Um Ambiente para Detecção de Cenários Implícitos a partir de Rastros de Execução

Felipe Cantal de Souza, Nabor C. Mendonça

Mestrado em Informática Aplicada, Universidade de Fortaleza
Av. Washington Soares, 1321, CEP 60811-905 Fortaleza – CE
felipe_cantal@yahoo.com.br, nabor@unifor.br

Resumo. *Um cenário descreve como um ou mais componentes de um sistema interagem para oferecer um conjunto de funcionalidades. Devido a cada cenário representar apenas uma visão parcial do comportamento global do sistema, especificações baseadas em cenários podem esconder comportamentos inusitados, denominados "cenários implícitos", não previstos nos cenários originais. A presença de cenários implícitos tanto pode indicar falhas na especificação do sistema, como comportamentos indesejados a serem evitados. Este trabalho apresenta um ambiente de engenharia reversa para apoiar a extração e detecção de cenários implícitos a partir de rastros de execução. A principal contribuição do trabalho é permitir que os desenvolvedores se beneficiem do conceito de cenários implícitos, até então restrito às fases iniciais do ciclo de vida de software, também para apoiar atividades de compreensão e teste de sistemas existentes.*

Abstract. *A scenario describes how one or more system components interact to provide a certain set of functionalities. Because scenarios only represent partial views of the system's global behavior, scenario-based specifications may hide unexpected behaviors, called "implied scenarios", which are not expressed in any scenario individually. The presence of implied scenarios may well indicate errors in the system's original specification, or undesired behavior that should be avoided. This work presents a reverse engineering environment to support extraction and detection of implied scenarios from execution traces. The main contribution of the work is to allow developers to benefit from the concept of implied scenarios, which thus far has been restricted to the early phases of the software life-cycle, also to support comprehension and testing of existing systems.*

1. Introdução

Especificações baseadas em cenários, geralmente expressas utilizando notações como *Message Sequence Charts* (MSC's) [9] e Diagramas de Seqüência [12], têm sido cada vez mais empregadas em ambientes corporativos para a descrição de requisitos de software. Um cenário descreve como um ou mais componentes de um sistema, seu ambiente externo, e seus usuários concorrem e interagem para oferecer um conjunto específico de funcionalidades. Dessa forma, cada cenário representa uma visão parcial do comportamento global do sistema, o qual, para ser entendido por completo, depende da combinação dessas diferentes visões [23].

Cenários têm se mostrado uma forma intuitiva e flexível de representar os diferentes fluxos de mensagens entre componentes que caracterizam a execução de um sistema concorrente [10]. Por outro lado, esse tipo de especificação sofre de uma limitação intrínseca, cujos efeitos só começaram a ser adequadamente investigados e entendidos mais recentemente. A limitação está no fato de que um modelo de comportamento de um sistema nem sempre é a soma exata do conjunto de comportamentos expressos nas especificações de seus cenários individuais. Combinações inesperadas no modo como os componentes interagem podem forçar o aparecimento de comportamentos inusitados, não presentes nos cenários originais. Tais comportamentos, denominados cenários "decorrentes" ou "implícitos" (do inglês *implied scenarios*) no trabalho pioneiro de Alur *et al.* [1], surgem principalmente porque os componentes têm em geral uma visão local do que está acontecendo no sistema, e não uma visão do comportamento global esperado. Um processo automatizado e mais abrangente para a detecção de cenários implícitos durante a fase de elaboração de requisitos foi posteriormente proposto por Uchitel *et al.* [20][21].

A existência de cenários implícitos tanto pode indicar omissões na especificação do sistema, no caso de cenários implícitos considerados válidos, mas que por algum motivo não foram originalmente especificados; ou simplesmente situações indesejadas não previstas nos cenários originais, no caso de cenários implícitos considerados inválidos do ponto de vista do comportamento esperado dos componentes [22]. Em ambos os casos, a detecção e a análise de potenciais cenários implícitos constituem uma atividade de fundamental importância para o processo de especificação de requisitos, seja para corrigir eventuais deficiências de especificação, seja para aumentar a confiança dos desenvolvedores nas especificações existentes.

Este trabalho apresenta um ambiente de engenharia reversa para apoiar a extração e detecção de cenários implícitos a partir de informações coletadas durante a execução de aplicações concorrentes existentes. O ambiente oferece um conjunto de ferramentas desenvolvidas ou reutilizadas com o objetivo de apoiar as diversas etapas envolvidas neste processo. Estas etapas incluem (i) a execução monitorada da aplicação alvo; (ii) a extração de cenários a partir dos rastros de execução gerados; e (iii) a descoberta e visualização de potenciais cenários implícitos tendo como base os cenários extraídos. Dessa forma, a principal contribuição do trabalho é viabilizar a utilização do conceito de cenários implícitos, até então restrita às fases iniciais do ciclo de vida de software, como mais um instrumento de apoio às atividades de compreensão, teste e manutenção de sistemas existentes.

A próxima seção dá uma visão geral do conceito de cenários implícitos e das ferramentas disponíveis para apoiar a sua detecção. A seção 3 descreve o processo de extração de cenários implícitos proposto, incluindo suas etapas e ambiente de suporte. A seção 4 apresenta os resultados de um estudo preliminar do uso do ambiente em uma aplicação de comércio eletrônico. A seção 5 discute os méritos e limitações da abordagem, a qual é comparada com alguns trabalhos relacionados na seção 6. Por fim, a seção 7 oferece algumas conclusões e sugestões para trabalhos futuros.

2. Cenários Implícitos

Conforme mencionado na seção anterior, cenários implícitos capturam comportamentos inusitados que não foram especificados explicitamente, mas que podem ocorrer devido à natureza parcial e concorrente das especificações baseadas em cenários [1]. Por essa

