a proporcionar um certo conjunto de funcionalidades (SOUZA; MENDONÇA, 2007).

Várias abordagens têm sido propostas na literatura para a representação de cenários, incluindo protótipos e gráficos de sequência (ROBERTSON; ROBERTSON, 1999). Cada cenário fornece uma especificação parcial, que quando combinado com todos os outros cenários, contribui para a especificação geral do sistema (UCHITEL; BRUNET; CHECHIK, 2007). Todavia, em algumas circunstâncias, a arquitetura especificada do sistema não reflete a arquitetura que é ou será implementada, pois o comportamento global de um sistema nem sempre é totalmente refletido em suas especificações de comportamento parciais (UCHITEL; KRAMER; MAGEE, 2004b). Por isso, independentemente de sua simplicidade e representação gráfica intuitiva, especificações baseadas em cenários possuem limitações.

Por esse motivo, podem ocorrer combinações imprevistas de alguns componentes no momento de interagir com seus cenários participantes, podendo levar a padrões de interação inesperados que não fazem parte de qualquer cenário especificado do sistema. Tais padrões foram chamados de cenários implícitos (ALUR; ETESSAMI; YANNAKAKIS, 2003). Define-se cenários implícitos neste trabalho como os caminhos de comportamentos que podem ser extraídos a partir do modelo de máquina de estado, mas não existe nas especificações iniciais do sistema (MUCCINI, 2003).

A presença de cenários implícitos em uma especificação baseada em cenários indica

lacunas que são o resultado de se especificar o comportamento de um sistema a partir de uma perspectiva local, porém esperando o comportamento a ser fornecido de forma global (UCHITEL; KRAMER; MAGEE, 2001). Isso significa que, se a arquitetura do sistema implementado não fornece componentes com uma visão local bastante rica do que está acontecendo em um nível de sistema, os componentes não podem ser capazes de representar o comportamento do sistema a que se destina (SOUZA; MENDONÇA, 2007). Cenários implícitos nem sempre são considerados comportamentos errados, às vezes eles podem ser visto como situações inesperadas devido à incompletude de alguma especificação (MUCCINI, 2003).

Nesse contexto, cenários implícitos podem ser definidos de duas maneiras, a primeira como os *traces* que não estavam contidos nos rastros inciais do sistema, no momento da geração da modelagem comportamental, ou apenas especificações que pertencem ao sistema, porém não foram implementadas na fase de desenvolvimento do *software* (GIANNAKOPOULOU, 1999), isto é, cenários positivos. Enquanto o cenário negativo são caminhos que não coincidem com as atribuições do sistema, resultando em erros(SOUZA; MENDONÇA, 2007). Essas falhas fazem com que a modelagem perca sua confiabilidade.

2.4 EXTRAÇÃO DE MODELOS

A extração de modelos (HOLZMANN; SMITH, 1999) consiste em um processo para construir automaticamente um modelo de comportamento a partir de uma implementação existente. A ideia principal é criar o modelo a partir da implementação, assim ele irá refletir exatamente o que o sistema faz e, por conseguinte, qualquer propriedade verificada no modelo será também uma propriedade preservada na implementação. Além disso, não é necessário conhecer a linguagem da modelagem, pois o modelo é automaticamente construído a partir do código. Por isso, essa técnica proporciona um processo para criação de um modelo diretamente aceito por uma ferramenta (DUARTE, 2013).

Conforme Duarte (2013), as abordagens existentes para a extração de modelo de comportamento podem ser divididas em três principais categorias, baseado no tipo de informação que eles usam para construir o modelo, conforme apresentado a seguir.

1) Estático: A abordagem constrói modelos com base nas informações estáticas do sistema obtidas diretamente do código fonte. Consequentemente, o modelo é construído sem a necessidade de executar o sistema. Modelos nesta categoria descrevem todas as prováveis execuções do sistema, garantindo a integralidade. Entretanto, não é garantida a exatidão, dado

que o modelo inclui todos os possíveis comportamentos, porém alguns podem não ser viáveis quando o sistema é executado. Esse problema surge porque o modelo não contêm informações que podem influenciar quais caminhos são tomados durante uma execução. Então, podem produzir falsos alarmes, alertando para um possível comportamento inválido que não é realmente inviável.

Devido a essa limitação, trabalhos em abordagens estáticas tem-se concentrado sobre a forma de eliminar falsos alarmes. Nesta direção, ferramentas como SLAM (BALL; RAJAMANI, 2002) e BLAST (HENZINGER *et al.*, 2003) aplicam uma combinação de programas *boolean*, onde todas as variáveis do sistema são convertidos em variáveis booleanas correspondentes aos valores das variáveis originais, sendo feito uma execução simbólica (KING, 1976). As ferramentas permitem que o sistema seja executado calculando quais caminhos são habilitados, levando em consideração os valores simbólicos das variáveis *boolean*. Quando os modelos são usado para verificar modelos, essas ferramentas podem empregar um técnica automática de refinamento que podem incluir novas variáveis, toda vez que um alarme falso ocorrer, eliminando assim automaticamente o caminho inviável. O funcionamento das ferramentas são baseadas em técnicas denominada *Counterexample - Guided Abstraction Refinement* (KROENING; GROCE; CLARKE, 2004), que tem sido utilizado em muitos instrumentos e técnicas.

2) Dinâmica: Constroem modelos a partir da inferência das execuções observadas. O tipo de informação utilizada para inferir sobre o modelo varia, dependendo do tipo de extração utilizada. Ela pode estar relacionada à ocorrência de certos eventos, às modificações de valores de variáveis, ao estado de execução, dentre outros. As informações são recolhidas durante o monitoramento das execuções com o auxílio de ferramentas ou através da inserção de anotações no código de saída dos dados necessários. O problema com estas abordagens é quando existe a necessidade de generalizar as amostras locais das execuções, para criar uma descrição global do comportamento do sistema.

3) Híbrido: Devido aos problemas das abordagens estáticas e dinâmicas, Ernst (2003) apresentou um abordagem híbrida na qual o modelo pode ser construído combinando informações estáticas e dinâmicas.

Em (NIMMER; ERNST, 2002), a abordagem híbrida é baseada em duas ferramentas: primeiramente utiliza-se uma que infere um possível modelo com base em informações dinâmicas, e outra que verifica estaticamente o modelo inferido para confirmar ou refutar os comportamentos incluídos. Com o uso de informações estáticas pode-se eliminar comportamentos falhos inferidos durante o processo dinâmico. Baseado em (DUARTE; KRAMER; UCHITEL,

2008), o conceito de contextos é proposto como uma forma de encapsular informações estáticas e dinâmicas em um único tipo de informação, excluindo a necessidade de trabalhar com duas ferramentas separadas. Basicamente, um contexto corresponde a um estado do sistema que é uma combinação de informação de controle de fluxo (bloco de código associados as condições de execução) e de valores de atributos (variáveis globais).

2.5 CONCLUSÕES DO CAPÍTULO

Neste capítulo foram apresentados os conceitos que embasam o presente estudo. Inicialmente foi esclarecido que modelos comportamentais representam o comportamento dinâmico do sistema e incluem os aspectos relevantes e observáveis para descrever o comportamento de um *software*. Apresentou-se alguns tipos de modelos, como os diagramas UML que são úteis para descrever o comportamento do *software*, porém carecem de formalização que permita a verificação de propriedades. E explanou sobre os conceitos que envolvem especificações formais, pois ajudam a detectar e resolver ambiguidades, omissões e inconsistências, uma vez que disponibiliza uma precisão matemática para análise de programas que fornecendo a possibilidade de certificar a corretude do *software*, algo que o teste e a revisão informal não conseguem.

O conceito de *Labelled Transition Systems* (LTS) é exposto em seguida, onde explora-se suas características. LTSs é um formalismo frequentemente utilizado para modelar comportamento de sistemas, sendo sua estrutura composta por um conjunto de estados e transições rotuladas, no qual indicam mudança de estado que se deve pela ocorrência da ação que rotula a transição entre eles. Cada modelo LTS refere-se a um componente do sistema que interage com os demais componentes através de ações compartilhadas. O modelo LTS pode representar uma função em qualquer nível de abstração, ou seja, desde o diagrama de contexto, que retrata todo o sistema como uma única função, até o nível mais baixo, em que as funções são atômicas, e não podem mais ser decompostas em outras funções elementares.

Cenários implícitos foram explanados na seção seguinte, explorando suas propriedade. Primeiramente foi expresso que cenários são considerados como descrições evolutivas de situações em um determinado ambiente, compostos por um conjunto ordenado de interações entre seus participantes. Um cenário descreve como um ou mais componentes do sistema e o usuários simultaneamente interagem de forma a proporcionar um certo conjunto de funcionalidades. Com base nesse conceito, cenários implícito denominam-se como combinações imprevistas de alguns componentes no momento de interagir com seus cenários participantes, podendo levar a padrões

de interação inesperados que não fazem parte de qualquer cenário especificado do sistema. Esses cenários podem ser positivos ou negativos. Positivos não representam erros, apenas atribuições não implementadas na fase de desenvolvimento do sistema. Cenário negativo, são considerados erros ou falhas na abordagem, e dessa forma a modelagem perde a confiabilidade.

Foi apresentada uma técnica denominada extração de modelos, que consiste em um processo para construir automaticamente um modelo de comportamento a partir de uma implementação existente, pois assim ele irá refletir exatamente o que o sistema faz e, por conseguinte, qualquer propriedade verificada no modelo será também uma propriedade preservada na implementação. As abordagens existentes para a extração de modelo de comportamento podem ser divididas em três principais categorias, baseado no tipo de informação que eles usam para construir o modelo: estática, dinâmica e híbrida.

3 TRABALHOS RELACIONADOS

Este capítulo descreve alguns trabalhos que possuem relação com a presente pesquisa. Encontra-se dividido nos seguintes temas: extração de modelos, que explora as abordagens estáticas, dinâmicas e híbridas; a detecção e uso de cenários implícitos; e refinamento de modelos.

3.1 EXTRAÇÃO DE MODELOS

3.1.1 Extração Estática

Com intenção de aperfeiçoar o processo de inferência, várias técnicas usam a extração de modelos a partir código para aplicar uma variação do algoritmo $L*$ proposto por Angluin (1987). O intuito é inferir uma linguagem utilizada para descrever modelos com base em exemplos de frases nesse idioma. Neste caso, a linguagem é o conjunto de comportamentos possíveis do sistema e as frases são rastros de execução. O principal requisito do algoritmo é que deve haver algum tipo de "professor" para orientar o processo, que é capaz distinguir frases válidas e inválidas no idioma desejado.

Uma técnica semelhante é empregada em Cook e Wolf (1998), baseada em uma inferência gramatical concentrada sobre os métodos que foram executados. A característica adicional dessa técnica é o uso de uma análise estatística sobre as execuções de exemplos como um meio de eliminar de antemão todos os comportamentos que não são frequentes. Assim, a premissa é que comportamentos frequentes são válidos. Por exemplo, se uma (chamada de método) ação B ocorre frequentemente depois de uma ação A nos *traces*, poderia ser inferido que há uma dependência entre eles de tal modo que B sempre ocorre depois de A. Esta análise é parametrizada de modo que um limite pode ser definido para determinar exatamente quantas vezes uma dependência entre as ações tem de aparecer nos *traces* para ser assumida como válida.

Em (CORBETT *et al.*, 2000) os autores usaram uma ferramenta para verificar modelos a partir do código-fonte java, integrada a um conjunto ferramentas para torná-la extensível, direcionada à análise de programas e componentes de transformação, denominada Bandera. A técnica permite a extração automática de modelos de estados-finitos diretamente do código-fonte do programa. A ferramenta recebe como entrada o código java e ao final do processo gera um modelo de programa na mesma linguagem inicial, mapeando a verificação das saídas e o retorno para o código-fonte original. Bandera usa uma arquitetura baseada em componentes para a extração de modelos projetados para maximizar a escalabilidade, exibilidade,

e extensibilidade, reduzindo as barreiras de aplicação do modelo verificado para *software*.

Em (LORENZOLI; MARIANI; PEZZè, 2008), uma extensão da inferência gramatical é proposta com a adição dos valores dos parâmetros de cada chamada de método de modo que é possível distinguir os diferentes comportamentos do sistema quando certos parâmetros são fornecidos. A metodologia concentrou-se na geração de modelos, e entre a relação dos valores de dados e a interações dos componentes. Apresentou-se GK-tail, uma técnica para gerar automaticamente máquinas de estado finito estendidas (EFSMs) após a interação de *traces*.

3.1.2 Extração Dinâmica

Com base na definição de *trace* Maoz (2008), objetivou-se a criação da uma ferramenta computacional denominada S2A, que fornece a geração de código do comportamento a partir de especificações baseadas em cenários declarativos visuais. Com base nas aplicações o trabalho desenvolveu métodos para analisar os rastros, conjuntamente com vários tipos de captações de aspectos do *software* ao longo dos mesmos modelos. O resultado obteve duas vantagens: o código é gerado automaticamente, onde pode ser monitorado a partir dos modelos, e o código do sistema investigado é alheio aos modelos observados. A visualização e interação suportada pelo *trace* permite que um usuário humano possa explorar e analisar os *traces* complexos a partir dos modelos, que de maneira manual seria muito difícil de tratar.

Para parametrizar autômatos finitos, Dury, Hallal e Petrenko (2009) apresentaram uma abordagem que captura o fluxo de controle das aplicações e as variações dos seus dados, a partir de aplicações negociais baseadas na *web*. A pesquisa trabalha com modelagens Web-Based Business Applications (WBA) voltada para a validação de mineração dos dados. A abordagem utilizou a inferência do modelo comportamental de uma aplicação de negócios, com base nos rastros memorizados por monitoramento da execução da aplicação, enquanto era utilizado por diferentes usuários. Os resultados indicaram que os modelos obtidos podem ser próximos do comportamento da aplicação, usando rastros de execução gerados aleatoriamente. Contudo, a qualidade dos modelos permanece diretamente relacionada ao número e à diversidade dos rastros recolhidos.

Krka *et al.* (2010) realizaram uma pesquisa que tem por finalidade melhorar o estado da arte na geração de modelos comportamentais gerados de forma dinâmica. A abordagem utiliza *traces* de execução e programas de inferência para obter um modelo no mesmo nível dos objetos que descrevem todas as sequências das execuções. O método tem duas fases, no

qual inicialmente é obtido um (Autômato de Estados Finitos) (FSA) com base nos modelos comportamentais. A ideia principal da fase 1 é simular, a partir do autômato, os *traces* do método de invocação dinâmica, para ser obtido informações internas do sistema. Na fase 2, é usada a extração dinâmica sobre os *traces* recolhidos, com a intenção de refinar as informações. Os autores citam que que a abordagem pode ser diretamente aplicada sobre contexto da análise estática com *traces* dinâmicos para melhorar a detecção de informações estáticas.

