Um Mecanismo de Rastreamento da Evolução de Cenários Baseado em Transformações

Ulf Bergmann
Instituto Militar de Engenharia
ulf@ime.eb.br

Julio Cesar S. do Prado Leite
Pontifícia Universidade
Católica do Rio de Janeiro
www.inf.puc-rio.br/~julio

Karin Koogan Breitman
Pontifícia Universidade
Católica do Rio de Janeiro
karin@inf.puc-rio.br

Resumo: Este artigo tem como objetivo apresentar a utilização da tecnologia transformacional na implementação de um mecanismo de rastreamento da evolução de cenários. Com isso obtemos a atualização automática das informações de rastreamento, através da identificação das mudanças ocorridas entre versões consecutivas dos cenários produzidos e o seu subseqüente armazenamento na forma de transformações. Os benefícios obtidos pelo uso deste mecanismo permitirão: a eliminação dos problemas da falta de atualização das informações de rastreamento e o conseqüente aumento do valor destas informações para o desenvolvedor; a sistematização do processo de rastreamento; e a diminuição do custo para se obter, atualizar e validar as informações de rastreamento. Além destes benefícios, o armazenamento das informações de rastreamento na forma de transformações incorpora um maior conhecimento sobre a modelagem realizada, ao contrário de outros mecanismos que armazenam somente os artefatos e seus inter-relacionamentos estáticos.
Palavras chave: Evolução de Software, Rastreamento, Sistemas Transformacionais.

Abstract: This work aims to show how the transformational approach helps the implementation of a scenario traceability mechanism. Using transformations it is possible to automate the process of scenario evolution by storing and identifying the changes among different consecutive versions of scenarios. These evolution traces are kept as transformations, thus improving the traceability process by formalizing the changes that do occur. This approach differs from traditional ones which capture just the artifacts and their static relationships.
Keywords: Software Evolution, Traceability, Transformation Systems.

1 Introdução

Rastreabilidade de requisitos é a habilidade de descrever e acompanhar a vida de um requisito do sistema em ambas as direções, avante e reversa, isto é, desde suas origens, passando pelo desenvolvimento e especificação, até sua subseqüente distribuição e utilização, através de refinamentos e interações contínuas em qualquer destas fases [1].

Apesar da importância da rastreabilidade na evolução de sistemas ser um fato amplamente reconhecido [2] [3] [4] e dos esforços despendidos atualmente na geração e validação das informações sobre os relacionamentos e interdependências entre o mundo real, os requisitos e o sistema de software, as informações de rastreamento freqüentemente não são confiáveis, devido principalmente à inconsistência causada pela evolução distinta do sistema e dos seus requisitos.

A partir do nosso entendimento sobre a importância da utilização de um mecanismo de rastreamento dos artefatos produzidos durante o desenvolvimento de sistemas de software, bem como das dificuldades relacionadas à sua efetiva implantação, visualizamos que as soluções para a obtenção de uma maior confiabilidade e aderência destas informações passam, necessariamente, pelo aumento do nível de automação das tarefas relacionadas à

captura e manutenção das informações de rastreamento.

Este artigo apresenta uma abordagem para o rastreamento da evolução dos cenários de um sistema, que utiliza a tecnologia transformacional como forma de obter um maior nível de automatização das tarefas de captura e manutenção das informações de rastreamento, permitindo a diminuição dos problemas decorrentes da falta de atualização das informações, e o conseqüente aumento de seu valor para o desenvolvedor, bem como a diminuição do custo para se obter, atualizar e validar as informações de rastreamento.

A principal diferença de nossa abordagem em relação aos mecanismos tradicionais de rastreamento reside na forma como as informações sobre a evolução dos artefatos são armazenadas. Enquanto que os mecanismos tradicionais armazenam somente os artefatos e seus inter-relacionamentos estáticos, nossa abordagem armazena as informações na forma de transformações que devem ser aplicadas em uma versão de um artefato de maneira a se obter a nova versão do mesmo. Esta forma de armazenamento carrega um maior conhecimento sobre a modelagem executada incorporando um maior valor semântico às informações sobre a evolução do sistema em desenvolvimento.

No restante desta seção serão apresentadas a Máquina Draco-Puc, ferramenta básica utilizada na implementação do mecanismo, e algumas considerações sobre evolução de software relacionadas a este trabalho. A seção 2 apresenta conceitos relacionados a cenários e sua evolução. A seção 3 descreve o mecanismo propriamente dito e a seção 4 apresenta o estudo de caso utilizado para investigar sua aplicabilidade. Concluindo, a seção 5 apresenta os benefícios da utilização do mecanismo, a comparação com propostas existentes na literatura e sinaliza para alguns trabalhos futuros. Ainda nesta seção é apresentada uma análise da aplicação do mecanismo de rastreamento baseado em transformações em outros artefatos oriundos do processo de desenvolvimento.