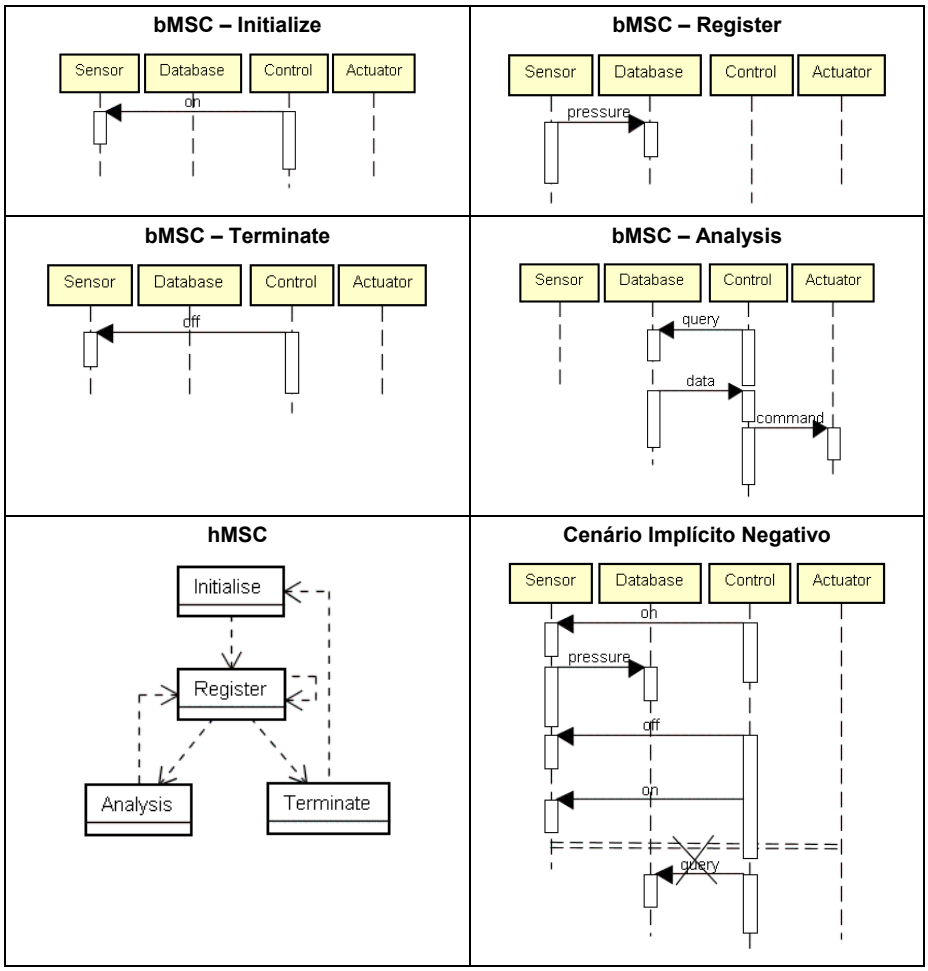

Figura 1. Exemplo de uma especificação de cenários contendo um cenário implícito.

razão, cenários implícitos precisam ser detectados, validados e categorizados (como positivos ou negativos [22]) no contexto do sistema estudado. Cenários implícitos negativos são situações não aceitáveis que demandariam desvios do comportamento esperado, ocasionando erros. Cenários implícitos positivos, por sua vez, também descrevem situações não previstas nas especificações iniciais, mas plenamente aceitáveis do ponto de vista do comportamento global esperado do sistema.

A ferramenta LTSA-MSC [19] foi a primeira a oferecer suporte automatizado para a detecção de cenários implícitos. Ela foi desenvolvida como uma extensão da ferramenta de verificação de modelos concorrentes LTSA (*Labelled Transition Systems Analyzer*) [10]. A extensão incorporada pela LTSA-MSC consiste numa linguagem baseada em diagramas MSCs para a descrição de cenários de alto nível (chamados hMSCs) a partir da composição paralela de cenários básicos (chamados bMSCs). Uma especificação formada por um cenário de alto nível e o conjunto de cenários básicos que o compõem constitui o insumo necessário para que a LTSA possa detectar a presença de cenários implícitos nessa especificação.

A Figura 1 mostra um exemplo (retirado de [21]) de uma especificação de cenários para um sistema de caldeira contendo um cenário implícito. No cenário básico *Initialize*, a mensagem do componente *Control* para o componente *Sensor* representa a ativação do segundo pelo primeiro. O cenário básico *Register* por sua vez representa a

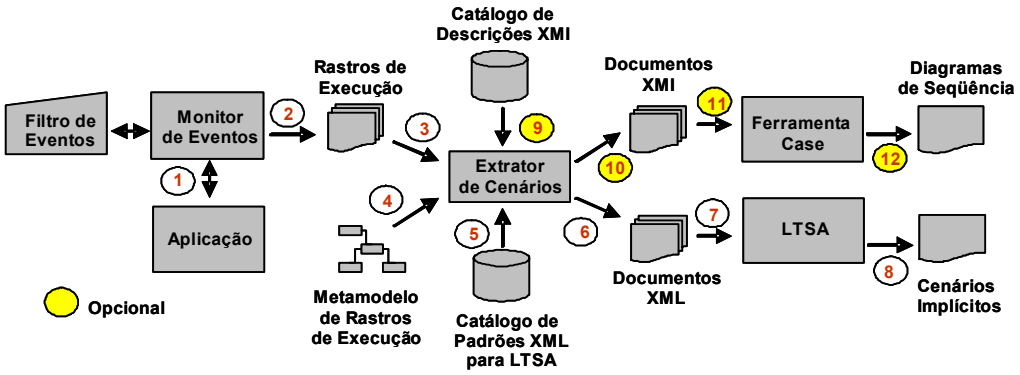

Figura 2. Processo de detecção de cenários implícitos em sistemas existentes.

ação de registro dos dados de pressão coletados pelo sensor no componente *Database*. O cenário básico *Analysis* representa uma seqüência de mensagens entre os componentes *Control*, *Database* e *Actuator*. Essas mensagens correspondem ao processo de análise do sistema, que pode resultar no aumento ou diminuição da temperatura da caldeira, dependendo dos valores de pressão coletados pelo sensor e armazenados no banco de dados. Por fim, o cenário básico *Terminate* representa a ação de desativação do sensor pelo componente de controle. O cenário de alto nível (hMSC) mostrado na figura define as possíveis relações de continuidade entre os quatro cenários básicos do sistema. Note que, do ponto de vista individual de cada cenário, todos os componentes apresentam comportamentos válidos.

No entanto, ao receber essa especificação como insumo, a ferramenta LTSA detecta a presença de pelo menos um cenário implícito, conforme mostrado na parte de baixo à direita da Figura 1. Esse cenário implícito revela que, devido à concorrência entre os componentes, o componente de controle corre o risco de acessar dados de pressão desatualizados, uma vez que é permitida a consulta ao banco de dados logo após a ativação do sensor, mesmo que este ainda não tenha registrado nenhum novo valor de pressão. Uma discussão mais aprofundada das razões que levam ao aparecimento deste e de outros cenários implícitos presentes nesse exemplo pode ser encontrada em [21].

3. Processo para Detecção de Cenários Implícitos em Sistemas Existentes

O processo de detecção de cenários implícitos proposto neste trabalho é realizado em 12 passos, organizados em quatro etapas (ver Figura 2): (i) seleção de cenários de interesse e geração dos seus respectivos rastros de execução (passos 1 e 2); (ii) extração e abstração dos elementos de cada cenário a partir dos rastros gerados (passos 3 a 5); (iii) detecção de cenários implícitos a partir dos cenários extraídos na etapa (ii) (passos 6 a 8); e (iv) visualização dos cenários na forma de digramas de seqüência da UML (passos 9 a 12). Esta última etapa é opcional, e visa tão somente facilitar a compreensão dos cenários utilizando um formato compatível com o utilizado por ferramentas de modelagem comerciais [13]. As próximas subseções descrevem as três primeiras etapas do processo em mais detalhes.

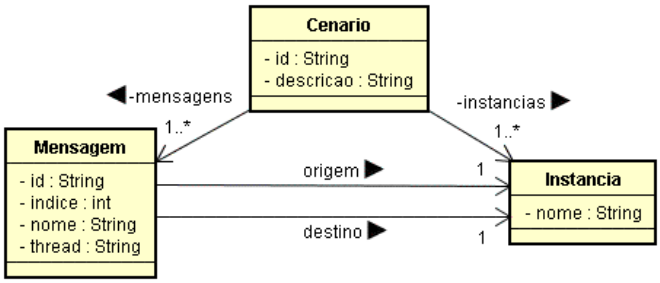

Figura 3. Metamodelo de rastros de execução.