Braun (2013) propôs uma abordagem direcionada para simplificação *traces* longos, a partir da filtragem dos componentes do *software* de baixo nível. Utilizou-se informações estáticas para identificar e remover pequenos componentes do *software* partir do rastreamento. Implementou-se o algoritmo denominado de *utility method*, no qual a metodologia direcionada em levar a possibilidade do componente ser acessado a partir de vários locais dentro do sistema, e assim fornecer aos *designers* a oportunidade de obter informações visuais do sistema. A aplicação mede o nível de importância de um método, cujo o calculo é feito usando diferentes combinações de variáveis dinâmicas e estáticas. Os métodos com valores elevados são detectados e removidos de forma iterativa. Com essa eliminação, os *traces* menores são visualizados usando a notação de mapas de caso de uso (UCM), onde é uma linguagem de cenário usada para especificar e explicar o comportamento de sistemas complexos (BUHR, 1998). Ao fazer isso, pode-se identificar o algoritmo que gera uma UCM mais próximo do modelo mental dos *designers*.

Em (MIHANCEA; MINEA, 2014) foi apresentado o JMODEX, uma ferramenta para extrair automaticamente modelos comportamentais de aplicações *web*, capturando os comportamentos de cada componente da aplicação sob a forma de uma máquina de estado finito. O modelo é expresso na linguagem ASLAN ++ e pode ser utilizado com um verificador de modelos para analisar as propriedades de segurança. A ferramenta pode extrair modelos comportamentais a partir de aplicações *web* pequenas, nas quais os modelos extraídos podem ser utilizados para identificar as vulnerabilidades de segurança.

Hendriks *et al.* (2016) definiram duas técnicas voltadas para a análise e execução de *traces*. A primeira abordagem chamada de análise de caminho crítico, pode identificar atividades e recursos, com o intuito de contribuir para um melhor desempenho do *software*, isto é, um tempo de execução total mais baixo do sistema. A análise também pode ser utilizada para ajustar o modelo, considerando toda a estrutura da execução. A segunda técnica, denominada distância do rastreio, é utilizada para visualizar e destacar as diferenças entre *traces*. As técnicas foram implementadas na ferramenta *Trace visualization tool* (MALONY; HAMMERSLAG; JABLONOWSKI, 1991) capaz de apresentar grandes conjuntos de atividades sobre recursos (e

dependências entre eles) do sistema. Os métodos foram criados para facilitar a interpretação dos rastros de execução, permitir aos *designers* encontrarem pontos de erros nos sistemas e comparar de forma eficiente o comportamento do sistema em um nível estrutural para fins de ajustes e validação.

Krämer, Brandt e Borchers (2016) criaram uma ferramenta de suporte que fornece uma análise dinâmica usando *traces* gerados de uma execução para sugerir alterações de documentação e testes de unidade, denominada de Vesta. A ideia principal é oferecer atualizações para a documentação imediatamente após execuções manuais, feitas por desenvolvedores. O objetivo da ferramenta é aumentar a precisão significativamente da documentação, realizando manutenções das informações, e a verificação automática como um benefício adicional do protótipo.

3.1.3 Extração Híbrida

Para mostrar como as informações provindas de contextos são capazes de orientar um processo de construção de modelos representados como (LTSs), Duarte, Kramer e Uchitel (2006) sugeriram um novo procedimento voltado à extração de modelos comportamentais automáticos, combinando informações estáticas e dinâmicas. A ideia explora e unifica as definições de contextos descritos pela avaliação de controle e valores de atributos, e cria uma estrutura alinhada que facilita a extração de informações e as relações entre ações do sistema. A implementação foi parcialmente executada pela ferramenta LTS Extractor (LTSE) (DUARTE; KRAMER; UCHITEL, 2006). O estudo indicou que o processo de extração em modelos comportamentais gera boas informações sobre a parte interna do sistema, e pode ser usado para a análise do comportamento e verificação das propriedades a que o *software* está vinculado.

O contexto deste trabalho tem similaridade com a ideia da pesquisa abordada por Krka *et al.* (2010), que utiliza rastros de execução para obter um modelo no mesmo nível dos objetos que descrevem todas as sequências das execuções. Nesta pesquisa, os rastros são utilizados no intuito de gerar um modelo comportamental coerente com as especificações de um determinado sistema.

A proposta de (MIHANCEA; MINEA, 2014) apresentou uma ferramenta voltada para a extração automática de modelos comportamentais de aplicações *web*, denominada JMO-DEX. A abordagem proposta nesta pesquisa não inicia na fase de extração, ela trabalha com os rastros e modelo comportamental gerados dinamicamente.

3.2 DETECÇÃO DE CENÁRIOS IMPLÍCITOS

Uchitel, Kramer e Magee (2001) construíram um algoritmo que gera modelos comportamentais a partir de Message Sequence Charts (MSCs) de alto nível, em que descreve as execuções internas do sistema para uma especificação baseada em *Labelled Transitions Systems*. A técnica é integrada à ferramenta LTSA, e pode detectar e fornecer *feedback* sobre a existência de cenários implícitos. Outra contribuição do estudo refere-se à construção de um *framework* que sintetiza a modelagem para especificação baseada em cenários, fornecendo um critério para avaliar e classificar se uma especificação possui comportamentos implícitos.

Em (CASTEJÓN; BRÆK, 2006) foi apresentado uma abordagem de especificação do serviço baseado em diagramas UML 2.0 que lida com a detecção de cenários implícitos. A pesquisa detecta cenários implícitos a partir de especificações de serviços baseados em colaboração. A detecção é feita por análise estática das sub-sequências individuais, que foram verificadas separadamente, de modo que a complexidade seja linear, para trabalhar em um alto nível de abstração. O método de detecção de cenário implícito proposto demonstrou que pode ser utilizado tanto para descrever explicitamente os recursos dependentes do sistema, quanto para a análise e compreensão da simultaneidade em interfaces.

O trabalho de Souza e Mendonça (2007) apresentou um ambiente de engenharia reversa para suportar a extração e detecção de cenários implícitos a partir de informações providas de sistemas, capturadas durante as execuções das aplicações. Desenvolveu-se um conjunto de ferramentas e outras reutilizadas especificamente com o objetivo de apoiar os passos necessários do processo, como instrumentação da aplicação, extração de cenários e abstração a partir dos rastros de execução recolhidos com uso da ferramenta LTSA-MSC (UCHITEL *et al.*, 2003), detecção e visualização de potenciais cenários implícitos baseados no conjunto de cenários extraídos. O ponto principal encontra-se em conceder aos desenvolvedores ganhos aplicando o conceito de cenários implícitos, anteriormente privativo apenas às fases iniciais do ciclo de vida de *software*, para apoiar atividades de compreensão e teste de sistemas existentes, e na descoberta de eventos indesejados a partir de informações coletadas dinamicamente durante a execução de uma aplicação concorrente existente.

Para analisar o impacto da presença de cenários implícitos sobre a confiabilidade de um sistema de *software* concorrente, foi elaborada por Roriz, Rodrigues e Laranjeira (2014) uma metodologia que fornece informações importantes para analisar as propriedades na composição dos componentes do sistema. As informações orientam as ações do refinamento arquitetural,

permitindo a construção de um *software* com maior confiabilidade. A análise quantitativa da confiabilidade foi realizada com base no conceito de Cadeias de Markov Discretas (DTMC), que propicia verificar quantitativamente o comportamento dinâmico do sistema, implementada na ferramenta PRISM (PARKER, 2013), utilizada para modelagem formal e análise de sistemas que exibam comportamento probabilístico.

O modelo comportamental foi gerado no LTSA-MSC. A etapa qualitativa agregou a semântica aos cenários implícitos negativos encontrados, e foram avaliados a natureza das falhas originadas pelos cenários implícitos considerados. Com base nos resultados obtidos nas aplicações, foi evidenciada a importância de levar em consideração a presença dos cenários implícitos e nortear um projeto de arquitetura que melhor atenda aos requisitos de confiabilidade que se espera de um sistema de *software* de natureza concorrente.

Para descrever as execuções internas do sistema, esta pesquisa usa a modelagem comportamental gerada a partir de rastros de execução, ao contrário do algoritmo proposto por Uchitel, Kramer e Magee (2001), que para analisar internamente um *software* utiliza modelos comportamentais criados a partir de MSCs de alto nível.

Para encontrar potenciais cenários implícitos, o trabalho de Souza e Mendonça (2007) apresentou um ambiente de engenharia reversa voltado para a detecção de incoerências em *softwares*, com base em de informações providas de sistemas, capturadas durante as execuções das aplicações. Diferente desta pesquisa, que para inferir sobre possíveis falhas, extrai os rastros de execução do modelo comportamental usando o algoritmo *All One-Loop Paths*, e compara rastros de execução do modelo e sistema inicial, e dessa forma os rastros que diferirem são identificados como cenários implícitos.

3.2.1 Refinamento

Para tentar coibir o problema que ocorre em decorrência da presença de cenários implícitos negativos em modelos comportamentais gerados a partir de *Traces*, Ramchandani, Russo e Alrajeh (2009) sugeriram uma solução que remove a transição do LTS causadora da ação indesejada. A proposta refina diretamente o modelo LTS de modo que o resultado final seja uma transformação do LTS inicial, que rejeita todos os cenários negativos. O processo passa por duas fases distintas, a primeira visa desenvolver o comportamento do sistema, a fim de preservar as propriedades desejáveis. A segunda etapa faz uso do modelo generalizado juntamente com as execuções indesejáveis do sistema, para incorporar todo o comportamento positivo e, ao

mesmo tempo, impedir qualquer conduta negativa. Para desenvolver a técnica utilizou-se um protótipo como parte da ferramenta *Labelled Transition System Analyser - Message Sequence Charts* (LTSA-MSC) existente, que mostra o algoritmo em operação funcionando precisamente, retornando a saída esperada.

Maia (2011) propôs uma técnica com suporte ferramental direcionado à geração de modelos comportamentais analiticamente probabilísticos, de modo que a verificação de qualquer propriedade contra esses modelos produz resultados precisos. Apresentou-se o conceito de refinamento para reestruturar a modelagem, em que transforma um modelo em um modelo observacionalmente equivalente, introduzindo cópias de estados específicos, incluindo a estrutura de ramificação (estados e transições) que se seguem.

O trabalho apresentado por Ramchandani, Russo e Alrajeh (2009) refina diretamente o LTS removendo a transição causadora da ação indesejada, de modo que resultado final seja uma transformação do LTS inicial. A abordagem exposta nesta pesquisa também traz a oportunidade de detectar cenários implícitos em LTS e refinar a modelagem, de maneira que remova as transições causadoras de erros. A principal diferença para com o trabalho primeiramente citado é que toda a fase de detecção e refinamento ocorre de forma interativa com um determinado usuário que entende sobre toda a estrutura e especificação do sistema.

3.3 CONSIDERAÇÕES FINAIS

Neste Capítulo foram apresentados trabalhos que se relacionam a algum dos aspectos mais importantes da presente pesquisa. Primeiramente, foram abordados trabalhos sobre a extração de modelo e suas características. A seção foi divida em três partes, onde inicialmente exploraram-se as pesquisas voltada para a extração estática, a segunda apresenta trabalhos que envolve a extração dinâmica e sua características, e ao final expõe algumas técnicas que tratam sobre a extração híbrida.

Denotou-se que no contexto da detecção e uso de cenários implícitos exitem várias técnicas que utilizam diferentes métodos voltados para a detecção, como abordagem de especificação do serviço baseado em colaborações de UML, engenharia reversa e a partir do uso das *Message Sequence Charts*.

Por fim, foram apresentados estudos sobre o refinamentos em modelo comportamentais. Duas abordagens foram expostas, sendo que a primeira refina diretamente o modelo LTS de modo que o resultado final seja uma transformação do LTS inicial, removendo diretamente

a transição causadora do cenário errôneo. A segunda refinou modelos comportamentais analiticamente probabilísticos, de modo que a verificação das propriedade desses modelos produz resultados precisos.

4 ABORDAGEM PROPOSTA

Este capítulo apresenta os detalhes da abordagem que este trabalho propõe para detectar e remover cenários implícitos de modelos comportamentais. Inicialmente é mostrada, na Seção 4.1, a descrição da abordagem, na qual são detalhadas as fases de detecção de cenários implícitos, análise e os detalhes do processo de refinamento através de um exemplo motivador. Em seguida, na Seção 4.2, apresenta-se o suporte ferramental, expondo como a abordagem foi implementada e disponibilizada em uma ferramenta de modelagem.

4.1 DESCRIÇÃO DA ABORDAGEM

Este trabalho propõe uma abordagem semi-automática direcionada à identificação de cenários implícitos em um modelo de comportamento, representado como um LTS, e seu iterativo refinamento de acordo com a análise do usuário especialista, até que um modelo que represente apenas comportamentos do sistema válidos seja alcançado. Vale ressaltar que usa-se o termo "usuário" para representar o usuário da abordagem, que espera-se ser alguém com bom conhecimento do código ou do funcionamento da aplicação, e não usuário do sistema sob análise.

Todo processo de refinamento é feito com base nas perspectivas do usuário. Por esse motivo, ele deve ser apto a raciocinar sobre os cenários implícitos detectados, no intuito de inferir se os desvios encontrados são cenários positivos, ou seja, não representam erros, ou cenários negativos, refletindo que o modelo não é adequado para representar o comportamento real do sistema. Neste caso, ele deve ser capaz de verificar se o problema encontrado refere-se a como o sistema foi implementado, isto é, a arquitetura do *software* não está consolidada como projetada na fase inicial de desenvolvimento, ou se o motivo da presença de cenários negativos também pode localizar-se no algoritmo de geração de modelo.

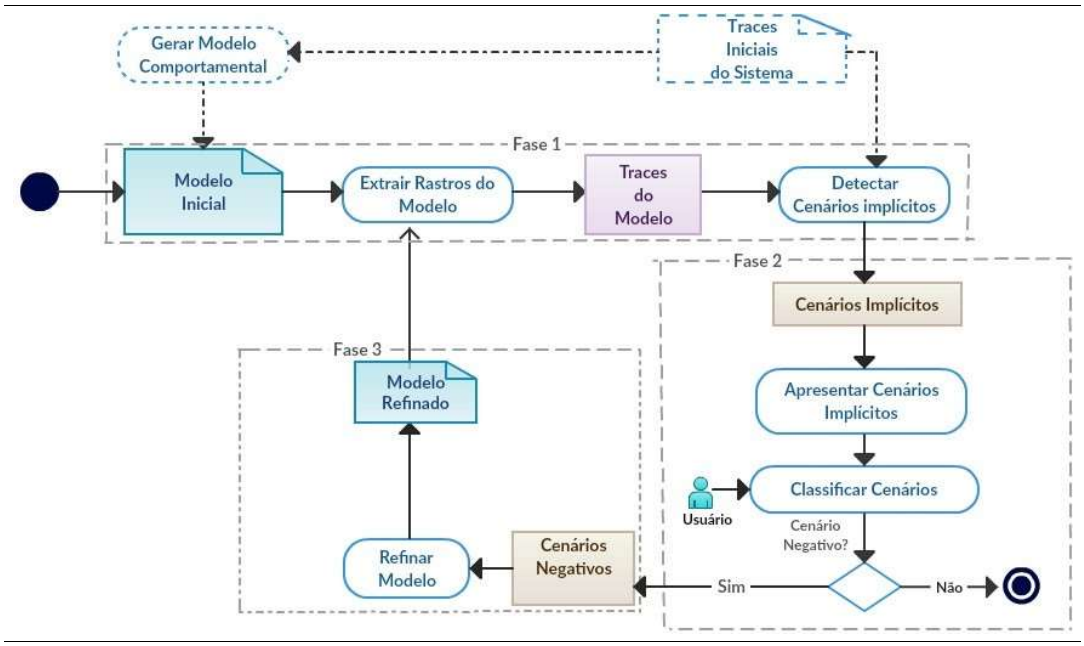
A abordagem consiste em três fases principais, conforme ilustrado na Figura 3. A primeira, *detecção de cenários implícitos*, recebe como entrada o modelo comportamental inicial do sistema, representado como um LTS, e através de um algoritmo utilizado na área de testes baseados em modelos, gera um conjunto de *traces* a partir do modelo inicial. Esses *traces* são comparados com os *traces* usados para gerar o modelo inicial do sistema. Os *traces* que foram gerados pelo algoritmo, mas que não estão nos *traces* de entrada, são considerados cenários implícitos.