1.1 Máquina Draco-Puc

O paradigma Draco [5][6] pode ser caracterizado como um conjunto de linguagens de domínios utilizadas no processo de construção de sistemas, através de um mecanismo transformacional capaz de gerar um programa executável a partir de uma especificação descrita em uma linguagem específica a um domínio (DSL – Domain Specific Language).

Diferentemente da maneira tradicional de obtenção de reuso, baseada em biblioteca de componentes, no paradigma Draco os elementos reutilizáveis, chamados de domínio, são definidos por linguagens formais. Estas linguagens são construídas com o objetivo de encapsular objetos e operações de um determinado domínio. Os programas escritos nas linguagens de domínio necessitam de um processo de refinamento para que possa ser obtido o programa executável que atende as especificações constantes do programa original.

A Máquina Draco-PUC é um sistema de software que implementa parcialmente o paradigma Draco. Sua utilização é feita através da construção de domínios descritos pela sua linguagem específica. Um domínio Draco-PUC precisa especificar sua sintaxe, descrita através da sua gramática, e sua semântica, expressa por transformações entre domínios. Uma transformação é composta basicamente pelo padrão de reconhecimento (LHS- left hand side) e pelo padrão de substituição (RHS – right hand side): quando é identificada uma ocorrência do padrão de reconhecimento no programa original, esta ocorrência é substituída pelo padrão de substituição.

Uma visão geral do uso da Máquina Draco-PUC é mostrada na figura 1. O parser do domínio do programa fonte é utilizado pela Máquina Draco-PUC para converter o programa na forma interna utilizada pela máquina (DAST – *Draco Abstract Syntax Tree*). Transformações podem ser aplicadas para modificar a DAST. Estas transformações podem ser intra-domínio, caso em que a DAST modificada pertence ao mesmo domínio original, ou

inter-domínio, quando a DAST modificada pertence a um novo domínio. A qualquer momento pode ser utilizado o *pretty-printer* correspondente para realizar o *unparse* de uma DAST e obter sua descrição na linguagem de domínio correspondente.

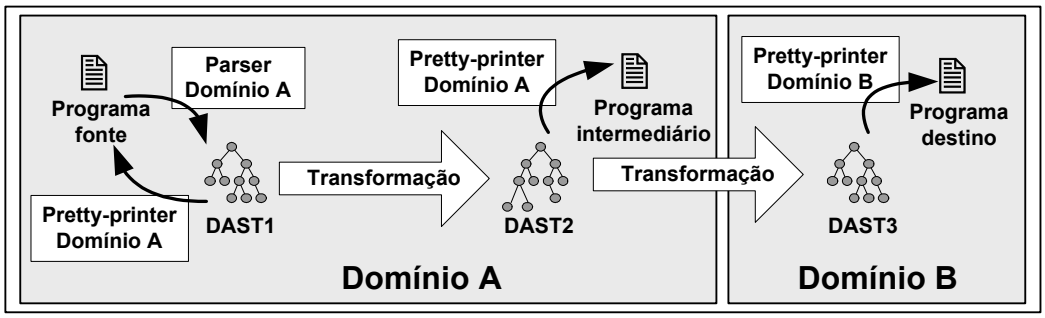

Figura 1 - Visão Geral do Uso da Máquina Draco-PUC

1.2 Evolução de Software

Consideramos o termo evolução de software no seu sentido mais amplo, englobando todo o ciclo de vida do software, desde sua concepção, passando pelo seu desenvolvimento, operação e manutenção, até a sua retirada de produção.

Desta maneira, podemos entender a evolução de um sistema de software como o resultado da aplicação, ao longo do tempo, de uma série de operações realizadas pelos desenvolvedores sobre os artefatos que compõe o sistema. Estas operações realizam um conjunto de modificações na configuração do sistema, ora inserindo ou removendo artefatos, ora modificando os artefatos existentes.

O mecanismo proposto neste artigo se propõe a rastrear a evolução dos cenários a partir da identificação das operações realizadas sobre os mesmos ao longo da vida do sistema. Em relação ao modelo de referência para rastreamento proposto por Ramesh [7], nosso trabalho busca identificar as ligações Evolui-Para (Evolves-To), que especifica que um objeto evolui para outro objeto através de alguma ação executada; e a Ligação Justifica (Rationale), que especifica a razão pela qual se deu determinada evolução de um objeto em outro.

Nossa abordagem para o rastreamento da evolução de um sistema é fundamentada na existência de uma formalização e categorização das operações possíveis de serem executadas sobre cada tipo de artefato. Utilizamos em nosso trabalho a taxonomia para a evolução de cenários definida por Breitman [8] e apresentada na seção 2.2. Esta taxonomia define um conjunto de operações sobre cenários e uma série de heurísticas que podem ser utilizadas na sua identificação. A partir da existência de uma taxonomia de evolução para outros tipos de artefatos, esta mesma abordagem pode ser utilizada no rastreamento de sua evolução. A seção 5.4 mostra as modificações e extensões necessárias para possibilitar a utilização da abordagem transformacional no rastreamento da evolução de outros tipos de artefatos.