3.1. Seleção dos Cenários e Geração de Rastros de Execução

O processo se inicia com a escolha de quais cenários ou conjuntos de funcionalidades da aplicação alvo serão executados para a geração dos rastros. Essa escolha normalmente é feita com a ajuda de um usuário ou desenvolvedor da aplicação, ou de algum especialista em seu domínio. Em seguida, a aplicação alvo é configurada para que sua execução seja monitorada por uma ferramenta denominada Monitor de Eventos. O Monitor de Eventos ficará responsável por monitorar cada evento ou ação (criação de objetos, chamada de métodos, etc.) executada pela aplicação. As ações classificadas como parte de um ou mais dos cenários selecionados serão incluídas no(s) rastro(s) de execução correspondente(s) a esse(s) cenário(s).

Os elementos que compõem um rastro de execução foram definidos a partir de um metamodelo de rastros de execução (ver Figura 3). Esse metamodelo foi criado para padronizar o registro e a manipulação das informações contidas em cada rastro, visando facilitar sua abstração na forma de cenários. O metamodelo definido neste trabalho tomou como base o metamodelo proposto no trabalho de Briand *et al*. para a recuperação de diagramas de seqüência da UML [3].

O elemento principal do metamodelo é a classe Cenario, que representa unicamente um cenário da aplicação monitorado pelo Monitor de Eventos (ex.: Incluir aluno, Realizar compras, etc). Essa classe possui um identificador único (id) para as suas instâncias como também uma sumária descrição (descricao). Cada cenário monitorado possui associado a ele um conjunto de instâncias (Instancia) e um conjunto de mensagens (Mensagem) atribuídas a essas instâncias, sendo ambos associados ao elemento cenário por meio de uma coleção ordenada de itens. A classe Instancia possui como único atributo um identificador (nome) único dentro do cenário ao qual está associada. Além de um identificador (id) único dentro do cenário, a classe Mensagem possui como atributos um índice (indice) que representa a sua ordem de execução no cenário, um nome (nome) que representa o nome do método invocado entre as instâncias, a identificação da linha de execução (thread) que originou a mensagem, e as instâncias de origem (origem) e destino (destino) da mensagem.

Ao monitorar um cenário específico, o Monitor de Eventos registra os elementos que compõem seu rastro de execução em estruturas de dados baseadas no meta-modelo descrito acima. Essas estruturas são mantidas em memória e podem facilmente ser traduzidas em documentos XML compatíveis com o metamodelo utilizando ferramentas apropriadas [24].

Tabela 1. Elementos filtrados do rastro de execução em cada nível de abstração

Nível de Abstração	Elementos Filtrados
1	Entidades externas ao(s) pacote(s) utilizado(s) pela aplicação.
2	Classes periféricas ao processo de negócio do cenário e classes utilitárias.
3	Chamadas a métodos construtores de classe.
4	Chamadas a métodos de uma mesma classe.
5	Padrões de mensagens repetidos.

Outra grande vantagem de se utilizar um metamodelo para a geração de rastros de execução é que ele permite a coleta e manipulação dos elementos dos rastros de forma independente da ferramenta utilizada como Monitor de Eventos e da linguagem de programação da aplicação alvo. Isto porque a estrutura do metamodelo representa genericamente quaisquer interações entre componentes de código em sistemas orientados a objetos. Assim, ele serve como ponte para a extração de cenários e descoberta de cenários implícitos em sistemas escritos em diferentes linguagens, bastando, para isso, que sejam utilizadas ferramentas de monitoração de eventos apropriadas para cada linguagem.

3.2. Extração e Abstração dos Cenários

A abstração dos elementos que compõem os rastros de execução da aplicação na forma de cenários é um processo subjetivo e complexo, e que varia conforme o cenário que está sendo extraído. Na realidade, não existe uma forma única para conseguir êxito nesse tipo de operação, exatamente porque cada cenário possui características próprias. Apenas um estudo minucioso de cada cenário, suas características e o ambiente arquitetural envolvido poderá indicar que passos serão utilizados no processo de abstração e em que ordem. Tal estudo deve ter como base um conhecimento profundo do processo de negócio representado no cenário bem como o seu contexto na aplicação.

O processo de abstração de cenários proposto consiste, basicamente, na aplicação de filtros e modificadores de conteúdo aos elementos originalmente capturados nos rastros. Operacionalmente, os cenários têm seus fluxos de mensagens modificados interativamente pela ferramenta de extração. Cada saída da ferramenta corresponde a uma nova versão do cenário utilizado como entrada e, caso o nível de abstração desejado ainda não tenha sido alcançado, este poderá ser entrada para um novo conjunto de filtros e modificadores de conteúdo. Esse refinamento sucessivo e incremental permite que o especialista de negócio defina o nível de abstração mais adequado para representar o cenário, podendo voltar a submetê-lo à ferramenta ou finalizar o processo quando estiver satisfeito com o resultado.

A atual versão da ferramenta de extração de cenários inclui cinco tipos de filtros. Esses filtros, quando aplicados aos elementos capturados nos rastros de execução, produzem cenários representados em cinco diferentes níveis de abstração. A Tabela 1 descreve quais elementos dos rastros são filtrados em cada um desses cinco níveis.

O primeiro nível de abstração tem o objetivo de melhor isolar o que se quer visualizar do sistema. Ele delimita a primeira fronteira do que vai ser extraído e garante que o universo de mensagens coletado deverá pertencer exclusivamente ao(s) pacote(s)

utilizado(s) na aplicação alvo, excluindo, assim, mensagens que envolvam classes externas (ex.: classes de pacotes distribuídos na plataforma Java). O segundo nível de abstração isola o processo de negócio, preservando somente os componentes de software diretamente envolvidos na implementação do(s) cenário(s) de interesse. Também nesse nível são removidas todas as classes consideradas utilitárias, ou seja, classes que oferecem funcionalidades ou serviços comuns a diversas outras classes da aplicação, e que, portanto, pouco contribuem para o entendimento dos objetivos específicos do cenário. O mecanismo de identificação de utilitários implementado é baseado na análise do grau de entrada e de saída de cada vértice (instância) que compõe o fluxo de mensagens capturado no rastro de execução, utilizando uma estratégia similar à abordagem proposta por Hamou-Lhadj et al. [6]. O terceiro e o quarto níveis de abstração eliminam informações consideradas irrelevantes do ponto de vista da interação entre os componentes, no caso, chamadas a métodos construtores de classe e chamadas a métodos internos (definidos na própria classe), respectivamente. Por fim, o quinto nível de abstração elimina padrões de mensagens que se repetem entre os componentes do cenário. A recorrência desses padrões geralmente está associada à utilização de instruções de controle de repetição pela aplicação, de modo que sua representação de forma repetida pouco acrescenta ao melhorar o entendimento do objetivo principal do cenário.

É importante enfatizar que o processo não impõe nenhuma estratégia rígida de extração de cenários, e tanto os níveis de abstração quanto os tipos de filtros definidos para cada nível podem ser alterados de acordo com as necessidades dos desenvolvedores e as características da aplicação alvo.