A segunda fase ocorre a *análise* dos cenários pelo usuário especialista, que os classifica em positivos e negativos. Para tomar a decisão sobre os cenários, o usuário pode

utilizar de diferentes artifícios, como verificar alguma modelagem alternativa de comportamento (como um diagrama de sequência), executar o sistema utilizando cada *trace* em análise como caso de teste, ou mesmo usar sua expertise do sistema para avaliar se os comportamentos representados pelos *traces* podem ou não ser encontrados no sistema. Na abordagem proposta neste trabalho é fundamental que o usuário designado a interagir com a aplicação seja o responsável por entender a toda a estrutura e requisitos do sistema, pois a abordagem é interativa e depende das escolhas do usuário para que ao final do processo seja obtidos os resultados desejados.

Por fim, acontece o *refinamento* do modelo, no qual os caminhos que geram os cenários implícitos serão removidos e um novo modelo é gerado. O modelo refinado resultante serve como entrada para a primeira fase para que a abordagem possa novamente acontecer. O processo é repetido até que nenhum cenário implícito seja detectado, seja porque o usuário removeu todos os negativos ou marcou os existentes como positivos. Como a abordagem não introduz novos cenários após a remoção, após algumas iterações, a lista de cenários implícitos será vazia.

Figura 3 – Fases da abordagem

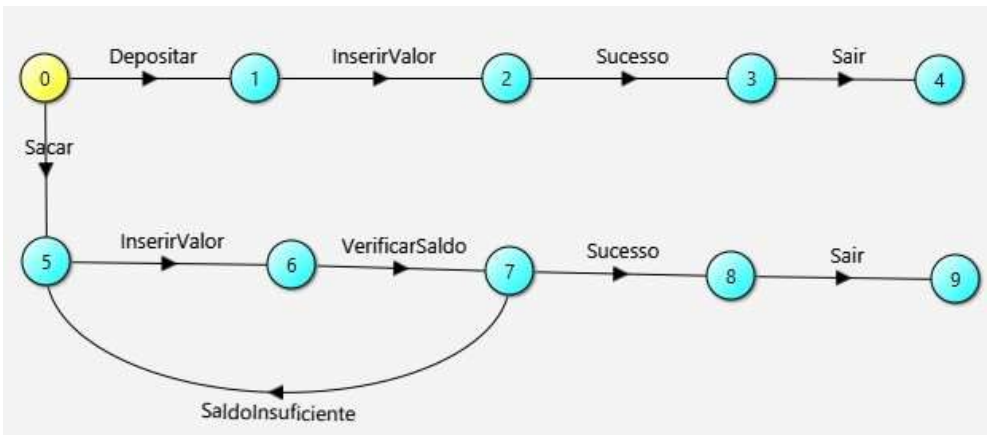


Fonte – Elaborado pelo autor

Para melhor explanar as fases e objetivos da abordagem proposta, considere um pequeno sistema bancário que fornece apenas duas funcionalidades para o correntista: depositar e sacar. No primeiro caso, o correntista insere o valor a ser depositado (abstrai-se aqui a forma de depósito), que é creditado com sucesso na conta, encerrando assim a solicitação. No segundo

caso, após o correntista indicar o valor a ser sacado, é feita uma verificação se ele possui saldo suficiente para retirar esse valor. Caso ele tenha, o valor é sacado e o processo finaliza, caso contrário uma mensagem de erro é informada, permitindo que o correntista entre com um novo valor. A Figura 4 apresenta o comportamento do sistema. A seguir é descrito como cada fase da abordagem é acontece para esse exemplo.

Figura 4 – Modelo de comportamento original do Sistema bancário



Fonte – Elaborado pelo autor

4.1.1 Detecção de cenários implícitos

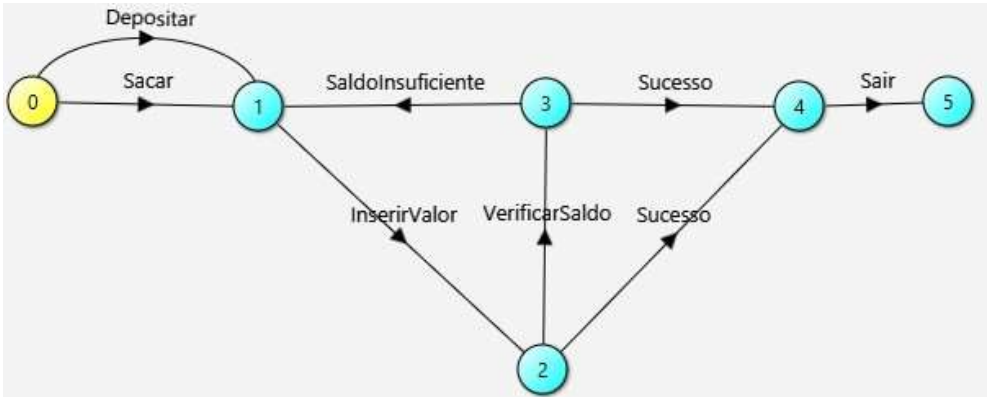
A primeira fase começa com o usuário informando um modelo de comportamento do sistema gerado através de alguma técnica de extração dinâmica. Considere que o sistema foi, de alguma forma, executado algumas vezes, gerando um conjunto de *traces* mostrados na Tabela 1, que foram usados como entrada para uma ferramenta de geração de modelos. O resultado é o modelo representado na Figura 5. Esse modelo é usado como entrada da fase 1 da abordagem.

Quadro 1 – Traces gerados a partir da implementação inicial do sistema

Depositar, InserirValor, Sucesso, Sair
Sacar, InserirValor, VerificarSaldo, SaldoInsuficiente, InserirValor, VerificarSaldo, Sucesso, Sair
Sacar, InserirValor, VerificarSaldo, Sucesso, Sair

Fonte – Elaborado pelo autor

Figura 5 – Modelo comportamental do sistema bancário gerado a partir de uma técnica de extração dinâmica



Fonte – Elaborado pelo autor

Em seguida, a abordagem utiliza o algoritmo *All One-loop Paths* (ALP) (UTTING; LEGEARD, 2010) no intuito de extrair os rastros de execução do modelo comportamental inicial. Esse algoritmo é frequentemente usado como técnica para geração de casos de testes em testes baseados em modelos, percorrendo todos os possíveis caminhos do modelo passando por um mesmo *loop* apenas uma vez. Dessa forma, todos os possíveis comportamentos que não possuem ciclos, juntamente com os comportamentos que contém apenas uma execução dos *loops*, serão retornados. Vale ressaltar que espera-se que todos os *traces* de entrada estejam contidos no modelo inicial. Portanto, os únicos comportamentos que podem estar nos *traces* iniciais e não estar nos *traces* do modelo são aqueles que possuem mais de uma repetição de *loops*. Contudo, considera-se que esses comportamentos podem ser representados pelos *traces* que contém uma repetição dos *loops*. Assim, não há prejuízo em desconsiderá-los. A Tabela 1 apresenta os rastros extraídos a partir do algoritmo ALP.

Tabela 1 – Traces do modelo comportamental extraídos com o ALP

Traces gerados pelo One Loop Path
Sacar, InserirValor, VerificarSaldo, SaldoInsuficiente, InserirValor, Sucesso, Sair
Sacar, InserirValor, Sucesso, Sair
Depositar, InserirValor, Sucesso, Sair
Sacar, InserirValor, VerificarSaldo, Sucesso, Sair
Sacar, InserirValor, VerificarSaldo, SaldoInsuficiente, InserirValor, VerificarSaldo, Sucesso, Sair
Depositar, InserirValor, VerificarSaldo, Sucesso, Sair
Depositar, InserirValor, VerificarSaldo, SaldoInsuficiente, InserirValor, VerificarSaldo, Sucesso, Sair
~~Depositar, InserirValor, VerificarSaldo, SaldoInsuficiente, InserirValor, Sucesso, Sair~~

Fonte – Elaborado pelo autor

Logo após a atividade da extração dos rastros da modelagem comportamental, acontece a comparação entre os *traces* gerados pelo algoritmo ALP e os rastros de entrada fornecidos pelos usuário. Comparando-se os traces das Tabelas 1 e 1, percebe-se que os seguintes traces estão na segunda (ou seja, foram gerados pelo ALP), mas não estão na primeira, consistindo assim nos cenários implícitos encontrados. Esses cenários são apresentados na Tabela 2.

Tabela 2 – Cenários Implícitos - Modelo bancário

Cenários Implícitos detectados
Depositar, InserirValor, VerificarSaldo, SaldoInsuficiente, InserirValor, Sucesso, Sair
Depositar, InserirValor, VerificarSaldo, SaldoInsuficiente, InserirValor, VerificarSaldo, Sucesso, Sair
Depositar, InserirValor, VerificarSaldo, Sucesso, Sair
Sacar, InserirValor, Sucesso, sair
Sacar, InserirValor, VerificarSaldo, SaldoInsuficiente, InserirValor, Sucesso, sair

Fonte – Elaborado pelo autor

4.1.2 Análise de Cenários Implícitos

A segunda fase começa com a apresentação dos cenários implícitos detectados na fase anterior para o usuário especialista no sistema. Ao analisar a Tabela 2, o usuário verifica quais cenários são positivos e quais são negativos.

Pegando como exemplo o cenário **{Depositar, InserirValor, VerificarSaldo, SaldoInsuficiente, InserirValor, Sucesso, Sair}**, o usuário detecta que esse comportamento é inválido porque não existe a possibilidade da ação **VerificarSaldo** acontecer após um depósito. Já o rastro **{Sacar, InserirValor, Sucesso, sair}** não ocorre no sistema inicial pois a ação **Sucesso** não pode vir imediatamente após a ação **InserirValor** em uma solicitação de saque, pois indica que o saldo do correntista não foi verificado antes de realizar a retirada. Ambos cenários são considerados negativos, pois eles não são comportamentos aceitáveis do sistema e devem, portanto, ser removidos do modelo. Neste exemplo, é indiferente se os cenários implícitos são gerados por problemas na implementação do sistema ou no algoritmo de geração do modelo.

4.1.3 Refinamento

Refinar é a complexa tarefa de implementar um novo modelo de comportamento que satisfaça todas as especificações inicialmente estabelecidas no sistema inicial. O objetivo é retirar as transições que estão causando cenários negativos, baseada na expertise do usuário,

para que ao final do processo, o modelo comportamental possa ser realista ao *software* a que representa. No caso de cenários positivos, a estratégia consiste em marcá-los como cenários válidos e não excluí-los do modelo de forma que, em uma próxima iteração de detecção de cenários implícitos, eles não sejam mais apresentados ao usuário.

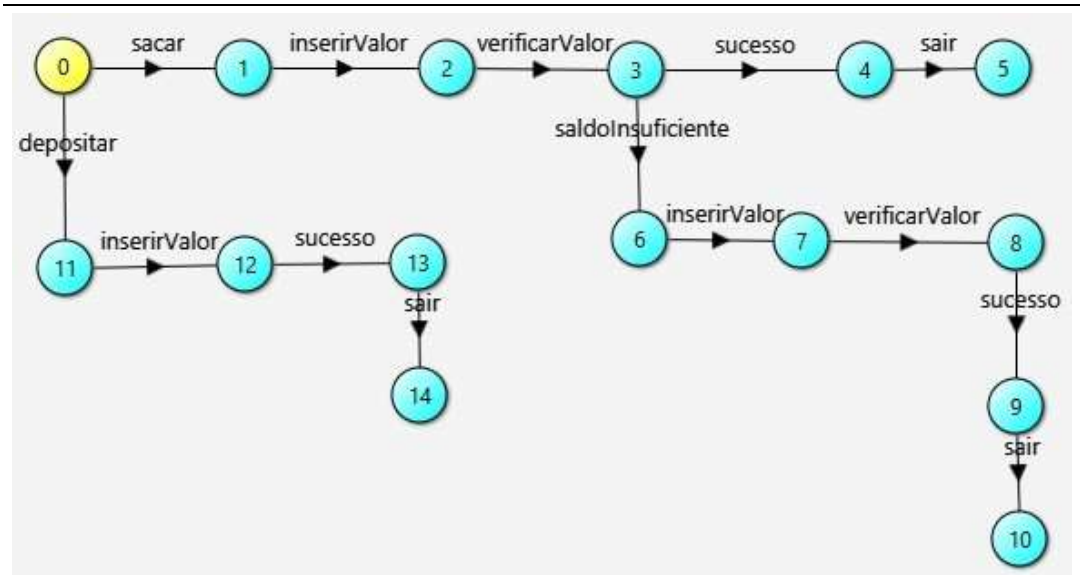
A técnica do refinamento é realizada nas seguintes etapas, como mostra a Figura 6.

Figura 6 – Etapas do refinamento

Fonte – Elaborado pelo autor

Quando uma remoção é solicitada, o modelo é "aberto", ou seja, todos os ciclos são transformados em caminhos não cíclicos, mas de modo que conserve todos os comportamentos obtidos pelo uso do ALP, ou seja, não há perda de comportamento. A Figura 7 mostra o modelo do sistema bancário aberto. Com esse modelo aberto, é possível excluir os caminhos que geram os cenários implícitos de forma que a exclusão não interfira em outros caminhos.