2 Cenários

Cenários são considerados descrições evolutivas de situações num ambiente [9], sendo compostos por um conjunto ordenado de interações entre seus participantes. No contexto do desenvolvimento de sistemas, usualmente esta interação se dá entre o sistema em desenvolvimento e os atores externos, que podem ser usuários ou outros sistemas. Segundo Carroll [10], a propriedade que melhor define um cenário é o fato do mesmo projetar uma descrição concreta de uma atividade em que o usuário se engaja no momento em que está realizando uma tarefa específica.

Cenários têm sido tradicionalmente utilizados no processo de elicitação dos requisitos de sistemas de software. Neste contexto, cenários são utilizados para descrever as situações de uso do sistema pelos seus usuários e os relacionamentos entre o sistema em

desenvolvimento e outros sistemas externos, auxiliando no entendimento e na descoberta de novos requisitos.

Leite et al. [8][9] acreditam que cenários podem ser utilizados e produzidos não somente na fase de definição de requisitos, mas durante todo o processo de desenvolvimento de um sistema. Desta maneira existem desde cenários iniciais que modelam o macrosistema onde o sistema será desenvolvido e instalado até cenários que representam as interações dos usuários com o sistema quando este estiver em funcionamento, passando por várias versões intermediárias ao longo do processo de desenvolvimento do software.

Nossa visão da utilização de cenários corresponde a este espectro mais amplo de uso. Neste contexto, conjuntos de cenários são criados e utilizados em todas as fases do processo de desenvolvimento. Utilizando como exemplo o ciclo de vida tradicional (cascata), seriam criados conjuntos de cenários correspondentes aos requisitos, ao projeto (desenho), a implementação, aos testes e à manutenção. Da mesma forma que os outros artefatos do processo, as informações descritas nos cenários estão em constante evolução. Mudanças realizadas em um cenário devem ser propagadas em ambas as direções. Por exemplo, mudanças nos cenários de projeto devem ser propagadas tanto para os cenários de implementação quanto para os de análise e de requisitos.

Em relação à técnica de casos de uso proposta por Jacobson [11], nosso conceito de cenário difere do conceito de cenário ali apresentado. Nossos cenários podem ser considerados casos de uso com maior contextualização, possuindo uma representação mais sofisticada e uma utilização mais ampla durante o processo de desenvolvimento.

2.1 Notação para a Descrição de Cenários

A notação para cenários utilizada neste trabalho é a proposta por Leite [9]. Esta notação utiliza uma linguagem natural semi-estruturada (ancorada no mundo real), partindo da premissa que a utilização da linguagem da aplicação, e não a do software, facilita o entendimento e a validação dos requisitos por parte dos clientes. Esta notação é composta pelos seguintes elementos:

Título (*title*): identifica o cenário.

Objetivo (*goal*): estabelece a finalidade de um cenário. O cenário deve descrever de que modo este objetivo deve ser alcançado.

Contexto (*context*): descreve o estado inicial de um cenário, suas pré-condições, o local (físico) e tempo. Na sua definição podem ser especificadas restrições sobre estes elementos (*constraint*).

Recurso (*resource*): identifica os objetos passivos com os quais lidam os atores. Na sua definição podem ser especificadas restrições sobre os objetos a serem lidados pelo cenário (*constraint*).

Ator (*actor*): Pessoa ou estrutura organizacional que tem um papel no cenário.

Episódio (*episode*): Cada episódio representa uma ação realizada por um ator onde participam outros atores utilizando recursos disponíveis.

A figura 2 mostra um exemplo de utilização da notação para a descrição de um cenário para um sistema de controle de luzes, utilizado no estudo de caso apresentado na seção 4.

Um importante complemento para a descrição dos cenários é o Léxico Ampliado da Linguagem (LAL), um metamodelo projetado para ajudar a capturar a linguagem utilizada no domínio. Seu principal objetivo é registrar palavras ou frases peculiares ao domínio em questão. Cada entrada no LAL possui dois tipos de descrição. O primeiro tipo chama-se *Notion*, cujo objetivo é descrever a denotação da palavra ou frase. O segundo tipo, denominado *Behavioral Response*, objetiva descrever a conotação da palavra ou frase, isto é, informações adicionais no contexto. Na descrição do cenário mostrado na figura 2, os termos

sublinhados são os símbolos descritos no LAL. Um exemplo parcial da definição destes símbolos no LAL é mostrado na figura 3, onde os elementos sublinhados correspondem a outros símbolos descritos no LAL.