3.3. Detecção de Cenários Implícitos

Nesta etapa, os cenários extraídos na etapa anterior são exportados para um formato compatível com os arquivos de entrada esperados pela ferramenta de detecção de cenários implícitos, LTSA-MSC. Especificamente, cada cenário recuperado é editado como um cenário básico (bMSC). Em seguida, um cenário de alto nível (hMSC) é criado descrevendo os relacionamentos entre os diversos cenários básicos. Conforme descrito anteriormente, esse conjunto de cenários constitui o insumo básico para a identificação dos cenários implícitos pela ferramenta LTSA.

O processo de detecção de cenários implícitos também ocorre de forma incremental, com o analista de negócios modificando ou refinando sucessivamente o diagrama hMSC, de acordo com os resultados produzidos pela LTSA. No caso em que nenhum cenário implícito é encontrado, o analista pode tentar modificar o diagrama hMSC com a remoção de alguns cenários e/ou inclusão de novos cenários. Em último caso, o analista pode decidir por retornar à primeira etapa do processo, e extrair novos cenários da aplicação alvo, ou então descartar o conjunto de cenários investigados como estando livre de cenários implícitos.

3.4. Ambiente de Suporte

Como visto na Figura 2, o ambiente de detecção de cenários implícitos inclui um Monitor de Eventos, para gerar os rastros de execução da aplicação, um Extrator de Cenários, para abstrair os elementos dos cenários a partir das informações contidas nos rastros de execução, e a ferramenta LTSA, para a detecção dos cenários implícitos. Uma vez que a LTSA pôde ser reutilizada integralmente, o esforço de implementação do ambiente ficou concentrado no Monitor de Eventos e no Extrator de Cenários. Essas duas ferramentas são descritas brevemente a seguir.

3.4.1. Monitor de Eventos

O Monitor de Eventos foi desenvolvido utilizando a plataforma Java 2, Standard Edition (J2SE versão 6.0) [18]. Como mecanismo de captura dinâmica dos eventos gerados pela aplicação alvo, a ferramenta utiliza o arcabouço JPDA (*Java Platform Debugger Architecture*) [17], que provê uma infra-estrutura genérica e distribuída para o desenvolvimento de depuradores para a linguagem Java. Dessa forma, nenhuma mudança é necessária no código fonte ou compilado da aplicação alvo, bastando que a mesma seja executada sob controle da plataforma JPDA.

3.4.2. Extrator de Cenários

O extrator de cenários também foi desenvolvido na plataforma Java. Ele utiliza tecnologias existentes de geração e manipulação de documentos XML, no caso, XSLT [24], para aplicar os filtros de abstração aos elementos do rastro de execução, bem como para converter os cenários extraídos nos formatos compatíveis com a ferramenta LTSA e ferramentas de modelagem comercias. A conversão para ambos os formatos é feita apoiada num catálogo de transformações XSLT, criadas especificamente para este fim. A definição dos filtros empregados em cada nível é feita em um arquivo de configuração que serve de insumo para o Monitor de Eventos, utilizando uma sintaxe baseada em XML com elementos marcadores específicos para cada tipo de filtro. O detalhamento deste arquivo está fora do escopo deste artigo.

4. Exemplo

Em um esforço inicial de validação, o ambiente proposto foi utilizado para detectar cenários implícitos em uma aplicação existente de comércio eletrônico. Aplicações de comércio eletrônico são caracterizadas por cenários que apresentam um alto grau de concorrência, particularmente em operações que acessam seus componentes de negócios. Isso torna os cenários extraídos a partir da execução dessas aplicações potenciais candidatos para a detecção de cenários implícitos.

A aplicação alvo escolhida foi a MyPetStore, que é uma versão simplificada, desenvolvida pelos autores, da aplicação de comércio eletrônico PetStore [16]. Como o nome indica, PetStore é uma aplicação para comercialização de animais em uma hipotética loja virtual de animas de estimação. Ela é uma aplicação de código aberto, desenvolvida e mantida pela SUN como exemplo de trabalho para ilustrar os principais recursos de programação (interfaces dinâmicas, persistência, segurança, transações, etc.) disponíveis na plataforma J2EE. No caso da MyPetStore, foram implementadas apenas algumas funcionalidades básicas referentes à comercialização de produtos, além de operações para a manutenção de usuários e dos produtos comercializados, compreendendo um total de 21 classes. A Tabela 2 apresenta a relação completa das funcionalidades e operações implementadas na MyPetStore.

A razão da opção pela MyPetStore, ao invés da PetStore original, é de ordem estritamente prática, já que com a primeira teríamos total controle sobre como cada cenário de uso da aplicação estaria implementado em nível de código. Como mencionado anteriormente, esse nível de conhecimento é fundamental para guiar o processo de extração e abstração de cenários a partir dos rastros de execução gerados. Uma vez que nosso principal interesse era validar o processo proposto, ao invés de precisamente re-documentar uma aplicação existente, a utilização da MyPetStore como aplicação alvo do nosso estudo provou ser a solução de melhor custo-benefício, antes de avaliarmos nossas

Tabela 2. Funcionalidades implementadas na aplicação MyPetStore

Funcionalidade	**Operações Implementadas**
Entrar e sair do Sistema	*Login, Logout*
Manter usuários	*Find, Insert, Update, Delete*
Manter produtos	*Find, Insert, Update, Delete*
Realizar compra	*ClearShopCart, ListShopCart, CloseShopCart, InsertItemInShopCart*

ferramentas em aplicações concorrentes de maior porte. Além disso, o fato da MyPetStore também ter sido desenvolvida seguindo as melhores práticas e padrões de projetos recomendados pela SUN para a plataforma J2EE reforça a nossa confiança de que os resultados obtidos nesse estudo também poderão ser reproduzidos em outras aplicações concorrentes com características similares.

4.1 Seleção de Cenários e Geração de Rastros de Execução

Devido à natureza concorrente da aplicação alvo, algumas de suas operações podem ser executadas no contexto de mais de um cenário. Como exemplo, temos o caso da operação de encerramento do carrinho (*CloseShopCart*), que além de desencadear o cenário de preparação para a compra, ainda dispara os de validação do usuário para a compra, o de verificação de itens em estoque e o de efetivação da compra propriamente dita. Esse nível de associação ou dependência entre os cenários ilustra bem a dificuldade de se definir exatamente onde começa e onde termina cada cenário, e realça ainda mais a necessidade da presença de especialistas da aplicação durante o processo de extração. A Tabela 3 descreve os cenários escolhidos para o exemplo, bem como as operações da aplicação que foram monitoradas para gerar os seus respectivos rastros de execução.

Tabela 3. Cenários selecionados para o exemplo

Cenário	**Operação Monitorada**
Login	*Login*
Logout	*Logout*
Preparar compra	*CloseShopCart*
Autenticar usuário	
Comprar	
Excluir usuário	*Delete (User)*

4.2. Extração e Abstração dos Cenários

O processo de abstração dos cenários consistiu no processamento das informações contidas nos rastros de execução da aplicação, de acordo com os tipos de filtros definidos para cada nível. No primeiro nível, foram descartados todos os elementos que não pertenciam ao pacote principal da aplicação. No segundo nível, foram eliminados elementos oriundos de classes identificadas como utilitárias (incluindo classes da camada de apresentação). Esse nível conseguiu isolar com precisão o subconjunto de chamadas (mensagens) realizadas entre componentes de negócio da aplicação, obtendo uma redução

significativa (sempre superior a 85%) do conjunto de elementos inicialmente capturado pelo Monitor de Eventos. Em alguns cenários que fazem uso freqüente de componentes visuais e utilitários, como é o caso do cenário de *Exclusão de Usuários*, essa redução foi superior a 97%. A Tabela 4 apresenta a evolução dos números de elementos presentes nesse cenário ao longo dos cinco níveis de abstração. Note que no quinto nível, esse cenário apresentou uma redução de 50% no número de mensagens em relação ao nível anterior. A razão é que o cenário inicialmente refletia a exclusão de dois usuários, cujo padrão de mensagens era exatamente o mesmo nos dois casos.