Figura 7 – Cenário negativo selecionados - Modelo bancário

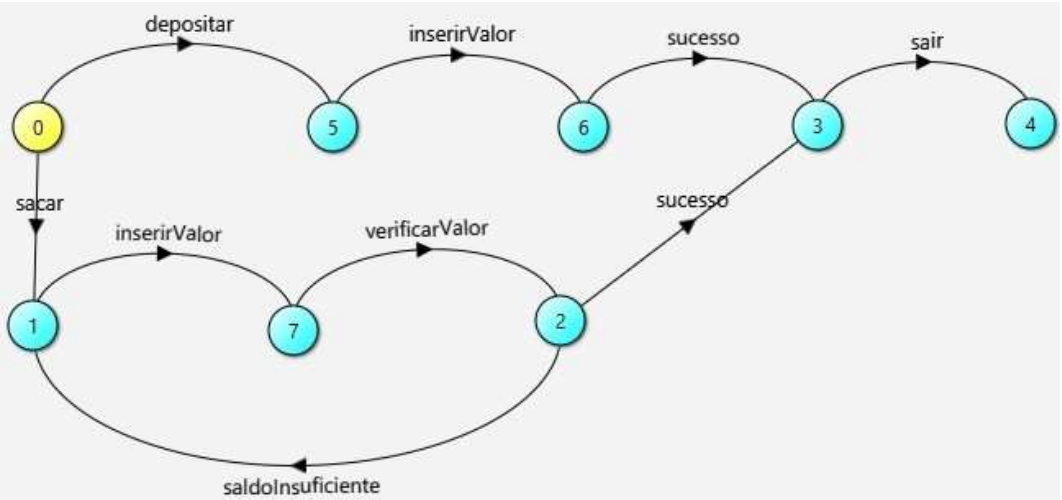


Fonte – Elaborado pelo autor

O último passo desta fase consiste em unir as transições do modelo aberto utilizando um algoritmo de minimização de modelos, gerando assim um novo modelo refinado. Vale ressaltar que esse algoritmo não deve gerar novos cenários negativos no modelo. A Figura 8

mostra o modelo refinado do sistema bancário após a remoção dos cenários negativos detectados na fase anterior.

Figura 8 – Modelo refinado bancários



Fonte – Elaborado pelo autor

Como o modelo refinado pode ainda conter outros cenários implícitos, ele serve de entrada para a primeira fase da abordagem para que o usuário possa começar o processo novamente. No caso do exemplo em questão, após o início da fase 1 com o novo modelo, é realizada novamente a extração de *traces* do modelo para realizar a detecção de cenários implícitos. A Tabela 3 apresenta os rastros extraídos da nova modelagem.

Tabela 3 – Traces do modelo refinado do sistema bancário

Traces gerados pelo One Loop Path - Modelo bancário refinado
Depositar, InserirValor, Sucesso, Sair
Sacar, InserirValor, VerificarSaldo, SaldoInsuficiente, InserirValor, VerificarSaldo, Sucesso, Sair
Sacar, InserirValor, VerificarSaldo, Sucesso, Sair

Fonte – Elaborado pelo autor

Comparando os rastros da Tabela 3 com os rastros iniciais do sistema (Tabela 1), constata-se que não existe mais cenários diferentes, isto é, não há mais cenários implícitos. Isso significa que o modelo da Figura 8 representa de maneira correta o comportamento do sistema de acordo com a opinião do usuário. De fato, comparando-se o modelo inicial do sistema apresentado na Figura 4 com o modelo final da Figura 8, nota-se que ambos são equivalentes. Com isso, o processo é finalizado.

4.2 SUPORTE FERRAMENTAL

A abordagem foi implementada dentro da ferramenta LoTuS, uma ferramenta de código aberto, extensível, que permite ao usuário modelar o comportamento do sistema utilizando LTSs e realizar análises do comportamento modelado. LoTuS permite tanto a modelagem gráfica, que possibilita a construção de modelos comportamentais através de um mecanismo de *drag and drop*, quanto textual, que baseia-se numa extensão probabilística da linguagem FSP (MAGEE; KRAMER, 2006). LoTus ainda tem como característica um amplo conjunto de técnicas de análise de modelos, como simulação, execução, detecção de *deadlock* e verificação probabilística de propriedades de alcançabilidade (*reachability*). A aplicação foi implementada em Java, utilizando a tecnologia para construção de interfaces JavaFX, e está disponível publicamente no GitHub[1].

Uma das principais vantagens do uso do LoTuS é que ele fornece uma API que permite desenvolvedores adicionarem novas funcionalidades através de *plugins*. Dentre os plugin já desenvolvidos e disponibilizados pela ferramenta, pode-se citar:

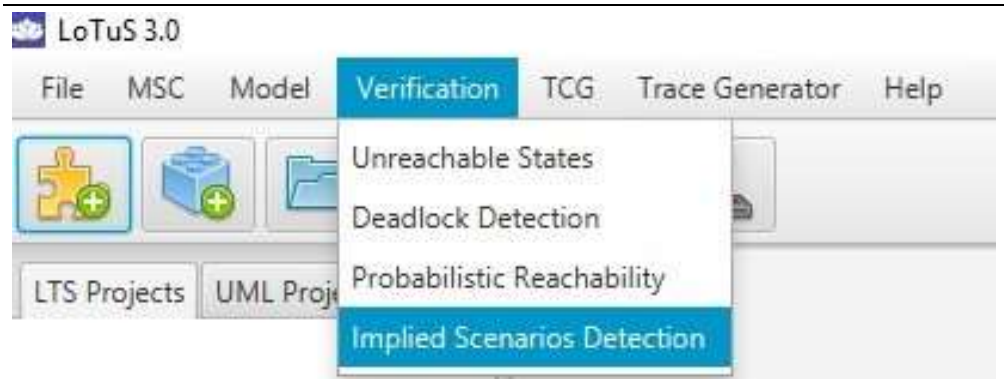
- *Trace generator*: gera um conjunto de *traces* para um dado modelo de acordo com a quantidade solicitada pelo usuário;
- *Model from Trace*: dado um conjunto de *traces* em formato *csv*, um novo modelo é gerado através de um algoritmo que, para cada *trace* novo no conjunto de entrada, insere um novo caminho no modelo, de forma que ações equivalentes são concentradas na mesma transição. A heurística para a geração do modelo é bem simples e consiste em tentar criar um novo caminho no modelo para cada rastro do arquivo. Caso o rastro contenha caminhos que já estejam no modelo, então os caminhos são sobrescritos, até que uma nova ação apareça. Caso haja ações repetidas no caminho, a segunda ação é retornada para o estado final da primeira, constituindo um ciclo.
- *Probability annotator*: dado um conjunto de *traces* e um modelo que contenha aqueles *traces*, um algoritmo anota as probabilidades de ocorrência de cada transição a partir da varredura do conjunto de *traces*.
- *Test Case Generator* (TCG) (MUNIZ; NETTO; MAIA, 2015): é uma ferramenta para geração e seleção de casos de testes baseados em modelos. Ele disponibiliza um conjunto de técnicas de geração e seleção tanto para modelos trandicionais como probabilísticos. Dentre as técnicas de seleção, destaca-se o All One-loop Path, usado também neste

[1] https://github.com/lotus-tool/lotus-tool

trabalho.

A abordagem proposta neste trabalho é disponibilizada na ferramenta LoTuS através do *plugin Implied Scenarios Detection* (Figura 9). Os *traces* de entrada devem ser agrupados em um arquivo *csv*, onde cada linha do arquivo representa um *trace* do sistema. O modelo de entrada pode tanto ser modelado diretamente no LoTuS quanto gerado utilizando-se o *plugin Model from Trace*, descrito anteriormente, e que pode beneficiar o usuário caso ele não tenha outra ferramenta ou técnica disponível para geração de modelos a partir dos *traces*. Vale salientar que a abordagem proposta é independente do algoritmo de extração dinâmica de modelos. Portanto, caso o usuário queira, ele pode implementar seu próprio algoritmo como um *plugin* do LoTuS.

Figura 9 – *Plugin Implied Scenarios Detection*



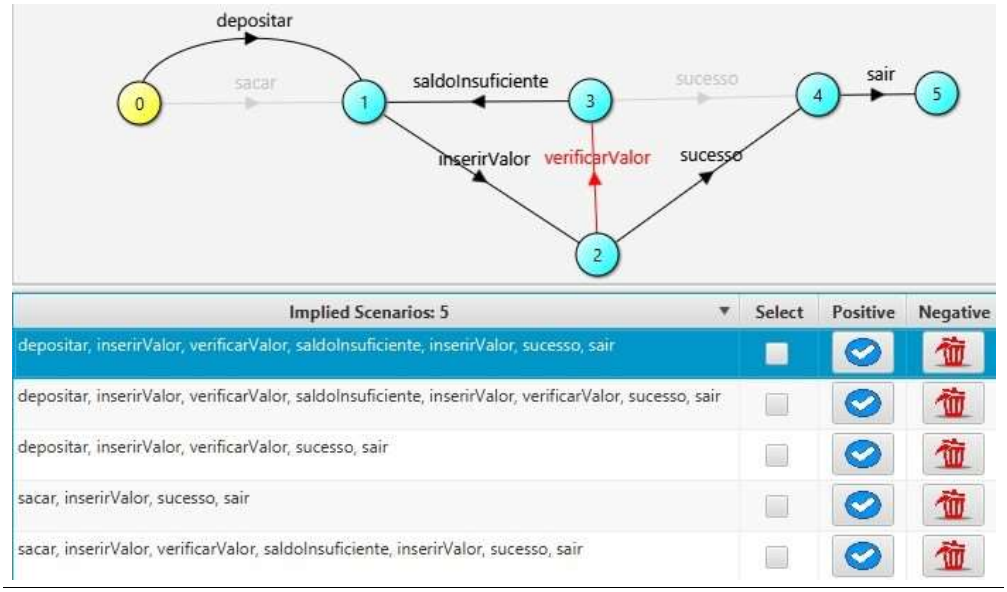
Fonte – Elaborado pelo autor

A Figura 10 mostra a interface do LoTuS com o *plugin Implied Scenarios Detection* em execução. Na parte superior da figura há o modelo comportamental do sistema. Na parte inferior observa-se o cenários implícitos detectados, sendo o número total apresentado na primeira linha (no caso do exemplo mostrado na figura, há cinco cenários implícitos). Quando o usuário clica em uma linha da tabela, o comportamento selecionado é realçado no modelo. A transição em vermelho informa a transição a partir da qual o cenário implícito acontece.

Para cada cenário implícito na tabela, há um botão *Positive* e outro *Negative*, que permitem ao usuário informar se aquele comportamento é positivo ou negativo, respectivamente. Ao clicar num desses botões, a ação de remoção do cenário implícito é iniciada. Caso o usuário queira remover vários cenários de uma vez, ele pode selecionar os cenários clicando em *Select*. Ao ser marcado como positivo, o cenário não é removido do modelo (como acontece com o negativo) pela ferramenta, mas sim insere o *trace* em uma lista de cenários válidos que são consultados antes de se apresentarem os cenários implícitos, de forma que só são mostrados os cenários que não estão nessa lista. O modelo final refinado contém todos os cenários inferidos

como positivos. O comportamento identificado como negativo é excluído do modelo. O processo é feito de maneira interativa com o usuário, e acontece quantas vezes for necessário.

Figura 10 – *Interface do LoTuS*



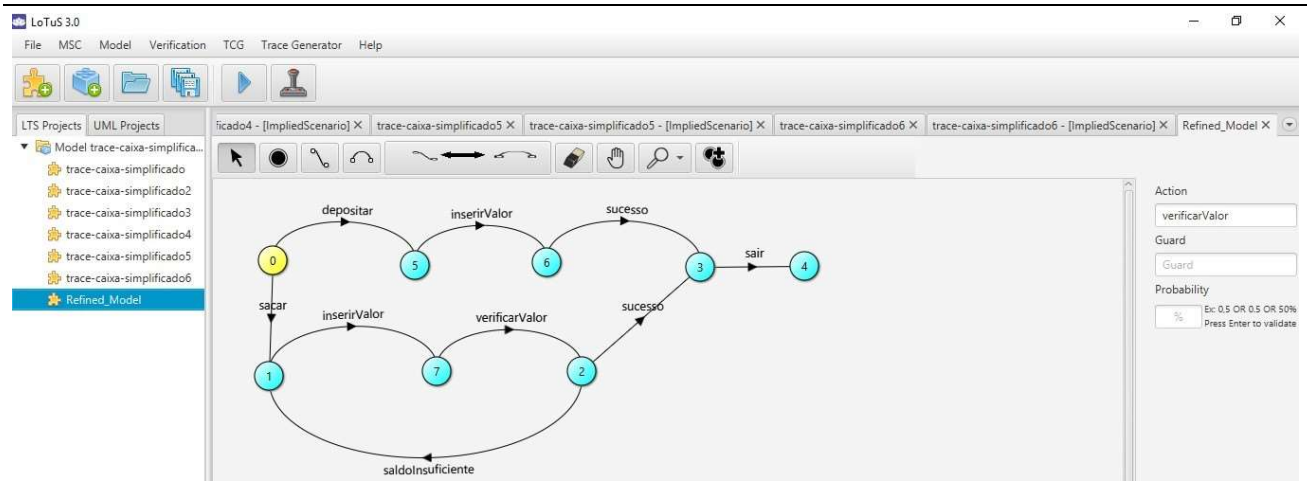
Fonte – Elaborado pelo autor

O algoritmo implementado para minimizar o modelo após a abertura e remoção checa se o modelo refinado criou novos cenários implícitos ou gera novamente os cenários removidos. A implementação é realizada em duas fases distintas, onde a primeira identifica os *loops* nas transições e a segunda une as transições de maneira que os cenários negativos não sejam reintroduzidos. Para impedir a criação de cenários errados, dentro do código da minimização, faz-se a detecção de cenários implícitos e, assim, as transições que gerem cenários negativos são automaticamente impedidas de serem criadas. O algoritmo implementado é apresentado no Apêndice A desta dissertação.

A Figura 11 apresenta a interface da ferramenta LoTuS após a remoção de todos os cenários negativos. Note, do lado esquerdo da figura, a presença de vários arquivos de modelos (representados com o nome inicial de “trace-caixa-simplificado”. Isso acontece pois, para cada processo de refinamento, uma novo modelo é gerado. No exemplo mostrado na figura, foram realizados seis refinamentos (ou seja, foram seis interações com o usuário solicitando refinamento) até se chegar ao modelo refinado final.

A última etapa do processo é a minimização do modelo aberto, em que as transições são unidas de forma que não crie cenários negativos, e gere um modelo que contenha todas as especificações iniciais, que foi projetado nas fases inciais de desenvolvimento para o *software*. O modelo resultante após a minimização deve conter todas as transições que correspondem aos cenários positivos.

Figura 11 – Modelo Final



Fonte – Elaborado pelo autor

4.3 CONCLUSÕES DO CAPÍTULO

Neste Capítulo foi apresentada a descrição da abordagem, onde explanou-se sobre toda metodologia usada para construir a técnica apresentada neste trabalho. Apresentou-se cada etapa que sucede durante a pesquisa, de maneira minuciosa explorando suas característica. A abordagem tem início a partir da modelagem comportamental, porém, a ferramenta LoTuS disponibiliza através de *plugins* internos, a possibilidade de construir o modelo inicial, e extrair seus rastros de execução.

Inicialmente, é detalhada a técnica usada para inferir possíveis falhas em modelagens comportamentais, denominada de detecção de cenários implícitos. A detecção é feita a partir da comparação dos rastros de execução de sistema e modelo comportamental, no qual o resultado final são os *traces* que não estão condizendo com as expectativas iniciais de desenvolvimento do *software*. Logo após, é elucidada a respeito da análise de cenários implícitos realizada pelo usuário que interage com a técnica.

Em seguida, é esclarecido a respeito da estrutura do refinamento, no qual é voltado para remoção de cenários negativos identificados na modelagem. Refinar significa restaurar o modelo comportamental de forma que ao final gere um novo modelo, que condiz com o *software* a que representa. Toda a etapa de refinamento é realizada com base na interação do usuário.