Title	Malfunction
Goal	how to proceed when a malfunction occurs
Context	4th floor of building 32
Actor	Control system
Resource	Ceiling light groups
Resource	Control panel
Episodes	inform facility manager of malfunction
	inform user of malfunction
	Control system supports the facility manager in finding the reasons of malfunction
	facility manager can correct manually malfunction undetected by the Control system

Figura 2 – Exemplo de descrição de um cenário

Name	Malfunction
Notion	Incorrect behavior of a device
Behavior	There can be a malfunction of the motion detector
	There can be a malfunction of the outdoor light sensor
	All malfunctions are stored and reported on request
	The facility manager has to be informed
	The control system supports the facility manager by finding the reason for a malfunction

Figura 3 – Exemplo de definição do LAL

2.2 Evolução de Cenários

Cenários descrevem o comportamento de um sistema quando inserido no ambiente do usuário. A percepção deste comportamento evolui ao longo do processo de desenvolvimento do sistema, pois novas necessidades são identificadas, bem como aumenta o entendimento do mundo real dos usuários, tornando a evolução uma característica intrínseca dos cenários.

Este caráter evolutivo dos cenários é uma característica aceita pela maior parte dos desenvolvedores de sistemas e obriga a existência de um gerenciamento desta evolução, provendo mecanismos para manter a rastreabilidade, o controle de configuração e a eliminação de inconsistências e desatualizações entre os cenários gerados ao longo do processo de desenvolvimento.

Breitman [12] realizou uma extensa análise sobre evolução de cenários, propondo um modelo genérico de evolução de cenários. Os componentes estáticos deste modelo referem-se aos relacionamentos existentes entre cenários de uma mesma configuração, enquanto que os componentes dinâmicos são caracterizados pelas operações sobre cenários que são aplicadas pelo desenvolvedor durante o processo de desenvolvimento do sistema, ou seja, operações aplicadas sobre os cenários de uma configuração de maneira a criar uma nova configuração. Os conjuntos de relacionamentos e operações foram organizados na taxonomia para evolução de cenários apresentada na figura 4.

Cada um dos relacionamentos e operações entre cenários possui um conjunto de heurísticas que pode ser utilizado na sua identificação. Por exemplo, o relacionamento de Complemento, que relaciona dois cenários que conjuntamente respondem a um objetivo maior, pode ser identificado quando dois cenários possuírem objetivos idênticos, semelhanças entre seus contextos e recursos, atores idênticos e pequena coincidência entre os episódios de

cada um dos cenários envolvidos.

Relacionamentos	Operações	
Aspectos Estáticos	**Aspectos Dinâmicos**	
	intra cenário	**Inter cenários**
Complemento	Inclusão	Fusão
Equivalência	Modificação	Encapsulamento
Contenção	Remoção	Consolidação
Pré-Condição		Divisão
Detour		Divisão Múltipla
Exceção		Especialização
Inclusão		
Possível Precedência		

Figura 4 – Taxonomia para evolução de cenários

Nossa abordagem para o rastreamento da evolução dos cenários busca identificar quais operações foram aplicadas pelos desenvolvedores ao longo do processo de desenvolvimento. A implementação da identificação dos relacionamentos existentes entre cenários se tornou necessária pois, além desta informação adicional ser importante para um melhor entendimento do sistema em desenvolvimento, é utilizada como heurística para identificação das operações entre cenários.

3 Rastreamento da Evolução de Cenários

A abordagem proposta [13] para o rastreamento da evolução baseia-se na captura da história da evolução de cada artefato através da análise das diferenças existentes entre duas versões consecutivas do artefato. A figura 5.A apresenta uma visão geral da abordagem utilizada, implementada na ferramenta DracoTrackTool. O uso do mecanismo se inicia a partir da obtenção (*check-out*) de um conjunto de artefatos armazenados e gerenciados pelo mecanismo de rastreamento. Estes artefatos são modificados pelos desenvolvedores, que realizam uma série de operações sobre eles utilizando alguma ferramenta externa. Os artefatos modificados são então submetidos ao mecanismo de rastreamento (*check-in*) onde é executada a atividade de geração das transformações de rastreamento. Esta atividade tem como objetivo gerar um conjunto de transformações que represente as mudanças realizadas pelos desenvolvedores ao evoluírem os artefatos de sua versão inicial, obtida no momento do *check-out*, para a versão subseqüente, inserida no momento do *check-in*. A geração destas transformações baseia-se na comparação das duas versões do artefato, identificando quais mudanças foram realizadas pelo desenvolvedor.