Tabela 4. Evolução do número de elementos do cenário de exclusão de usuário durante o processo de abstração

Nível de Abstração	Quantidade de Elementos Coletados no Cenário		Percentual de Redução em Relação à Coleta Inicial	
	# Inst.	# Mens.	# Inst.	# Mens.
1	19	455	–	–
2	4	10	78,95%	97,80%
3	4	10	78,95%	97,80%
4	3	6	84,21%	98,68%
5	3	3	84,21%	99,34%

A Figura 4 mostra os diagramas de seqüência representando todos os seis cenários extraídos da aplicação MyPetStore após o processo de abstração dos rastros de execução.

4.3. Detecção de Cenários Implícitos

Cada um dos seis cenários extraídos na etapa anterior foi tratado como um cenário básico (bMSC) a ser alimentado como insumo da ferramenta LTSA. Assim, para viabilizar o processo de detecção de cenários implícitos pelo LTSA, foi necessário ainda definir novos cenários de alto nível (hMSC's) estabelecendo as possíveis relações entre dois ou mais cenários básicos. Dentre diferentes configurações de cenários de alto nível investigadas, três resultaram em cenários implícitos. Devido a restrições de espaço, aqui descreveremos apenas duas.

Cenário Implícito 1: Usuário sai do sistema durante compra

Para a detecção desse cenário implícito, foi definido um cenário de alto nível envolvendo cinco dos seis cenários básicos extraídos da aplicação: *Login*, *Logout*, *Preparar Compra*, *Autenticar Usuário* e *Comprar* (ver Figura 5). Esses cenários foram selecionados porque compartilham um ou mais componentes de negócio, como é o caso dos componentes *UserControllerImpl* (utilizado em *Login*, *Logout* e *Autenticar Usuário*) e *ShopControllerImpl* (utilizado em *Preparar Compra*, *Autenticar Usuário* e *Comprar*).

O cenário de alto nível inclui um nó inicial (*init*), cuja única transição leva ao cenário *Login*. A partir de *Login* há também uma única transição, para o cenário *Preparar Compra*. A partir desse cenário há três transições possíveis: para *Logout*, o que forçaria o sistema a retornar ao estado inicial; para *Autenticar Usuário*, dando continuidade ao processo de compra de itens; e, por último, de volta a ele mesmo, indicando que uma outra compra poderia ser preparada em substituição à atual. No cenário *Autenticar Usuário* há somente uma transição possível, para o cenário *Comprar*, e deste para o

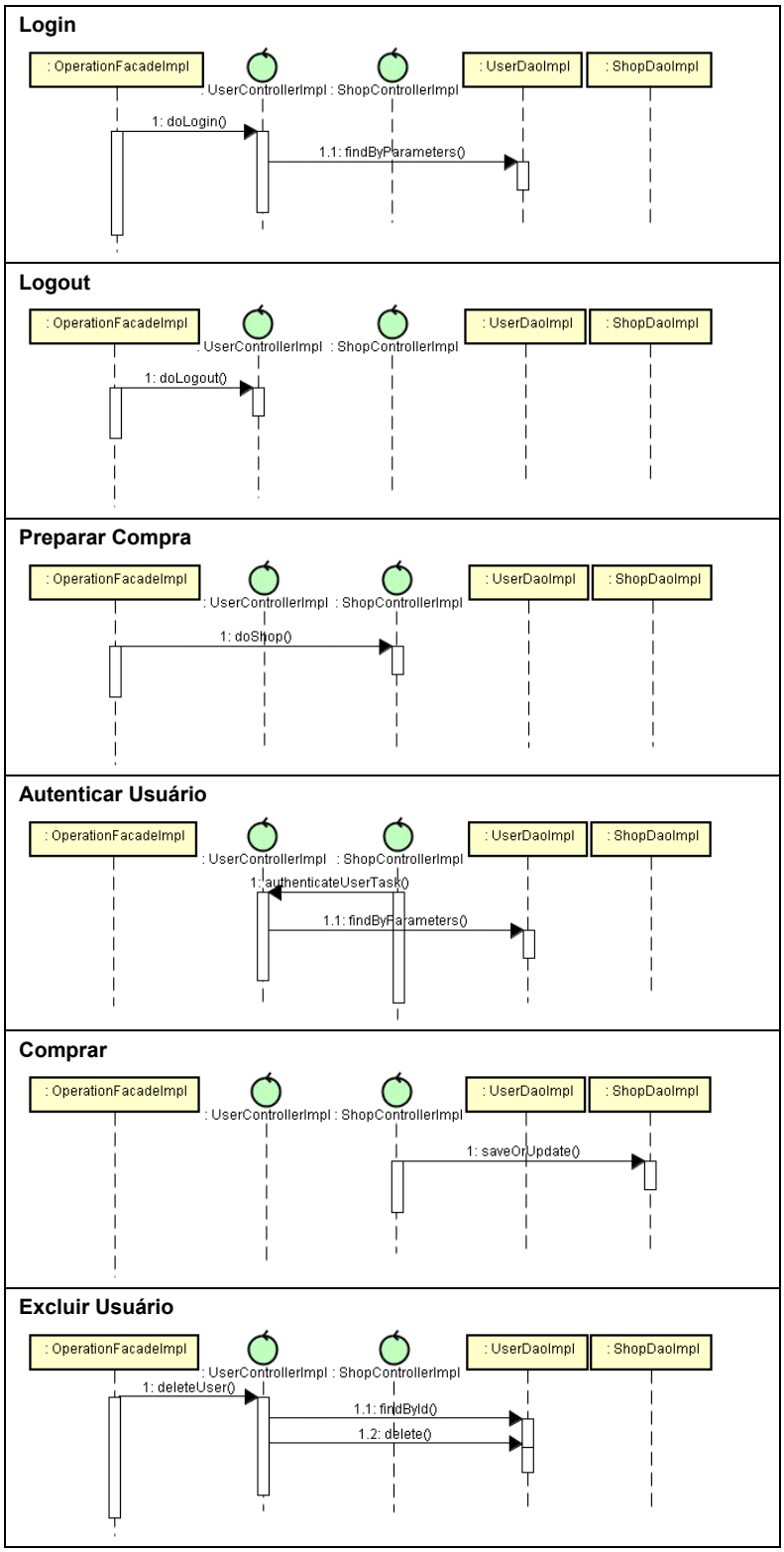

Figura 4. Cenários extraídos da aplicação MyPetStore.

cenário *Preparar Compra*, iniciando um novo ciclo de compras. Esse ciclo (*Preparar Compra* → *Autenticar Usuário* → *Comprar* → *Preparar Compra*) faz parte da operação de encerramento de carrinho, na qual os três cenários devem sempre acontecer nessa ordem.