Por fim, é apresentada uma implementação da abordagem proposta na ferramenta de modelagem de comportamento de sistemas denominada LoTuS como sendo um *plugin* para a mesma. Foram mostradas telas de como as fases da abordagem foram desenvolvidas no *plugin*.

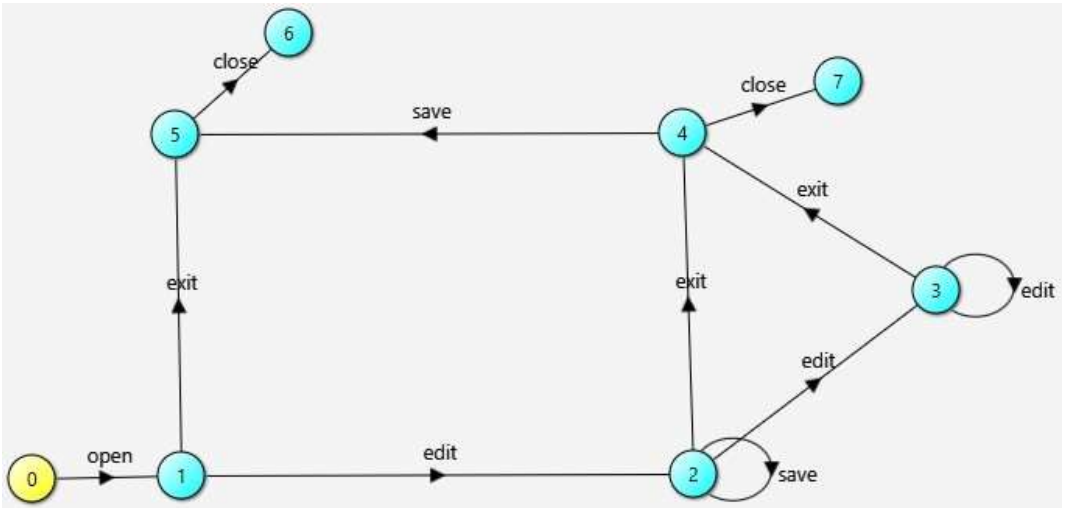
5 EXEMPLOS DE USO

Neste capítulo, três exemplos de uso são apresentados com o objetivo de demonstrar a aplicabilidade da abordagem. O primeiro caso trata de um sistema para a edição de texto, enquanto o segundo baseia-se em um aplicativo médico implementado como um serviço *web*. O terceiro caso aborda um sistema de controle de vendas *web*, no qual um desenvolvedor conhecedor do sistema participou das inferências sobre a modelagem.

5.1 SISTEMA EDITOR DE TEXTOS

O modelo comportamental deste exemplo representa um sistema voltado para a edição de textos (Figura 12). A aplicação permite ao usuário funções como abrir um determinado arquivo (*open*), editar o texto de forma desejada (*edit*), salvar ao finalizar a edição (*save*), sair do arquivo (*close*) e fechar o programa (*close*).

Figura 12 – Modelo comportamental editor

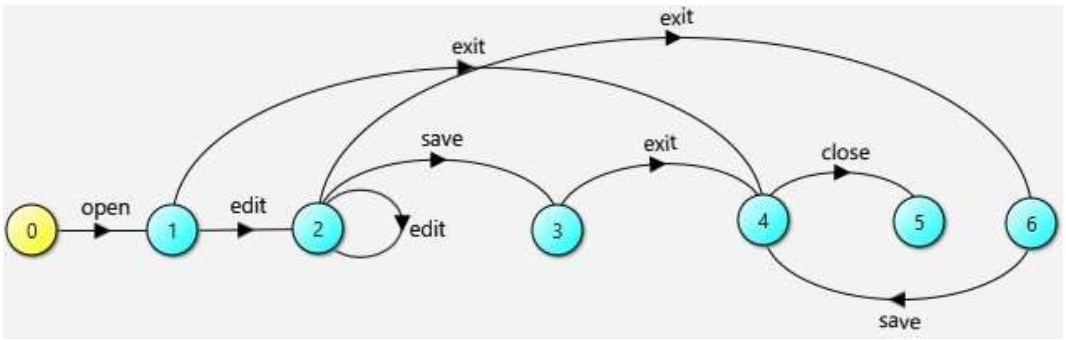


Fonte – Elaborado pelo autor

A Figura 12 mostra o comportamento esperado do sistema, que é uma versão adaptada da aplicação usada em (DUARTE, 2013). Como não se tinha o código-fonte da aplicação, utilizou-se o modelo da Figura 13 e o *plugin Trace Generator* do LoTuS para gerar os traces desse modelo, representando execuções do sistema. Com isso, foram extraídos de forma dinâmica 1000 (mil) rastros de execução do comportamento do sistema e, com esses *traces*, construído o modelo de comportamental inicial utilizando-se o plugin*Model from Trace* do LoTuS. Esse modelo inicial é mostrado na Figura 12 e serviu de entrada para a fase 1 da

abordagem. O usuário aqui é a própria autora deste trabalho.

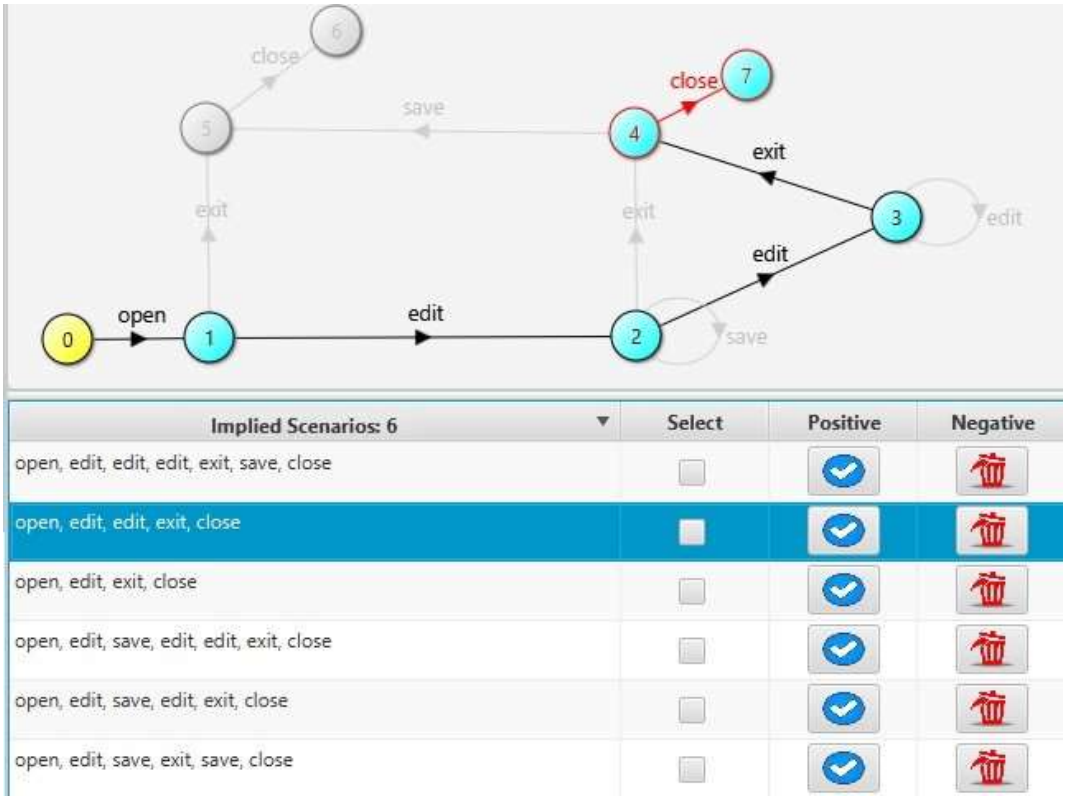
Figura 13 – Modelo inicial editor



Fonte – Elaborado pelo autor

No intuito de verificar a confiabilidade do modelo comportamental, realiza-se primeiramente a detecção de cenários. Os resultados dessa verificação indicaram a presença de seis cenários implícitos, conforme mostra a Figura 14.

Figura 14 – Cenários implícitos modelo editor



Fonte – Elaborado pelo autor

Na etapa de análise, o usuário inferiu que todos os cenários detectados pela técnica são considerados negativos e, portanto, representam erros que precisam ser removidos. As

inconsistências ocorrem pelo fato de que não pode fechar (*close*) a aplicação sem salvar (*save*) o arquivo, como o que acontece em *(..., edit, exit, close)* e *(..., save, edit, edit, exit, close)*. Outro motivo que causou o erro na modelagem é o fato de não se poder sair (*exit*) e logo após salvar (*save*) um arquivo, conforme identifica-se em *(..., save, exit, save, ...)*.

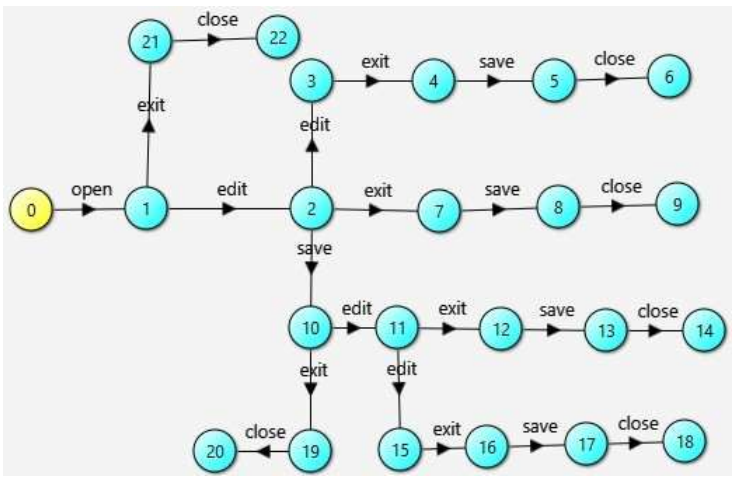
Em seguida, o usuário optou por remover todos os cenários de uma única vez a partir da função *Select* (Figura 15) e botão *Negative*, o que gerou o modelo aberto apresentado na Figura 16 e, posteriormente à remoção dos cenários e minimização, resultou no modelo refinado mostrado na Figura 17. O usuário voltou para a fase inicial da abordagem utilizando esse modelo como entrada. Ao se realizar novamente a detecção de cenários implícitos, nenhum cenário foi retornado, indicando que esse modelo representa corretamente o comportamento do sistema.

Figura 15 – Função Select - Modelo editor

Implied Scenarios: 6	Select	Positive	Negative
open, edit, edit, edit, exit, close	☑		
open, edit, edit, exit, close	☑		
open, edit, exit, close	☑		
open, edit, save, edit, edit, exit, close	☑		
open, edit, save, edit, exit, close	☑		
open, edit, save, exit, save, close	☑		

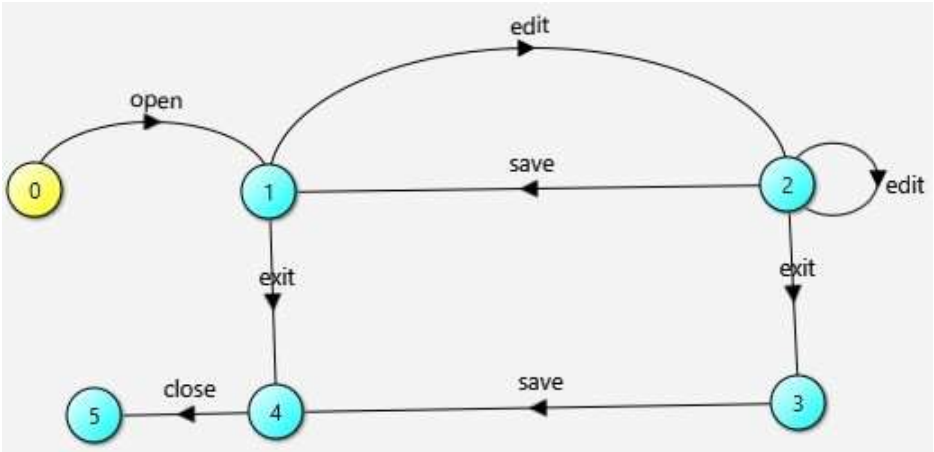
Fonte – Elaborado pelo autor

Figura 16 – Modelo final aberto - Modelo editor



Fonte – Elaborado pelo autor

Figura 17 – Modelo editor refinado



Fonte – Elaborado pelo autor

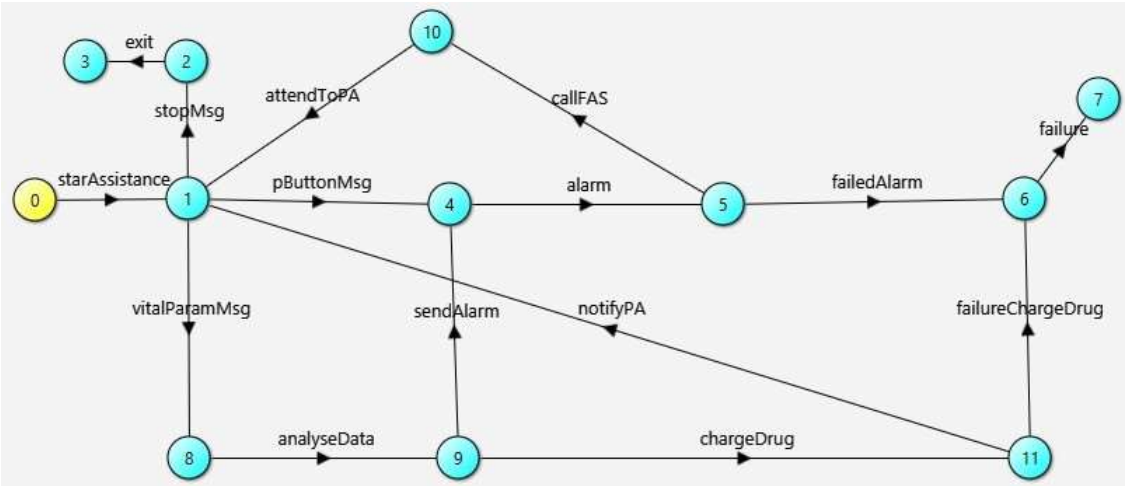
5.2 TELEASSISTANCE

Este exemplo é uma adaptação de um aplicativo chamado TeleAssistance (TA), um sistema distribuído para assistência médica remota, originalmente descrito em (EPIFANI *et al.*, 2009). Uma vez que o processo é iniciado pelo paciente, ele tem três escolhas: enviar os parâmetros vitais do paciente (*vitalParamMsg*), mandar um alarme de pânico pressionando o botão (*pButtonMsg*), ou interromper o aplicativo (*stopMsg*). No caso de envio de parâmetros vitais, os dados são analisados por um laboratório, que pode recomendar ao usuário que mude de remédio ou disparar um alarme para um time de primeiros socorros (*First-Aid Squad* - FAS), que vai até a casa do paciente atendê-lo. É possível que tanto esse alarme quanto a notificação de mudança de remédio falhem. Quando o usuário aperta o botão de pânico, o FAS também é acionado.

Diferentemente do exemplo anterior, o código-fonte estava disponível e foi utilizado para a geração dos *traces* do sistema. Foram gerados 500 *traces* e, a partir deles, criou-se o modelo comportamental da Figura 18 através do *plugin Model From Trace*. Neste exemplo, o usuário da abordagem também é a autora deste trabalho.