Estas mudanças constituem o *delta* entre as versões e representam o resultado de um conjunto de ações realizadas pelo desenvolvedor. O mecanismo prevê ainda a utilização de uma técnica de reconhecimento de planos [14] procurando identificar, dentre os planos preexistentes, aquele que fornece um resultado similar ao expresso no *delta* que foi anteriormente computado. O plano identificado fornece informações sobre qual o objetivo do desenvolvedor ao realizar a mudança do artefato original e, em conjunto com informações do contexto do trabalho do desenvolvedor (por exemplo, sua função e a fase atual do processo de desenvolvimento), qual a justificativa para a ação realizada. O armazenamento destas informações e o controle de versões são realizados pelo Gerenciador de Configurações. Se nenhum plano for identificado durante esta atividade um novo plano pode ser criado pelo desenvolvedor e utilizado em outras interações.

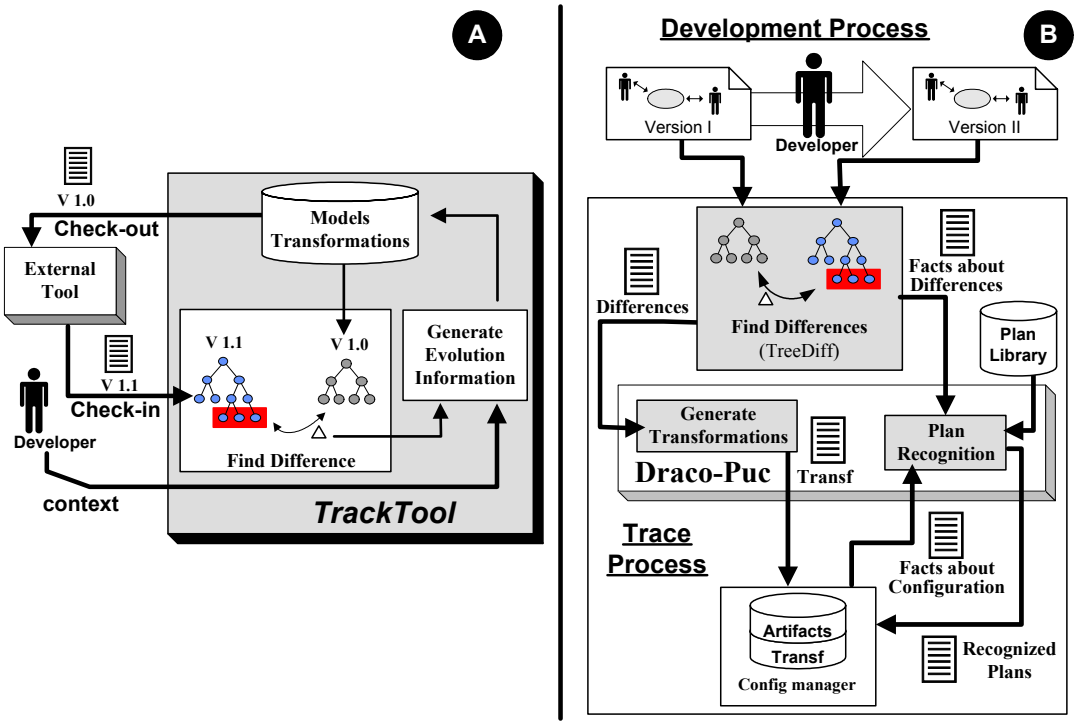

Figura 5 - Mecanismo de Rastreamento. A) Visão Geral e B) Visão detalhada do processo.

O processo de rastreamento da evolução de artefatos é realizado segundo as atividades mostradas na figura 5.B. Inicialmente é executada a atividade de Procura por Diferenças, onde duas versões de um artefato são comparadas, obtendo-se como resultado uma descrição das suas diferenças e um conjunto de fatos observados sobre as duas versões. A partir da descrição das diferenças é executada a atividade de Gerar Transformações, obtendo-se o conjunto de transformações equivalentes a estas diferenças. Os fatos observados sobre estas diferenças, em conjunto com os fatos sobre mudanças na configuração do sistema, servem como entrada para a atividade de Reconhecimento de Planos. Esta atividade procura identificar qual o plano de uma biblioteca de planos que pode ser utilizado para explicar os fatos observados. Todas estas informações são então armazenadas pelo gerenciador de configuração.

3.1 Representação dos Artefatos

De maneira a aumentar o potencial de uso do mecanismo de rastreamento proposto, permitindo o uso de ferramentas externas na edição dos artefatos, optou-se por utilizar a linguagem XML na descrição dos artefatos do sistema. O tratamento destes artefatos descritos em XML pela Máquina Draco-PUC é viabilizado pela utilização de um mecanismo de importação de aplicações XML [15]. Este mecanismo é constituído por um conjunto de transformações da Máquina Draco-Puc que cria de maneira automática um novo domínio Draco-PUC para cada definição de aplicação XML (em DTD ou XML Schema).