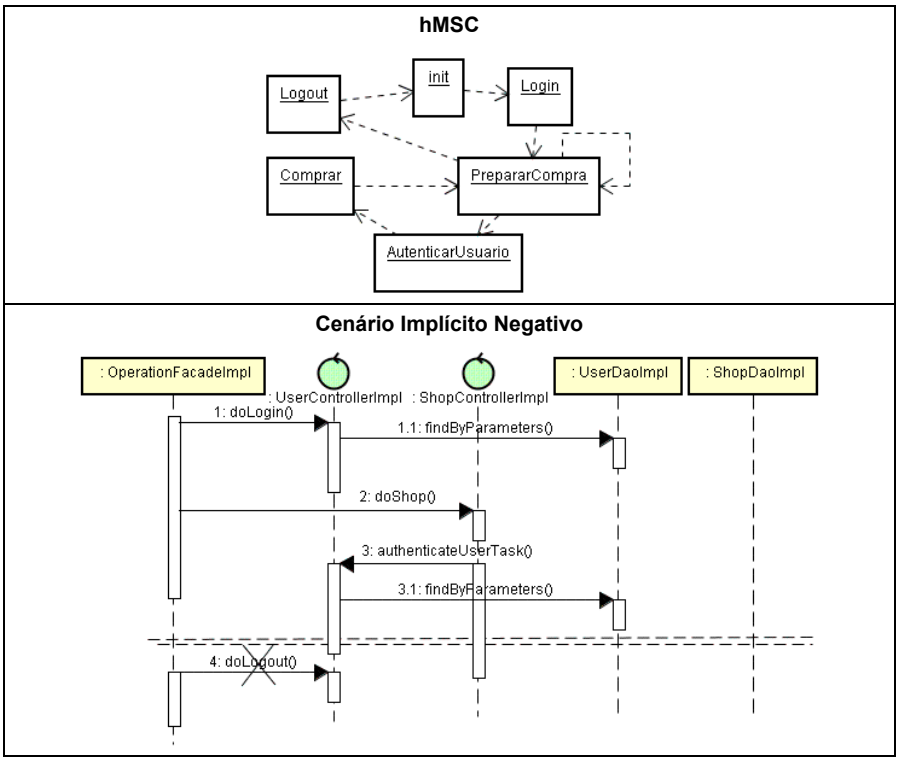

Figura 5. Cenário de alto nível e diagrama de seqüência do primeiro cenário implícito.

Tendo como entrada os cinco cenários básicos e o cenário de alto nível definido sobre eles, a ferramenta LTSA identificou o cenário implícito cujo diagrama de seqüência é mostrado na parte inferior da Figura 5. O contexto de execução desse cenário implícito é o seguinte:

i. Entrada (*login*) do usuário na aplicação (mensagens 1 e 1.1);

ii. Preparação para a compra (mensagem 2);

iii. Autenticação do usuário para a compra (mensagens 3 e 3.1);

iv. Saída (*logout*) do usuário da aplicação (mensagem 4).

O problema identificado neste cenário reside exatamente na troca da mensagem 4 entre os componentes OperationFacadeImpl e UserControllerImpl. Onde era esperada a efetivação da compra, através da execução do cenário Comprar, o cenário implícito mostra que existe a possibilidade de execução do cenário Logout. Isso significa que, caso o cenário implícito venha a acontecer, a aplicação poderia interromper um processo de compra de forma abrupta, contribuindo para o risco de perda de integridade nas informações que mantém. Outro problema, de impacto ainda maior que o anterior, seria a continuação do processo de compra após a operação de logout. Esse problema poderia ocorrer em algumas arquiteturas concorrentes, nas quais os componentes de cada cenário são executados em threads independentes. Nesse caso, a aplicação permitira a compra de itens mesmo após o usuário ter saído do sistema, representando uma falha de segurança.

Os problemas representados neste cenário ilustram bem os benefícios que a detecção de cenários implícitos pode trazer para a atividade de compreensão de um sistema existente. Muito embora cada cenário executado tenha apresentado um comportamento "correto" do seu ponto de vista local, diferentes combinações desses cenários podem levar a situações indesejadas com respeito ao comportamento global da

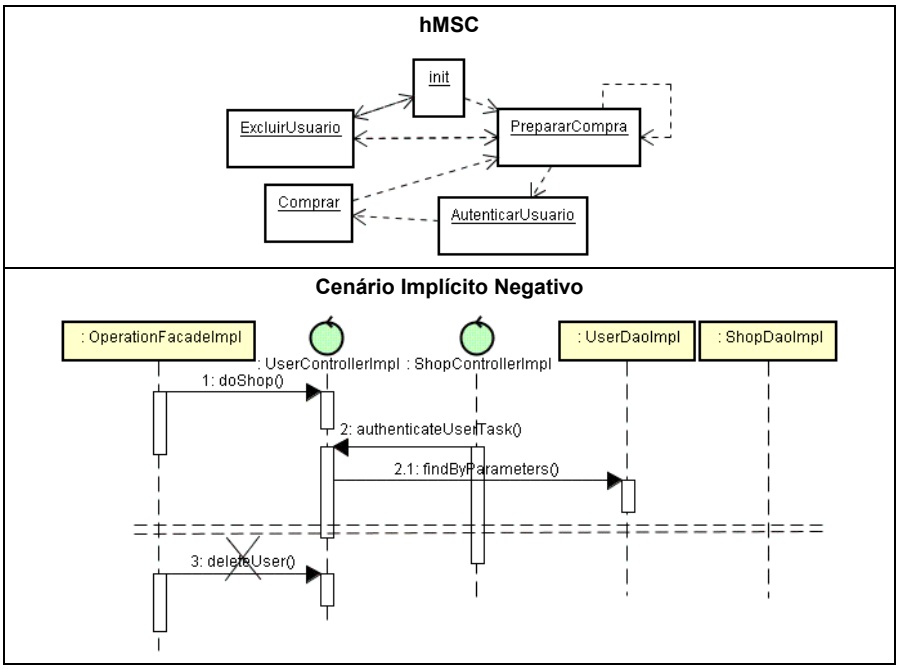

Figura 6. Cenário de alto nível e diagrama de seqüência do segundo cenário implícito.

aplicação, como aquelas discutidas acima. Portanto, a vantagem que a detecção de cenários implícitos oferece aos desenvolvedores é justamente poder ajudá-los a identificar e se planejar para essas situações problemáticas, antes que elas venham a acontecer na prática. Por exemplo, para evitar que os problemas identificados neste primeiro cenário implícito aconteçam, uma possível solução seria a aplicação realizar um teste no momento que a operação de logout fosse executada. Esse teste verificaria se existe alguma transação de qualquer espécie ainda em aberto na aplicação, e que tenha sido iniciada pelo usuário atual. Caso alguma transação ainda não tenha sido completada, a operação de logout poderia mascarar o problema retardando a sua execução até um estágio em que não houvesse mais nenhuma transação em aberto, ou poderia apenas tolerar o erro, informando ao usuário da impossibilidade da saída da aplicação naquele momento.