A fim de verificar se o modelo da Figura 18 é valido para representar o sistema TA, realizou-se a detecção de cenários implícitos, quando foram encontrados os nove cenários apresentados na Tabela 4. A Figura 19 apresenta a interface do LoTuS com o *plugin* de detecção em execução. Nela, tem-se o modelo com a transição **pButtonMsg** em vermelho mostrando o ponto onde é detectado o cenário implícito selecionado na tabela na parte de baixo da figura.

Figura 18 – Modelo comportamental inicial do TeleAssistance



Fonte – Elaborado pelo autor

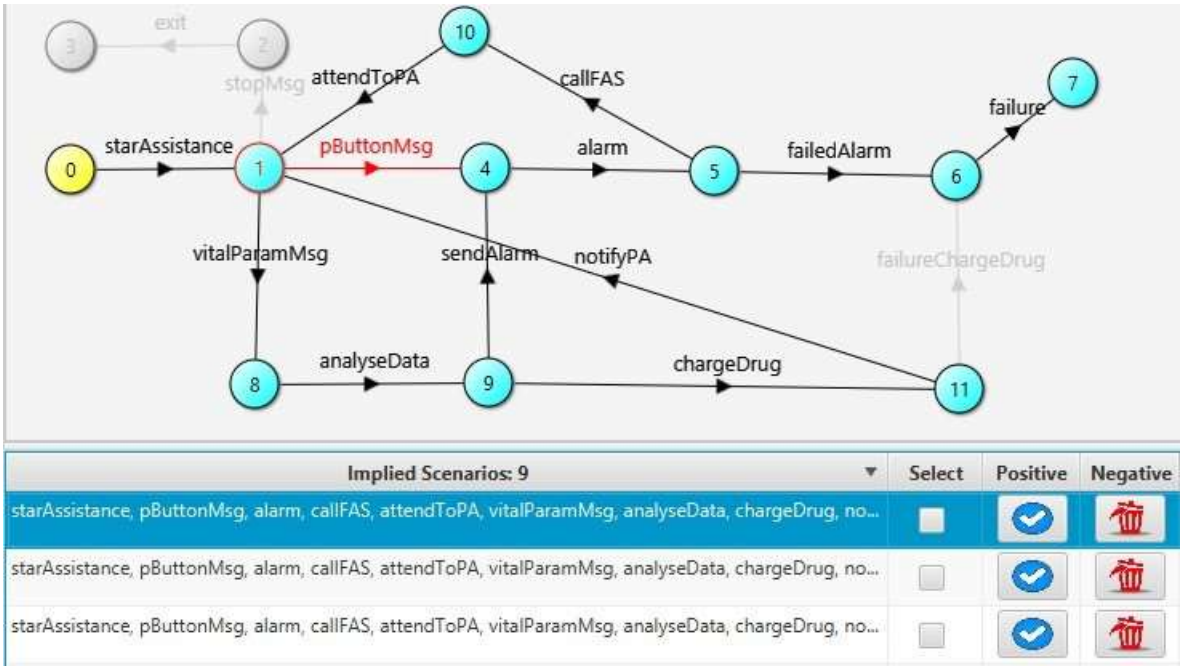
Tabela 4 – Tabela de Cenários Implícitos - Modelo Tele Assistance

Cenários Implícitos detectados
starAssistance, pButtonMsg, alarm, callFAS, attendToPA, vitalParamMsg, analyseData, sendAlarm, alarm, callFAS, attendToPA, vitalParamMsg, analyseData, chargeDrug, notifyPA, pButtonMsg, alarm, failedAlarm, failure
starAssistance, pButtonMsg, alarm, callFAS, attendToPA, vitalParamMsg, analyseData, sendAlarm, alarm, callFAS, attendToPA, vitalParamMsg, analyseData, chargeDrug, notifyPA, vitalParamMsg, analyseData, chargeDrug, failureChargeDrug, failure
starAssistance, pButtonMsg, alarm, callFAS, attendToPA, vitalParamMsg, analyseData, chargeDrug, notifyPA, vitalParamMsg, analyseData, sendAlarm, alarm, callFAS, attendToPA, pButtonMsg, alarm, failedAlarm, failure
starAssistance, pButtonMsg, alarm, callFAS, attendToPA, vitalParamMsg, analyseData, chargeDrug, notifyPA, vitalParamMsg, analyseData, sendAlarm, alarm, callFAS, attendToPA, vitalParamMsg, analyseData, sendAlarm, alarm, failedAlarm, failure
starAssistance, pButtonMsg, alarm, callFAS, attendToPA, vitalParamMsg, analyseData, chargeDrug, notifyPA, vitalParamMsg, analyseData, sendAlarm, alarm, callFAS, attendToPA, vitalParamMsg, analyseData, chargeDrug, failureChargeDrug, failure
starAssistance, vitalParamMsg, analyseData, sendAlarm, alarm, callFAS, attendToPA, pButtonMsg, alarm, callFAS, attendToPA, vitalParamMsg, analyseData, chargeDrug, notifyPA, vitalParamMsg, analyseData, sendAlarm, alarm, failedAlarm, failure
starAssistance, vitalParamMsg, analyseData, chargeDrug, notifyPA, pButtonMsg, alarm, callFAS, attendToPA, vitalParamMsg, analyseData, sendAlarm, alarm, callFAS, attendToPA, pButtonMsg, alarm, failedAlarm, failure
starAssistance, vitalParamMsg, analyseData, chargeDrug, notifyPA, vitalParamMsg, analyseData, sendAlarm, alarm, callFAS, attendToPA, pButtonMsg, alarm, callFAS, attendToPA, stopMsg, exit
starAssistance, vitalParamMsg, analyseData, chargeDrug, notifyPA, vitalParamMsg, analyseData, sendAlarm, alarm, callFAS, attendToPA, vitalParamMsg, analyseData, chargeDrug, failureChargeDrug, failure

Fonte – Elaborado pelo autor

Na fase de análise, o usuário realizou testes no código com cada cenário implícito identificado e verificou que todos eles são comportamentos possíveis na aplicação. Portanto, a

Figura 19 – Cenários implícitos tele assistance



Fonte – Elaborado pelo autor

modelagem comportamental não está errada e os rastros inferidos são cenários positivos. Um dos motivos para que possa ter ocorrido a presença desses cenários é o número baixo de rastros de execuções usados para construir o modelo comportamental. Outra possibilidade é que esses rastros identificados são bem extensos, representando usos possíveis de várias funcionalidades em sequência.

Então, para finalizar o processo, o usuário selecionou todos os cenários a partir da opção *Select* e, logo em seguida, a opção *Positive*. Assim todos os rastros positivos são considerados válidos no modelo final da abordagem que, neste caso, é o próprio modelo inicial ilustrado na Figura 18. Com isso, nenhum refinamento foi necessário, finalizando o processo.

5.3 SISTEMA DE CONTROLE DE VENDAS

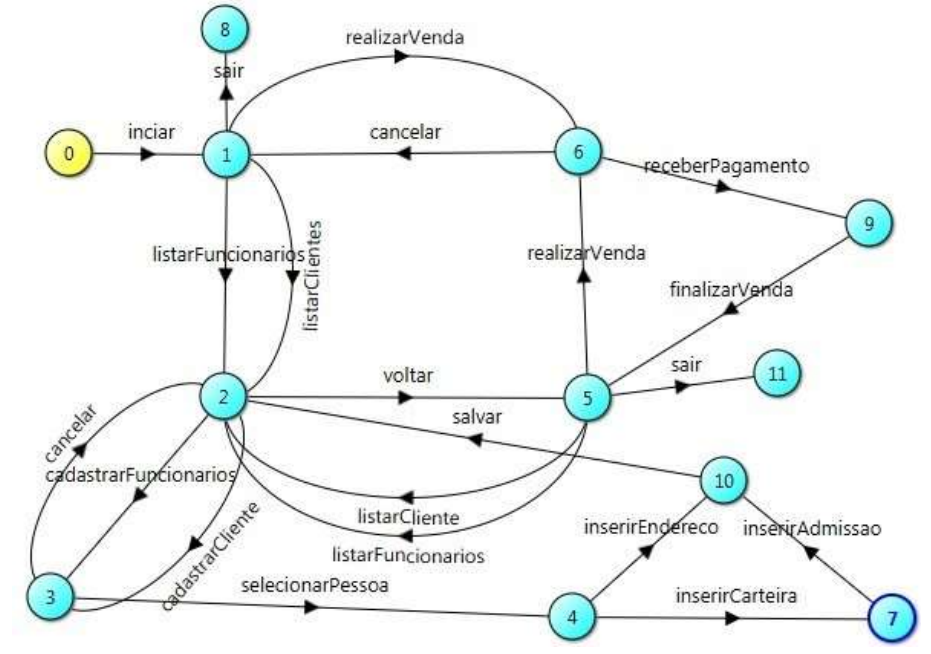
No terceiro exemplo foi utilizado um sistema real de controle de vendas disponível no Github [1]. O sistema possui como principais funcionalidades o cadastro e a busca de funcionários, clientes, produtos, e vendas. Tanto no cadastro de clientes quanto no de funcionários, é necessário primeiramente selecionar uma pessoa, que foi previamente cadastrada com seus dados pessoais, para em seguida adicionar informações específicas. No primeiro caso, insere-se o endereço do cliente, enquanto no segundo caso, insere-se o número da carteira de trabalho e a data de

[1] http://bit.ly/2vgOKnM

admissão do funcionário.

Para utilizar a abordagem, um desenvolvedor (que não é o autor da aplicação e nem a autora desta dissertação) estudou o código do sistema e realizou anotações de *log* em suas principais ações para que, quando executado, os *logs* gerassem um *trace* do sistema. Em seguida, o sistema foi executado 1000 vezes, o que gerou um arquivo *csv* com o mesmo número de *traces*. Como não havia modelo inicial do sistema, foi utilizado o plugin do LoTuS "Model from Trace", que gerou justamente o modelo apresentado na Figura 20.

Figura 20 – Modelo de comportamento inicial do sistema de vendas



Fonte – Elaborado pelo autor

Com esse modelo e o conjunto de *traces* de entrada, iniciou-se a fase 1 da abordagem. Neste momento, foram encontrados 18 cenários implícitos. Na fase de análise, o usuário classificou 9 (nove) como positivos, ou seja, são execuções válidas do sistema mas que não estavam contidos nos rastros iniciais e, portanto, não representam erros (Tabela 5). Um exemplo de cenário positivo detectado é *iniciar, listarFuncionarios, voltar, realizarVenda, cancelar, listarFuncionarios, voltar, realizarVenda, receberPagamento finalizarVenda, realizarVenda, cancelar, sair*. Possivelmente eles aconteceram devido a um baixo número de execuções do sistema e que não necessariamente exercitaram todas as possibilidades de comportamento do mesmo.

Tabela 5 – Cenários Implícitos Positivos - Modelo Vendas

Cenários Implícitos detectados
iniciar, listarFuncionarios, voltar, realizarVenda, cancelar, listarFuncionarios, voltar, realizarVenda, receberPagamento finalizarVenda, realizarVenda, cancelar, sair
iniciar, listarFuncionarios, voltar, realizarVenda, cancelar, listarFuncionarios, voltar, realizarVenda, receberPagamento finalizarVenda, realizarVenda, cancelar, listarFuncionarios, cadastrarFuncionario, selecionarPessoa, inserirCarteira, inserirDataAdmissao, salvar, voltar, realizarVenda, cancelar, sair
iniciar, listarFuncionarios, voltar, realizarVenda, cancelar, listarFuncionarios, voltar, realizarVenda, receberPagamento finalizarVenda, realizarVenda, cancelar, listarFuncionarios, cadastrarFuncionario, selecionarPessoa, inserirCarteira, inserirDataAdmissao, salvar, voltar, realizarVenda, cancelar, listarClientes, voltar, realizarVenda, cancelar, sair
iniciar, listarFuncionarios, voltar, realizarVenda, cancelar, listarFuncionarios, voltar, realizarVenda, receberPagamento finalizarVenda, realizarVenda, cancelar, listarFuncionarios, cadastrarFuncionario, selecionarPessoa, inserirCarteira, inserirDataAdmissao, salvar, voltar, realizarVenda, cancelar, listarClientes, voltar, realizarVenda, cancelar, realizarVenda, cancelar, sair
iniciar, listarFuncionarios, voltar, realizarVenda, cancelar, listarFuncionarios, voltar, realizarVenda, receberPagamento finalizarVenda, realizarVenda, cancelar, listarFuncionarios, cadastrarFuncionario, selecionarPessoa, inserirCarteira, inserirDataAdmissao, salvar, voltar, realizarVenda, cancelar, listarClientes, voltar, realizarVenda, cancelar, realizarVenda, cancelar, listarFuncionarios, voltar, realizarVenda, receberPagamento finalizarVenda, realizarVenda, cancelar, sair
iniciar, listarFuncionarios, voltar, realizarVenda, cancelar, listarFuncionarios, voltar, realizarVenda, receberPagamento finalizarVenda, realizarVenda, cancelar, listarFuncionarios, cadastrarFuncionario, selecionarPessoa, inserirCarteira, inserirDataAdmissao, salvar, voltar, realizarVenda, cancelar, listarClientes, voltar, realizarVenda, cancelar, realizarVenda, cancelar, listarFuncionarios, voltar, realizarVenda, receberPagamento finalizarVenda, realizarVenda, cancelar, listarFuncionarios, cadastrarFuncionario, selecionarPessoa, inserirEndereco, salvar, voltar, realizarVenda, cancelar, sair
iniciar, listarFuncionarios, voltar, realizarVenda, cancelar, listarFuncionarios, voltar, realizarVenda, receberPagamento finalizarVenda, realizarVenda, cancelar, listarFuncionarios, cadastrarFuncionario, selecionarPessoa, inserirCarteira, inserirDataAdmissao, salvar, voltar, realizarVenda, cancelar, listarClientes, voltar, realizarVenda, cancelar, realizarVenda, cancelar, listarFuncionarios, voltar, realizarVenda, receberPagamento finalizarVenda, realizarVenda, cancelar, listarFuncionarios, cadastrarFuncionario, selecionarPessoa, inserirEndereco, salvar, voltar, realizarVenda, cancelar, realizarVenda, receberPagamento finalizarVenda, realizarVenda, cancelar, sair
iniciar, listarFuncionarios, voltar, realizarVenda, cancelar, listarFuncionarios, voltar, realizarVenda, receberPagamento finalizarVenda, realizarVenda, cancelar, listarFuncionarios, cadastrarFuncionario, selecionarPessoa, inserirCarteira, inserirDataAdmissao, salvar, voltar, realizarVenda, cancelar, listarClientes, voltar, realizarVenda, cancelar, realizarVenda, cancelar, listarFuncionarios, voltar, realizarVenda, receberPagamento finalizarVenda, realizarVenda, cancelar, listarFuncionarios, cadastrarFuncionario, selecionarPessoa, inserirEndereco, salvar, voltar, realizarVenda, cancelar, realizarVenda, receberPagamento finalizarVenda, realizarVenda, cancelar, listarFuncionarios, cadastrarFuncionario, cancelar, voltar, realizarVenda, cancelar, sair
iniciar, listarFuncionarios, voltar, realizarVenda, cancelar, listarFuncionarios, voltar, realizarVenda, receberPagamento finalizarVenda, realizarVenda, cancelar, listarFuncionarios, cadastrarFuncionario, selecionarPessoa, inserirCarteira, inserirDataAdmissao, salvar, voltar, realizarVenda, cancelar, listarClientes, voltar, realizarVenda, cancelar, realizarVenda, cancelar, listarFuncionarios, voltar, realizarVenda, receberPagamento finalizarVenda, realizarVenda, cancelar, listarFuncionarios, cadastrarFuncionario, selecionarPessoa, inserirEndereco, salvar, voltar, realizarVenda, cancelar, realizarVenda, receberPagamento, finalizarVenda, realizarVenda, cancelar, listarFuncionarios, cadastrarFuncionario, cancelar, voltar, realizarVenda, cancelar,realizarVenda, receberPagamento, finalizarVenda, realizarVenda, cancelar, sair