A partir da notação para descrição dos cenários apresentada na seção 2.1, foram criados seus documentos de definição de tipos (DTD), permitindo que a descrição dos cenários e do LAL fossem realizados em XML. A aplicação do mecanismo de importação de domínios sobre as DTDs de cenários e do LAL criou de maneira automática os domínios equivalentes na Máquina Draco-Puc, ou seja, a Máquina Draco-Puc pôde ser utilizada na aplicação de transformações sobre cenários e LAL descritos em XML.

Exemplificando o uso da DTD para cenários, a figura 6 mostra a descrição parcial em XML do cenário apresentado anteriormente na figura 2. A principal diferença existente em

relação à notação original de cenários é a existência do elemento *reference*, que permite que se estabeleçam ligações de correspondência entre termos do cenário e suas definições no LAL

```
<scenario>
  <title>malfunction occurs</title>
  <goal address="1">how to proceed when a<reference name="malfunction" version="1"/>occurs</goal>
  <context address="2"><reference name="4th floor of building 32" version="1"/></context>
  <actor address="3"><reference name="Control system" version="1"/></actor>
  <actor address="4"><reference name="facility manager" version="1"/></actor>
  <resource address="5"><reference name="Ceiling light groups" version="1"/></resource>
  <resource address="6"><reference name="Control panel" version="1"/></resource>
  <episode address="7">
    <definition>inform<reference name="facility manager" version="1"/>of<reference name="malfunction" version="1"/></definition>
  </episode>     /* . . . Other episodes removed . . . */
</scenario>
```

Figura 6 – Descrição XML do cenário *malfunction occurs*

3.2 Procura por Diferenças

A procura por diferenças utiliza o algoritmo *TreeDiff* aplicado diretamente sobre as árvores DOM (*Document Object Model*) de cada artefato descrito em XML. Este algoritmo é baseado no algoritmo proposto por Wang et al. [16] e procura a maior subestrutura comum entre duas árvores, apresentando como resultado um conjunto de operações de edição (renomear, remover ou inserir um nó) que deve ser aplicado na primeira árvore de maneira a obter-se a segunda. Cada operação possui um custo associado e o computo da subestrutura comum é realizado procurando-se minimizar o custo desta edição. A complexidade de tempo deste algoritmo é dada por $O(|T_1| \times |T_2| \times \min(H_1, L_1) \times \min(H_2, L_2))$, onde H_i, i = 1, 2 é o tamanho da árvore T_i e L_i, i = 1, 2 é o número de folhas da árvore T_i.

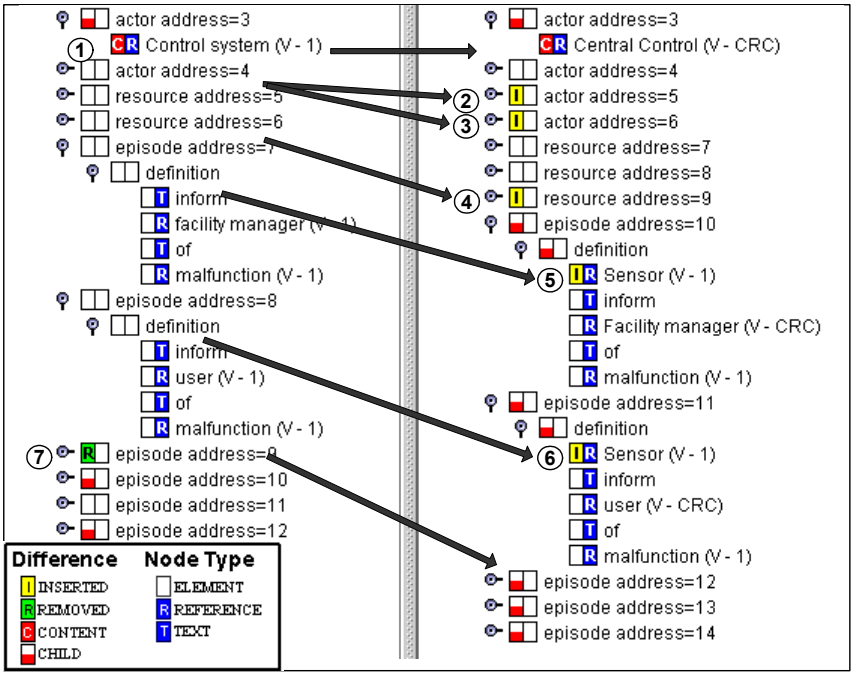

Figura 7 – Resultado da aplicação do algoritmo

A figura 7 mostra uma visualização do resultado da aplicação do algoritmo na identificação das diferenças entre duas versões do cenário *malfunction occurs*. Os ícones dos nós possuem dois retângulos: o da esquerda mostra informações sobre as diferenças (nó inserido, removido ou conteúdo renomeado) e o da direita sobre o tipo do nó (elemento, referência ou texto). Os nós marcados como *child* (metade inferior preenchido no retângulo da esquerda) são aqueles que não sofreram modificações, porém algum dos seus descendentes possui alguma diferença, como por exemplo o caso do ator pai da diferença marcada com o número 1 na árvore da esquerda.