Cenário Implícito 2: Usuário é removido do sistema durante compra

Para a descoberta deste segundo cenário implícito foi definido um cenário de alto nível contendo os seguintes cenários básicos: *Preparar Compra*, *Autenticar Usuário*, *Comprar* e *Excluir Usuário* (ver Figura 6). Novamente, esses cenários foram selecionados porque possuem componentes de negócio em comum. Nesse caso, os cenários considerados também compartilham os mesmos componentes compartilhados no caso anterior, ou seja, *ShopControllerImpl*, utilizado nos cenários *Preparar Compra*, *Autenticar Usuário* e *Comprar*, e *UserControllerImpl*, utilizado nos cenários *Autenticar Usuário* e *Excluir Usuário*.

O cenário de alto nível da Figura 6 mostra novamente um ciclo de transições entre cenários (*Preparar Compra* → *Autenticar Usuário* → *Comprar* → *Preparar Compra*), o qual também compõe a operação de encerramento de carrinho. Porém, dessa vez existe a possibilidade de execução do cenário de exclusão de usuário a partir do cenário de preparação de compra. O cenário de alto nível também estabelece transições de saída do cenário de exclusão de usuário, que tanto pode retornar para o cenário de preparação de

compra quanto para o cenário inicial.

Utilizando novamente os cenários básicos e o cenário de alto nível como insumos, a ferramenta LTSA identificou o cenário implícito cujo diagrama de seqüência é mostrado na parte inferior da Figura 6. O contexto de execução desse segundo cenário implícito é o seguinte:

i. Preparação para a compra (mensagem 1);
ii. Autenticação de usuário para a compra (mensagens 2 e 2.1);
iii. Exclusão de usuário (mensagem 3).

A execução do segundo e terceiro passos demonstram uma possível situação de erro. Isto porque essa ordem de execução irá possibilitar a um operador da aplicação a exclusão de um usuário durante o processo de compra de um item. Embora à primeira vista essas duas operações não aparentem estarem relacionadas, a sua execução, nessa ordem específica, pode levar à perda da integridade nos dados da aplicação, bastando, para isso, que o usuário excluído seja o mesmo autenticado durante o processo de compra. Levando-se em conta o paralelismo entre as duas operações, a continuidade da execução desses dois cenários resultará em um pedido de compra autenticado para um usuário que não mais existe na aplicação.

A atual versão da MyPetStore garante a integridade dos dados em operações de exclusão verificando as dependências dos itens a serem excluídos antes da efetivação da operação. Um exemplo seria antes de excluir um usuário verificar se algum pedido de compra já havia sido feito por seu intermédio. Em caso positivo, esse usuário não poderia ser excluído. Entretanto, no caso do cenário implícito, existe a possibilidade desse usuário estar sendo referenciado pela primeira vez na aplicação, não havendo ainda nenhuma ligação sua com algum pedido previamente realizado. Há duas visões para o tratamento desse tipo de problema. Como as operações de exclusão de usuários e de compra de itens não dependem necessariamente uma da outra, as soluções estão diretamente associadas à ótica de cada cenário. No primeiro caso, a exclusão de um usuário somente poderá ser concretizada quando nenhum pedido de compra estivesse em andamento referenciando o usuário que se deseja excluir. No segundo caso, a compra somente poderá ser efetivada se as dependências do pedido sejam satisfeitas, ou seja, o usuário que fez o pedido deverá necessariamente existir no sistema.

5. Discussão

Além do acréscimo notório à documentação de uma aplicação existente, a identificação de cenários implícitos pode indicar possíveis comportamentos inesperados aos responsáveis pela aplicação. Mesmo no caso em que a aplicação já tenha alguma solução implementada para contornar os problemas representados no cenário implícito, detectá-lo explicitamente ainda seria útil como forma de atrair a atenção para aspectos importantes da aplicação que, de outro modo (ou seja, com base apenas nos cenários executados), poderiam passar despercebidos a seus desenvolvedores.

A descoberta de cenários implícitos também pode ser útil no suporte a atividades de testes. Por exemplo, dado um conjunto de rastros de execução que foram gerados a partir de um conjunto de casos de teste, um cenário implícito seria uma indicação de um comportamento possível do sistema, mas ainda não coberto pelos casos de teste ou não produzido pela execução deles (por causa de um fluxo alternativo ocorrido devido a um comportamento não determinístico do sistema concorrente). Em essência, o cenário implícito indicará um comportamento do sistema ainda não explorado. Este mecanismo

pode ser particularmente útil em um processo de desenvolvimento dirigido por testes [2], onde cenários implícitos apontariam aspectos do sistema onde haveria uma maior necessidade de testes.

Por fim, deve-se destacar que a quantidade de cenários e de mensagens presentes nos cenários extraídos dos rastros de execução mostrou-se um fator limitante para o desempenho da ferramenta de detecção de cenários implícitos. Isto porque o LTSA foi desenvolvido inicialmente visando cenários tipicamente utilizados na fase de especificação e elaboração de requisitos, cujo tamanho e complexidade tendem a ser bastante reduzidos. Assim, um processo de abstração de cenários, para ser bem sucedido, deve sempre poder garantir uma redução muito significativa do total de elementos presentes nos rastros de execução coletados inicialmente em cada cenário (por exemplo, nos cenários extraídos no exemplo o fator de redução foi superior a 87%). Da mesma forma, a presença de um especialista da aplicação durante o processo de abstração dos cenários é imprescindível para não descaracterizar os cenários extraídos e garantir que estes de fato reflitam o comportamento da aplicação alvo.

6. Trabalhos Relacionados

Muitas ferramentas comerciais de engenharia reversa (por exemplo, Jinsight [8]) empregam a análise dinâmica do comportamento de programas em execução a fim de recuperar cenários representados como diagramas de seqüência ou MSCs. No âmbito acadêmico, três trabalhos merecem destaque. Briand *et al.* [3] propõem uma metodologia, também baseada na análise dinâmica, para a engenharia reversa de diagramas de seqüência da UML. A proposta utiliza a programação orientada a aspectos como mecanismo de instrumentação do código da aplicação, e define metamodelos para mapear as entidades e os relacionamentos tipicamente capturados em rastros de execução em elementos correspondentes dos diagramas de seqüência. Hamou-Lhadj *et al.* [4] propõem um ambiente para a recuperação de modelos de comportamento a partir de trilhas rastros de execução. Uma característica diferencial do ambiente é a remoção de componentes utilitários dos modelos recuperados pela técnica conhecida como análise *fan-in,* que é baseada na exploração do grafo de dependência entre as classes do sistema, obtido através da analisa estática do código, utilizada para identificar as classes mais referenciadas. Por fim, Richner e Ducassse [14] apresentam uma ferramenta para a recuperação de informações de colaboração genéricas (que podem ser aplicadas a qualquer linguagem de programação orientada a objeto) e posterior representação dessas informações em termos de padrões pré-estabelecidos. Outros trabalhos de engenharia reversa empregam a análise estática para a recuperação de diagramas de seqüência a partir de artefatos de código. Exemplos incluem os trabalhos de Rountev *et al.* [15], que propõem uma maneira de mapear grafos de controle de fluxo às recém lançadas primitivas de controle da nova geração da UML 2.0; e de Mansurov e Campara [11], que apresenta uma maneira de extrair cenários representados na forma de MSCs. Finalmente, alguns trabalhos recentes de pesquisa (ex.: [6][7]) endereçam especificamente o problema da diminuição da complexidade dos rastros de execução capturados através da análise dinâmica.