Fonte – Elaborado pelo autor

Por outro lado, o usuário também identificou 9 (nove) cenários como negativos (Tabela 6). Contudo, o usuário pôde observar que esses cenários acontecem devido a duas situações: a primeira é que, após selecionar “cadastrar funcionário”, o modelo indica que é possível inserir dados de cliente (*...,cadastrarFuncionario, selecionarPessoa, inserirEndereço,*

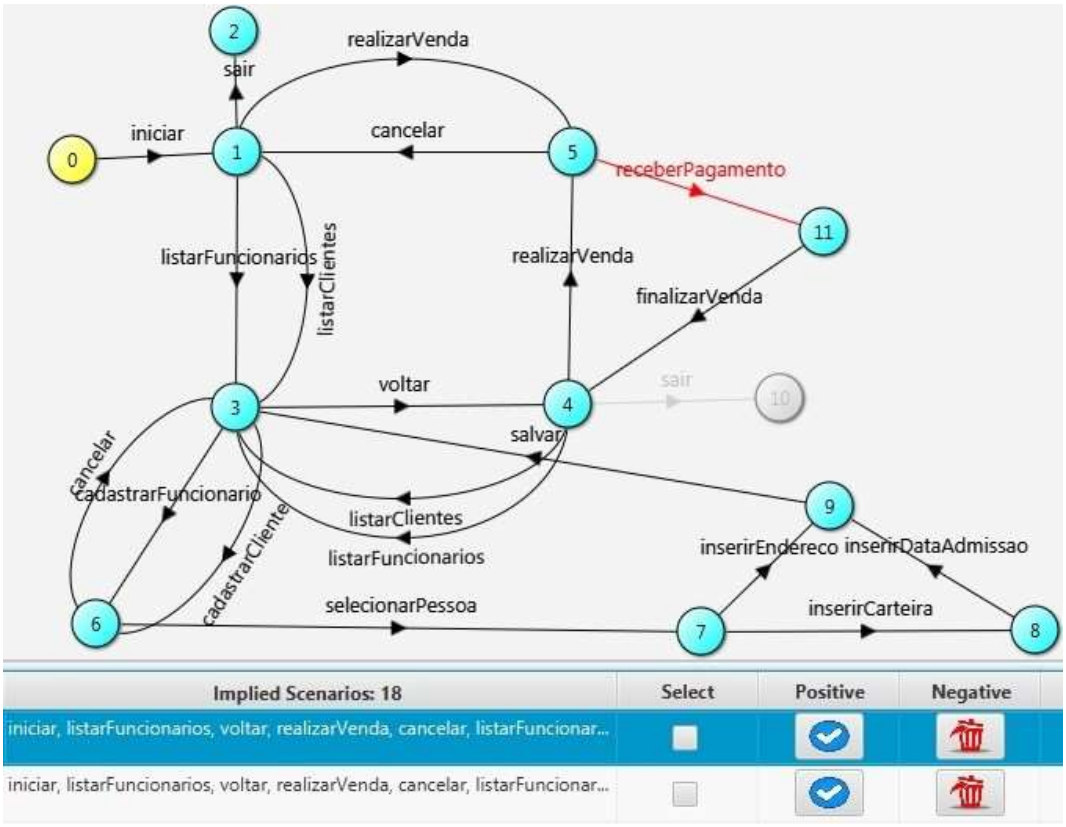
salvar...). Já a segunda é justamente o contrário: após selecionar “cadastrar cliente”, o modelo indica que é possível a entrada dos dados do funcionário (*...,cadastrarCliente, selecionarPessoa, inserirCarteira, inserirAdmissão, salvar...*) (Figura 21). O número maior de cenários são gerados devido à grande quantidade de opções de ações que podem ser feitas antes de se chegar a esses cenários (por exemplo, começar com uma venda ou listando funcionário ou cliente). Para se certificar se os cenários negativos eram problema do código da aplicação ou do gerador do modelo, o usuário tentou realizar as situações descritas anteriormente no sistema em execução. Como o sistema não permitiu essa possibilidade, o usuário concluiu que o problema estava na geração do modelo.

Tabela 6 – Cenários Implícitos Negativos- Modelo Vendas

Cenários Implícitos detectados
iniciar, listarFuncionarios, voltar, realizarVenda, cancelar, listarFuncionarios, voltar, realizarVenda, receberPagamento finalizarVenda, realizarVenda, cancelar, listarFuncionarios, cadastrarFuncionario, selecionarPessoa, (...)
iniciar, listarFuncionarios, voltar, realizarVenda, cancelar, listarFuncionarios, voltar, realizarVenda, receberPagamento finalizarVenda, realizarVenda, cancelar, listarFuncionarios, cadastrarFuncionario, (...)
iniciar, listarFuncionarios, voltar, realizarVenda, cancelar, listarFuncionarios, voltar, realizarVenda, finalizarVenda, realizarVenda, cancelar, listarFuncionarios, cadastrarFuncionario, (...)
iniciar, listarFuncionarios, voltar, realizarVenda, cancelar, listarFuncionarios, voltar, realizarVenda, receberPagamento finalizarVenda, realizarVenda, cancelar, listarFuncionarios, cadastrarFuncionario, selecionarPessoa, inserirCarteira, inserirDataAdmissao, salvar, voltar, realizarVenda, cancelar, listarClientes, voltar, realizarVenda, cancelar, realizarVenda, cancelar, listarFuncionarios, voltar, realizarVenda, receberPagamento finalizarVenda, realizarVenda, cancelar, listarFuncionarios, cadastrarFuncionario, (...)
iniciar, listarFuncionarios, voltar, realizarVenda, cancelar, listarFuncionarios, voltar, realizarVenda, receberPagamento finalizarVenda, realizarVenda, cancelar, listarFuncionarios, cadastrarFuncionario, selecionarPessoa, inserirCarteira, inserirDataAdmissao, salvar, voltar, realizarVenda, cancelar, listarClientes, voltar, realizarVenda, cancelar, realizarVenda, cancelar, listarFuncionarios, voltar, realizarVenda, receberPagamento finalizarVenda, realizarVenda, cancelar, listarFuncionarios, cadastrarFuncionario, selecionarPessoa, inserirEndereco, salvar, voltar, realizarVenda, cancelar, realizarVenda, (...)
(...), cadastrarFuncionario, selecionarPessoa, inserirEndereco, salvar, voltar, realizarVenda, cancelar, realizarVenda, receberPagamento, finalizarVenda, realizarVenda, cancelar, listarFuncionarios, cadastrarFuncionario, cancelar, voltar, realizarVenda, cancelar, realizarVenda, receberPagamento finalizarVenda, realizarVenda, cancelar, listarFuncionarios, cadastrarCliente, selecionarPessoa, inserirCarteira, (...)
(...), finalizarVenda, realizarVenda, cancelar, listarFuncionarios, cadastrarFuncionario, cancelar, voltar, realizarVenda, cancelar, realizarVenda, receberPagamento finalizarVenda, realizarVenda, cancelar, listarFuncionarios, cadastrarCliente, selecionarPessoa, inserirCarteira, inserirDataAdmissao, (...)
(...), inserirDataAdmissao, salvar, voltar, realizarVenda, cancelar, listarClientes, voltar, realizarVenda, cancelar, realizarVenda, cancelar, listarFuncionarios, voltar, realizarVenda, receberPagamento finalizarVenda, realizarVenda, cancelar, listarFuncionarios, cadastrarFuncionario, selecionarPessoa, inserirEndereco, (...)
(...), cadastrarFuncionario, selecionarPessoa, inserirCarteira, inserirDataAdmissao, salvar, voltar, realizarVenda, cancelar, listarClientes, voltar, realizarVenda, cancelar, realizarVenda, cancelar, listarFuncionarios, voltar, realizarVenda, receberPagamento finalizarVenda, realizarVenda, cancelar, listarFuncionarios, cadastrarFuncionario, selecionarPessoa, inserirEndereco, (...)

Fonte – Elaborado pelo autor

Figura 21 – Cenário Negativo - Modelo vendas

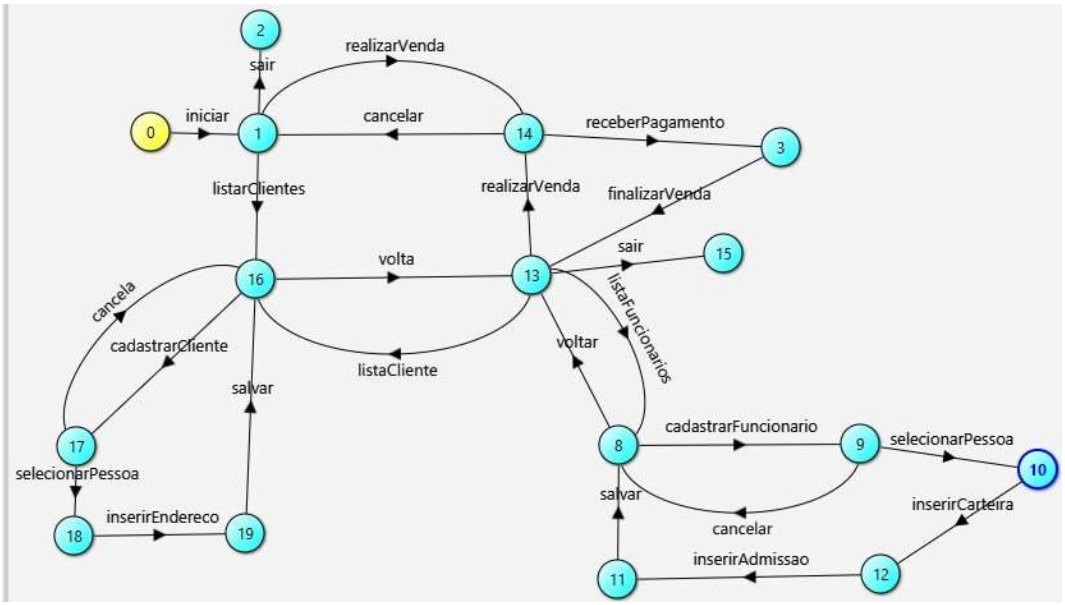


Fonte – Elaborado pelo autor

Na fase de refinamento, o usuário selecionou todos os cenários negativos e solicitou sua remoção. Seguindo a abordagem proposta, o modelo foi primeiramente expandido, em seguida os *traces* que reproduzem os cenários negativos foram removidos. Em seguida, o usuário selecionou todos os cenários positivos a partir da opção *Select* e, logo em seguida, a opção *Positive*. Dessa forma todos os *traces* positivos que forem considerados válidos são inseridos no modelo final da abordagem. E, por fim, o modelo foi minimizado, gerando assim um novo modelo refinado, que é apresentado na Figura 22.

O modelo então foi apresentado ao usuário, que solicitou novamente a detecção de cenários implícitos (fase 1 da abordagem), para tentar identificar novamente possíveis falhas. Porém, para o novo modelo, nenhum cenário implícito foi gerado. Portanto, o modelo refinado representa corretamente as funcionalidades do sistema, de acordo com o usuário. Com isso, o processo foi encerrado.

Figura 22 – Modelo vendas refinado



Fonte – Elaborado pelo autor

5.4 CONCLUSÃO DO CAPÍTULO

Este capítulo apresentou três exemplos de uso da abordagem proposta. O primeiro consistiu de uma adaptação de um sistema de edição de texto, pata o qual foram encontrados seis cenários implícitos, todos negativos, que foram devidamente removidos, gerando um novo modelo refinado.

O segundo dizia respeito a um sistema distribuído para monitoramento médico remoto. Nele, foram encontrados nove cenários implícitos mas, diferentemente do anterior, todos foram considerados positivos. Com isso, o modelo não precisou de refinamento.

O último era sobre um sistema de vendas na *web*, no qual um desenvolvedor conhecedor da aplicação participou da aplicação da abordagem. Foram detectados 18 cenários implícitos, sendo metade de cenários negativos e a outra metade de cenários positivos. O refinamento gerou um novo modelo que foi validado como correto pelo usuário.

6 CONCLUSÃO

Modelos comportamentais criados a partir de rastros de execução do sistema reproduzem o comportamento dinâmico do mesmo, incluindo os aspectos significativos e observáveis para descrever sua conduta. Entretanto, ao generalizar um comportamento global do sistema com base em um conjunto de amostras de comportamentos individuais, pode-se gerar modelos contendo condutas que não estão presentes nos *traces* originais do sistema, denominados neste trabalho de cenários implícitos. Estes podem representar tanto comportamentos válidos (cenários positivos), mas que não estavam presentes nos *traces* originais do sistema, ou comportamentos inválidos (cenários negativos), que não devem estar no modelo e que podem ter sido gerados por problemas na implementação do sistema ou no algoritmo de geração do modelo.

Nesse contexto, o presente trabalho apresentou uma abordagem semi-automática e interativa que identifica e remove cenários implícitos de modelos de comportamentos representados como Labelled Transition Systems. A abordagem, que possui três fases, utiliza o algoritmo *All One-loop Paths* como técnica para identificação dos cenários implícitos, o conceito de *man-in-loop* para introduzir o usuário especialista no processo de análise dos cenários, e técnicas para refinamento de modelo para remover os cenários indesejados. Uma implementação da abordagem na ferramenta LoTuS é também apresentada e disponibilizada para uso da comunidade. A aplicabilidade da abordagem foi demonstrada através de três exemplos de uso.

Pode-se concluir que a abordagem proposta é eficiente na detecção e remoção dos cenários implícitos. Através dela, o usuário consegue mais facilmente evoluir seu modelo de forma que ele contenha apenas comportamentos válidos para o sistema. Uma vantagem é que, apesar de usar um modelo formal de representação de comportamento do sistema, o usuário da abordagem não precisa conhecer mais profundamente outros métodos formais para manipulação do modelo, pois isso tudo é feito através de interface gráfica.

6.1 CONTRIBUIÇÕES

Visando consolidar as ideias apresentadas neste trabalho, lista-se a seguir as principais contribuições advindas da presente pesquisa:

(a) **Proposta de abordagem:** desenvolveu-se uma nova abordagem semi-automática para apoiar a descoberta e extração de cenários implícitos em modelos de comportamento gerados a partir de *traces* do sistema com a participação ativa de um usuário especialista no sistema.