A partir das diferenças identificadas são geradas suas descrições, utilizadas na atividade de Geração das Transformações, e a lista de fatos observados, utilizados na atividade de reconhecimento de planos. A geração destas informações será mostrada nas seções seguintes.

3.2.1 Geração da Descrição das Diferenças

A descrição das diferenças é realizada utilizando-se o domínio de diferenças Draco-PUC. A sintaxe deste domínio é mostrada na figura 8.A. Cada diferença encontrada é mapeada diretamente para a construção do tipo *diffElement* correspondente: *replaceSpec* para mudança de conteúdo, *insertSpec* para inserção de um nó e *removeSpec* para a remoção de um nó. A construção *replaceSpec* especifica o tipo do nó que foi modificado, o conteúdo original do nó e o novo conteúdo. A construção *insertSpec* especifica o tipo do nó que foi inserido, o nó a ser inserido e o local de inserção. A construção *removeSpec* define o tipo do nó que foi removido, o nó a ser removido e a localização do mesmo.

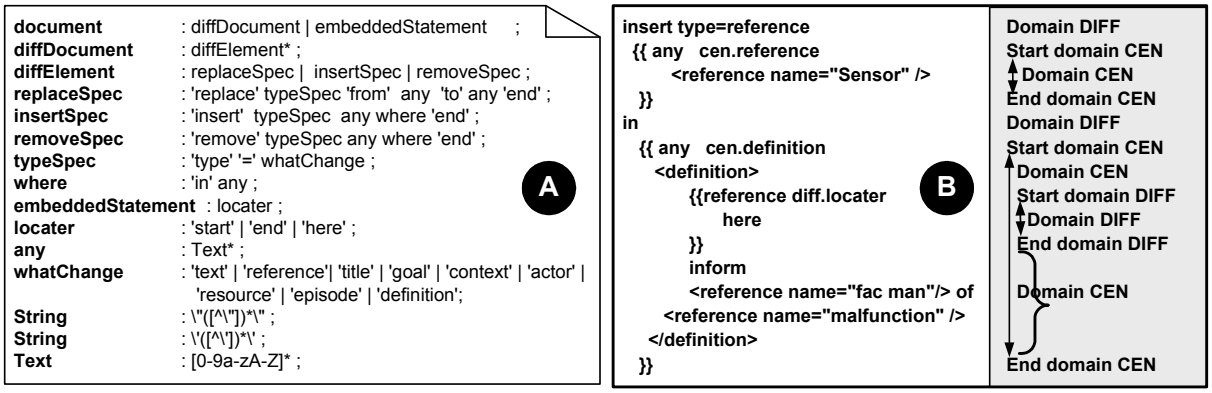

Figura 8 – Domínio de diferenças. A) Sintaxe e B) Exemplo

O local de inserção ou remoção de um nó é definido através da cópia do trecho da árvore original acrescido de uma marcação do local exato de aplicação da diferença. Na marcação deste local é utilizado um poderoso recurso da Máquina Draco-PUC que é a possibilidade de utilizar descrições pertencentes a sintaxe de diversos domínios dentro de um mesmo programa

A figura 8.B mostra a descrição da diferença marcada com o número 5 na figura 7, que corresponde à inserção de uma referência para o símbolo *Sensor* do LAL no local marcado (trecho sublinhado) da definição de um episódio. Esta descrição foi gerada a partir do resultado da aplicação do *TreeDiff* e utiliza tanto a linguagem do domínio de diferenças (*Diff*), na definição do tipo da diferença, quanto a linguagem do domínio de cenários (*Cen*), na especificação dos elementos inseridos e do local de inserção.

Na geração das descrições das diferenças três questões tiveram que ser tratadas. Primeiro, a descrição das diferenças é influenciada pela ordem com que as diferenças são tratadas para gerar sua descrição, ou seja, são obtidas descrições diferentes para o mesmo conjunto de diferenças de acordo com a ordem com que foram geradas. Para solucionar este problema tornou-se necessário garantir que a ordem de geração da descrição das diferenças e a ordem de aplicação das transformações correspondentes a cada diferença fossem as mesmas.

Uma segunda questão refere-se ao fato de que a aplicação das transformações é realizada de forma seqüencial, uma para cada diferença encontrada. Desta maneira, a transformação que gera uma determinada diferença será aplicada sobre a árvore modificada pela aplicação das transformações anteriores. A solução adotada foi, para cada diferença identificada, aplicar a diferença na árvore original imediatamente após a sua descrição ter sido gerada, possibilitando que todas as diferenças subseqüentes levassem em consideração as diferenças já tratadas.