Todos os trabalhos acima focam exclusivamente na tecnologia necessária para extrair diagramas de seqüência (ou similares), que constituem a base para a recuperação de cenários, a partir da análise estática ou dinâmica de aplicações existentes. Trabalhos pioneiros na detecção automática de cenários implícitos foram propostos por Uchitel *et al.* ([19]), com o objetivo inicial de facilitar o processo de especificação de requisitos para sistemas concorrentes. Conforme mencionado ao longo do artigo, nosso trabalho se

beneficia de resultados oriundos dessas duas áreas de pesquisa. Nesse sentido, a principal contribuição do trabalho está na utilização de técnicas e ferramentas modernas de extração de cenários na forma de MSCs para permitir a detecção de cenários implícitos a partir de aplicações existentes, oferecendo, assim, mais um importante instrumento de apoio às atividades de compreensão, teste e manutenção de sistemas.

7. Conclusões e Trabalhos Futuros

Este trabalho apresentou um processo e um ambiente automatizado para apoiar a descoberta de cenários implícitos a partir de informações coletadas dinamicamente durante a execução de uma aplicação concorrente existente. Essa abordagem, além de enriquecer as especificações de cenários originais, propicia subsídios que podem ser de grande ajuda na melhoria do processo de compreensão da aplicação alvo, trazendo uma maior segurança às atividades de teste, manutenção e evolução. Em particular, a abordagem possibilita que comportamentos inusitados, não previstos nos cenários executados originalmente, possam ser facilmente identificados, rastreados e tratados, contribuindo para a melhoria da qualidade do processo de desenvolvimento.

Como sugestão para trabalhos futuros, temos a realização de novos experimentos de avaliação, preferencialmente com aplicações concorrentes de código aberto de tamanhos e domínios variados. Outra sugestão seria a utilização de outras ferramentas de monitoração para extrair rastros de execução de aplicações escritas em diferentes linguagens de programação. A idéia é poder verificar na prática a característica de independência de linguagem do metamodelo de rastros utilizado no ambiente. Por fim, sugerimos, também, uma investigação de novas técnicas de abstração de cenários, particularmente de filtros que possam reduzir ainda mais o tamanho dos rastros de execução gerados pelo monitor de eventos. Essa linha de pesquisa é fundamental para viabilizar a utilização do ambiente proposto em aplicações "reais".

Referências

[1] Alur, R., Etessami, K., Yannakakis, M. (2000), Inference of Message Sequence Charts. In Proc. 22nd Int. Conf. on Software Engineering. (ICSE'00), Limerick, Ireland.

[2] Beck, K. (2003), Test-Driven Development by Example, Addison-Wesley.

[3] Briand, L.C., Labiche, Y., Miao, Y. (2003), Towards the Reverse Engineering of UML Sequence Diagrams. In Proc. 10th Work. Conf. on Reverse Engineering (WCRE'03), Victoria, BC, Canada.

[4] Hamou-Lhadj, A., Braun, E., Amyot, D., Lethbridge, T. (2005), Recovering Behavioral Design Models from Execution Traces. In Proc. 9th Eur. Conf. on Software Maintenance and Reengineering (CSMR'05), Manchester, U.K.

[5] Hamou-Lhadj, A., Lethbridge, T. C. (2004), A Survey of Trace Exploration Tools and Techniques. In Proc. 2004 Conference of the Centre for Advanced Studies and Collaborative Research (CASCON'04), Markham, Ontario, Canada, pp. 42-55.

[6] Hamou-Lhadj, A., Lethbridge, T. C. (2005), Reasoning about the Concept of Utilities. In Proc. 1st ECOOP Workshop on Practical Problems of Programming in the Large (PPPL'04), Oslo, Norway, LNCS Vol. 3344, Springer.

[7] Hamou-Lhadj, A., Lethbridge, T. C. (2003), Techniques for Reducing the Complexity of Object-Oriented Execution Traces. In Proc. 2nd DESIGNFEST on Visualizing

Software for Understanding and Analysis (VISSOFT'03), Amsterdam, The Netherlands.

[8] IBM Research, Jinsight: Visualizing the Execution of Java Programs. Disponível em http://www.research.ibm.com/jinsight.

[9] ITU-T Recommendation Z.120 (2004), Message Sequence Chart (MSC).

[10] Magee, J. and Kramer, J. (1999), Concurrency: State Models and Java Programs, John Wiley & Sons Ltd., New York.

[11] Mansurov, N., Campara, D. (2001), Using Message Sequence Charts to Accelerate Maintenance of Existing Systems. In Proc. 10th Int. SDL Forum (SDL'01: Meeting UML), LNCS Vol. 2078, Springer.

[12] OMG, Unified Modeling Language Specification V2.1.1. Disponível em http://www.omg.org/ technology/documents/formal/uml.htm.

[13] OMG, XML Metadata Interchange Specification V2.1. Disponível em http://www.omg.org/technology/documents/formal/xmi.htm.

[14] Richner, T., Ducasse, S. (2002), Using Dynamic Information for the Iterative Recovery of Collaborations and Roles. In Proc. 18th Int. Conf. on Software Maintenance (ICSM'02), Montreal, Quebec, Canada.

[15] Rountev, A., Volgin, O., Reddoch, M. (2004), Control Flow Analysis for Reverse Engineering of Sequence Diagrams, Tech. Report OSU-CISRC-3/04-TR12, Depart. of Computer Science and Engineering, Ohio State University.

[16] SUN, Java Pet Store. Disponível em http://java.sun.com/developer/releases/petstore/.

[17] SUN, Java Platform Debugger Architecture. Disponível em http://java.sun.com/ javase/6/docs/technotes/guides/jpda/index.html.

[18] SUN, Java SE 6 Platform. Disponível em http://java.sun.com/javase/.

[19] Uchitel, S., Chatley, R., Kramer, J., Magee, J. (2003), LTSA-MSC: Tool Support for Behaviour Model Elaboration Using Implied Scenarios. In Proc. 9th Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'03), LNCS Vol. 2619, Springer.

[20] Uchitel, S., Kramer, J., Magee, J. (2001), Detecting Implied Scenarios in Message Sequence Chart Specifications. In Proc. Joint 8th Euro. Software Engineering Conference (ESEC'01) and 9th ACM SIGSOFT Symp. on the Found. of Software Engineering (FSE'01), Vienna, Austria, pp. 74-82.

[21] Uchitel, S., Kramer, J., Magee, J. (2004), Incremental Elaboration of Scenario-based Specifications and Behaviour Models Using Implied Scenarios, ACM Trans. on Software Engineering and Methodology, 13(1).

[22] Uchitel, S., Kramer, J., Magee, J. (2002), Negative Scenarios for Implied Scenario Elicitation. In Proc. 10th ACM SIGSOFT Symp. on the Found. of Software Engineering (FSE'02), Charleston, SC, USA.

[23] Uchitel, S., Kramer, J., Magee, J. (2003), Synthesis of Behavioral Models from Scenarios, IEEE Trans. on Software Engineering, 29(2), pp. 99-115.

[24] W3C, XSL Transformations. Disponível em http://www.w3.org/TR/xslt.