(b) **Detecção dos cenários implícitos:** para detectar a presença de possíveis incorreções no modelo de comportamento, utilizou-se o algoritmo *All One-Loop Paths*, comum na área de testes baseados em modelos.

(c) **Refinamento:** elaborou-se uma técnica baseada na perspectiva do usuário, para que possa ser realizado o refinamento do modelo comportamental. O objetivo principal é remover os rastros de execução que não estão de acordo com o real funcionamento do sistema sob análise.

(d) **Suporte ferramental:** foi desenvolvida uma ferramenta como um plugin para o LoTuS que implementa todas as fases da abordagem proposta.

6.2 LIMITAÇÕES

Como limitações, pode-se citar que a abordagem é mais apropriada para modelos que contenham poucos *loops*, pois quanto maior o número de *loops*, maior também será o esforço computacional para extrair todos os possíveis comportamentos. Isto se reflete em uma demora de processamento no LoTuS e, muitas vezes, estouro de memória.

Também ressalta-se que para o caso de modelos muito grandes, a técnica de refinamento pode gerar problemas, pois a abertura do modelo gera um novo modelo bem maior. Logo, quanto maior o modelo inicial, maior serão os modelos abertos para o refinamento, o que pode também gerar uma sobrecarga de memória.

Outra limitação está na forma de visualização dos cenários implícitos, pois muitas vezes o cenário é muito grande e isso dificulta o seu entendimento por parte do usuário.

6.3 TRABALHOS FUTUROS

Como trabalhos futuros, planeja-se otimizar a técnica de remoção para suportar modelos maiores e mais complexos. Também há a intenção de melhorar a forma de visualização para facilitar a classificação dos cenários por parte do usuário. Com isso, será possível fazer novos estudos com sistema reais.

Por fim, pretende-se testar a abordagem com outras técnicas de extração dinâmica de modelos a fim de verificar o impacto que isso tem na detecção de cenários implícitos.

REFERÊNCIAS

ALUR, R.; ETESSAMI, K.; YANNAKAKIS, M. Inference of message sequence charts. **IEEE Transactions on Software Engineering**, v. 29, n. 7, p. 623–633, 2003.

ANGLUIN, D. Learning regular sets from queries and counterexamples. **Inf. Comput.**, v. 75, n. 2, p. 87–106, nov. 1987.

BALL, T.; RAJAMANI, S. K. The s lam project: debugging system software via static analysis. In: ACM. **ACM SIGPLAN Notices**. [S.l.], v. 37, n. 1, p. 1–3, 2002.

INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING, 33, 2011, Hawaii. Interface decomposition for service compositions. IEEE, 2011.

BOOCH, G.; RUMBAUGH, J.; JACOBSON, I. UML: guia do usuário. [S.l.]: Elsevier Brasil, 2006.

BRAUN, E. **Reverse engineering behavioural models by filtering out utilities from execution traces**. Tese (Doutorado) — Université d'Ottawa/University of Ottawa, 2013.

BUHR, R. J. Use case maps as architectural entities for complex systems. **IEEE Transactions on Software Engineering**, v. 24, n. 12, p. 1131–1155, 1998.

CASTEJÓN, H. N.; BRÆK, R. A collaboration-based approach to service specification and detection of implied scenarios. **Proceedings of the 2006 international workshop on Scenarios and state machines:** models, algorithms, and tools. [S.l.], 2006. p. 37–43.

CHAKI, S.; CLARKE, E. M.; GROCE, A.; JHA, S.; VEITH, H. Modular verification of software components in c. **IEEE Transactions on Software Engineering**, v. 30, n. 6, p. 388–402, 2004.

CLARKE, E.; WING, J. Formal methods: State of the art and future directions. **ACM Computing Surveys,** v. 28, n. 4, p. 626–643, 1996.

COOK, J. E.; WOLF, A. L. Discovering models of software processes from event-based data. **ACM Trans. Softw. Eng. Methodol.** v. 7, n. 3, p. 215–249, jul. 1998.

PROCEEDINGS INTERNATIONAL CONFERENCE, 15, 2000. **Bandera: Extracting finite-state models from java source code**. [S.l] IEEE. 2000.

THEORETICAL COMPUTER SCIENCE (WEIT). 2, 2013. **Behaviour model extraction from software.** [S.l.] ACM. 2013.

DUARTE, L. M.; KRAMER, J.; UCHITEL, S. Model extraction using context information. **Model Driven Engineering Languages and Systems**, [S.l.]: Springer, 2006. p. 380–394.

INTERNATIONAL CONFERENCE ON FUNDAMENTAL APPROACHES TO SOFTWARE ENGINEERING, 13, 2008, Hungary. Towards faithful model extraction based on con- texts. Hungary: SPRINGER, 1994.

DURY, A.; HALLAL, H. H.; PETRENKO, A. Inferring behavioural models from traces of business applications. **Web Services, 2009. ICWS 2009. IEEE International Conference on**. [S.l.], 2009. p. 791–798.

PROCEEDINGS OF TH INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING, 31, 2009, Washington. Model evolution by runtime parameter adaptation. Ieee Computer Society, Washington. 2009.

GEER, P. A. **Formal Methods in Practice: Analysis and Application of Formal Modeling to Information Systems**. Tese (Doutorado) — State University of New York Institute of Technology, 2011.

GIANNAKOPOULOU, D. **Model checking for concurrent software architectures**. Tese (Doutorado) — Imperial College, London, 1999.

GROUP, O. M. *et al.*. Unified Modeling Language 2. OMG. [S.l] 2012.

CONFERENCE ON SOFTWARE ENGINEERING, 24., 2002, Tracking down software bugs using automatic anomaly detection. ACM. [S.l.], 2002.

HENDRIKS, M.; VERRIET, J.; BASTEN, T.; THEELEN, B.; BRASSÉ, M.; SOMERS, L. Analyzing execution traces: critical-path analysis and distance analysis. **International Journal on Software Tools for Technology Transfer**, Springer, p. 1–24, 2016.

INTERNATIONAL SPIN WORKSHOP ON MODEL CHECKING OF SOFTWARE. 10, 2003, Software verification with blast. SPRINGER. [S.l.], 2003.

International Conference on Software Engineering. 21, 1990. A PRACTICAL METHOD FOR VERIFYING EVENT-DRIVEN SOFTWARE. ACM, [S.l.], 1999.

KELLER, R. M. Formal verification of parallel programs. **Communications of the ACM**, ACM, v. 19, n. 7, p. 371–384, 1976.

KING, J. C. Symbolic execution and program testing. **Communications of the ACM**, ACM, v. 19, n. 7, p. 385–394, 1976.

CONFERENCE ON HUMAN FACTORS IN COMPUTING SYSTEMS, 21, 2016, Using runtime traces to improve documentation and unit test authoring for dynamic languages. [S.l.], ACM. 2016.

SOFTWARE ENGINEERING, 2010 ACM/IEEE 32ND INTERNATIONAL CONFERENCE, 32, 2010, Using dynamic execution traces and program invariants to enhance behavioral model inference. [S.l.], IEEE. 2010.

INTERNATIONAL CONFERENCE ON FORMAL ENGINEERING METHODS, 6, 2004 Counterexample guided abstraction refinement via program execution. [S.l.] SPRINGER, 2004.

INTERNATIONAL SYMPOSIUM ON REQUIREMENTS ENGINEERING, 93, Annapolis. Enhancing a requirements baseline with scenarios. Annapolis: IEEE, 1993.

INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING, 30, 2008. Automatic generation of software behavioral models. [S.l.]: ACM, 2008.

MAGEE, J.; KRAMER, J. **Concurrency:** State Models and Java Programs. [S.l.]: Wiley, 2006.

MAIA, P. H. M. **Improving the accuracy of probablastic behaviour models using state refinement**. Tese (Doutorado) — Imperial College London, London, 2011.

MAIA, P. H. M.; FREITAS, F.; BORGES, M.; MUNIZ, L. L.; SILVA, A. S. da; XIMENES, J. Práticas e experiencias no ensino de engenharia dirigida por modelos. **iSys - Revista Brasileira de Sistemas de Informação**, v. 9, n. 2, p. 106–135, 2016.

INTERNATIONAL CONFERENCE OF THE AUSTRIAN CENTER FOR PARALLEL COMPUTATION. 1, 1991Traceview: A trace visualization tool. [S.l.], SPRINGER., 1991.

INTERNATIONAL CONFERENCE ON MODEL DRIVEN ENGINEERING LANGUAGES AND SYSTEMS, 11, 2008. Model-based traces. [S.l.], SPRINGER, 2008.

MEDVIDOVIC, N.; ROSENBLUM, D. S.; REDMILES, D. F.; ROBBINS, J. E. Modeling software architectures in the unified modeling language. **Transactions on Software Engineering and Methodology (TOSEM)**, ACM, v. 11, n. 1, p. 2–57, 2002.
MIHANCEA, P. F.; MINEA, M. Jmodex: Model extraction for verifying security properties of web applications. **Analysis**, v. 19, p. 20, 2014.

INTERNATIONAL CONFERENCE ON FUNDAMENTAL APPROACHES TO SOFTWARE ENGINEERING, 6, 2003. Detecting implied scenarios analyzing non-local branching choices. [S.l.], SPRINGER. 2003.

MUNIZ, L. L.; NETTO, U. S.; MAIA, P. H. M. Lotus-tcg: uma ferramenta para geração e seleção de casos de teste funcionais e estatísticos. **SAST 2014**, p. 91, 2015.

NIMMER, J. W.; ERNST, M. D. Automatic generation of program specifications. **ACM SIGSOFT Software Engineering Notes**, v. 27, n. 4, p. 229–239, 2002.

PARKER, D. Verification of probabilistic real-time systems. **Proc. 2013 Real-time Systems Summer School (ETR'13)**, 2013.

RAMCHANDANI, D.; RUSSO, A.; ALRAJEH, D. Refining labelled transition systems using scenario-based specifications. London: **Imperial College London,** Department of Computing, 2009.

ROBERTSON, S.; ROBERTSON, J. **Mastering the requirements process. 1º**. [S.l.]: ACM Press, 1999.

SOFTWARE COMPONENTS, ARCHITECTURES AND REUSE, 8, 2014, BRAZILIAN. Analysis of the impact of implied scenarios on the reliability of computational concurrent systems. BRAZILIAN: IEEE, 2014.

SELIC, B. The pragmatics of model-driven development. **IEEE software**, n. 5, p. 19–25, 2003.

SOMMERVILLE, I. **Software Engineering**. [S.l.]: Pearson, 2011. v. 9.

SOUZA, F. C. de; MENDONÇA, N. C. Um ambiente para detecção de cenários implícitos a partir de rastros de execução. **XXI Simpósio Brasileiro de Engenharia de Software,** 2007.

SYSTEMS, S. **Using UML Part Two**: behavioral modeling diagrams. [S.l.]: Enterprise Architect, 2007. p. 5-10.

PROCEEDINGS OF THE INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING, 29, 2007. Behaviour model synthesis from properties and scenarios. [S.l], IEEE 2007.

INTERNATIONAL CONFERENCE ON TOOLS AND ALGORITHMS FOR THE CONSTRUCTION AND ANALYSIS OF SYSTEMS, 9, 2003. Ltsa-msc: Tool support for behaviour model elaboration using implied scenarios. [S.l] SPRINGER, 2003.

UCHITEL, S.; KRAMER, J.; MAGEE, J. Detecting implied scenarios in message sequence chart specifications. ACM. **ACM SIGSOFT Software Engineering Notes**, [S.l.], v. 26, n. 5, p. 74–82, 2001.

UCHITEL, S.; KRAMER, J.; MAGEE, J. **Incremental elaboration of scenario-based specifi- cations and behavior models using implied scenarios**. 85 p. Tese (Doutorado), London, 2004.

UCHITEL, S.; KRAMER, J.; MAGEE, J. Incremental elaboration of scenario-based specificati- ons and behavior models using implied scenarios. **ACM Transactions on Software Engineering and Methodology**, v. 13, n. 1, p. 37–85, 2004.

UTTING, M.; LEGEARD, B. **Practical model-based testing:** a tools approach. [S.l.]: Morgan Kaufmann, 2010. 119–120 p.

VITAL, R. B. N.; VITAL, T. M. Utilização da modelagem uml em um sistema de gerenciamento de uma franquia do setor de alimentação. **Revista Eletrônica TECCEN**, v. 8, n. 2, p. 65–72, 2015.

PROCEEDINGS OF THE INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING, 28, 2006, Perracotta: mining temporal api rules from imperfect traces. [S.l.], ACM., 2006.

APÊNDICE

APÊNDICE A – Pseudocodigo

Algorithm Refinement

Input: pathsOLP; scenarioPositive; List pathsWithoutLoop; List pathsWithLoop;

output: refined LTS

Loop Detection

---

for All path ∈ pathsOLP DO
containsLoop = false

for All label ∈ path DO
transition = label

for All labelAux ∈ path DO
labelAfter = labelAux.next

if transition = transitionAfter

containsLoop = true

break

if containsLoop = true

break

if containsLoop = true

pathsWithLoop.add(path)

else pathsWithoutLoop.add(path)

Union of Transitions

---

for All path E pathsWithLoop DO

for i to path.size DO

transitionActual = path[i]

if transitionoActual exist

for j to i DO

pathPrevious = path[j]

if transitionoActual = pathPrevious

```
transitionActual.previous.turnson(pathPrevious)
pathsNegative = OLP(path).negative
if pathsNegative.size > 0
transitionActual.previous.turnsoff(pathPrevious)
transitionActual.create
break
else
transitionActual.create
for All path E pathsWithoutLoop DO
for i to path.size DO
transitionActual = path[i]
if transitionActual exist
transitionActual.turnson(path[i+1])
pathsNegative = OLP(path).negative
if pathsNegative.size > 0
transitionActual.turnsoff(path[i+1)
transitionActual.create
break
else
transitionActual.create
```

Detection of implicit scenarios

```
check.ImplicitScenarios(if repetition 1 to repetition 2 then loop)
boolean checkLoopPath(String path)
boolean contains = false;
String[] label =
path.split(",");
for(i de 0, ate i < label[i].size, i++)
String transicao = label[i];
if (i < label.size)
for(j de (i+1), to j <label.size, j++)
String transitionAfter = label[j];
```

if(transicao = transitionAfter)

contains = true;

exit loop;

if (contains = true)

exit loop;

returns contains;

for(i de 0, i<pathsOLP.size, i++) boolean containsLoop=checkLoopPath(pathOLP.get(i));

if(containsLoop) pathsWithLoop.add(pathsOLP.get(i));

else

pathsWithoutLoop.add(pathsOLP.get(i));

for (i to 0, i < pathsWithLoop.size, i++)

String[] labelActual = pathsWithLoop.get(i).split(",");

for(i de 0, to i <labelActual[i].size, i++)

String transition = labelActual[i];

if transition exist step **if** not create transition

if (i < labelActual.size)

for (j de (i+1), to j < labelActual.size, j++)

String transitionAfter = labelActual[j];

if (transition = transitionAfter)

contains = true;

exit loop;

if (contains = true)

exit loop;