O último ponto que necessitava ser tratado era quanto ao nível de descrição da diferença. Este nível relaciona-se à granularidade do elemento considerado na diferença: quanto mais baixo o nível maior será a granularidade do elemento. Em seu nível mais baixo (o nível 0), a diferença relaciona-se apenas ao nó que contém a diferença. Cada nível superior é formado pelo pai da diferença no nível imediatamente inferior. Quanto mais baixo o nível da descrição menor será a quantidade de informações necessárias na descrição e mais simples serão as transformações geradas. O problema da utilização dos níveis mais baixos reside na possibilidade de ocorrer a ativação da transformação em outros pontos do artefato e não somente naqueles que necessitam ser modificados. Por exemplo, se utilizarmos o nível mais baixo de descrição para a substituição de um texto por outro, esta transformação será aplicada em todos os locais da árvore que possuem o mesmo texto. A implementação do mecanismo permite a configuração do nível de geração das descrições em função do tipo de diferença encontrada (renomear, inserir e remover) e do tipo do nó (texto, referência ou elemento).

3.2.2 Geração da Lista de Fatos Observados

O outro resultado da atividade de identificação de diferenças que deve ser produzido é uma lista de fatos observados sobre as diferenças identificadas pelo algoritmo *TreeDiff*. Estes fatos são utilizados pela técnica de reconhecimento de planos, que será apresentada na seção 3.4, que tem como objetivo reconhecer quais dos planos existentes em uma biblioteca de planos podem ser utilizados para explicar os fatos observados. Em conseqüência, a definição de quais os fatos observados que devem ser gerados a partir das diferenças é função da biblioteca de planos que será utilizada.

3.3 Geração das Transformações

A geração das transformações equivalentes às diferenças encontradas entre os artefatos é realizada através da aplicação do transformador *HandleDiff* sobre a descrição das diferenças produzidas na atividade anterior. Para cada descrição de diferença existente é gerada uma nova transformação que, quando aplicada sobre o artefato original, irá modificar este artefato de maneira a produzir a diferença correspondente. A transformação correspondente a diferença mostrada na figura 8.B é apresentada na figura 9.

```
Set Of Transforms SOT1
    Transform TFM1
    Lhs:   {{any cen.definition
        <definition>
            inform
            <reference name="facility manager" version="1"/>of
            <reference name="malfunction" version="1"/>
        </definition>
    }}
    Rhs:   {{any cen.definition
        <definition>
            <reference name="Sensor" version="1"/>
            inform
            <reference name="facility manager" version="1"/>of
            <reference name="malfunction" version="1"/>
        </definition>
    }}
```

Figura 9 - Transformação gerada

3.4 Reconhecimento de Planos

O reconhecimento de planos pode ser definido como a tarefa realizada com o objetivo de determinar o melhor conjunto hierárquico de planos que explica uma série de fatos observados [17]. Esta tarefa geralmente é limitada a um domínio especifico que limita e restringe os fatos que possam vir a serem observados. Nossa implementação da técnica é baseada em um domínio Draco-PUC de planos (Plan), constituído de uma linguagem de planos e de um transformador que executa a construção de planos a partir dos fatos observados. A linguagem de planos é utilizada para descrever 3 tipos diferentes de programas: a biblioteca de planos, que define todos os fatos que podem ser observados e os planos que podem ser selecionados para explicar estes fatos; a descrição dos fatos efetivamente observados a partir dos quais se deseja realizar o reconhecimento do plano correspondente; e a descrição dos planos reconhecidos, gerada pela aplicação da técnica e que

contém os planos que explicam o conjunto de fatos observados e a sua justificativa, expressa através de uma hierarquia de eventos inferidos construída a partir dos fatos observados.

A figura 10.A mostra um exemplo parcial de descrição de uma biblioteca de planos neste domínio, utilizada para o reconhecimento das operações entre cenários. Os eventos definidos como *EndEvent* caracterizam os planos que podem ser selecionados para explicar um subconjunto dos fatos observados (o evento *fusionOperation* da figura 10.A), ou seja, representam o resultado da aplicação da técnica. Os demais eventos são utilizados como passos intermediários inferidos a partir de outros eventos ou fatos que foram diretamente observados.

Cada evento definido na biblioteca possui um nome, um conjunto de parâmetros e pode possuir ainda dois tipos de relacionamentos: de decomposição (*composedBy*) e de generalização (*isa*). O nome identifica unicamente o evento e os parâmetros são utilizados para contextualizar a ocorrência do evento dentro de uma determinada situação de uso da técnica. O relacionamento de generalização indica que toda vez que o evento declarado for inferido, o evento que representa a generalização deste automaticamente é inferido também. Neste caso a declaração dos parâmetros destes eventos devem ser os mesmos. A decomposição representa um relacionamento todo-parte e indica os eventos (partes) que contribuem para que o evento declarado (todo) seja inferido.

A

```
PlanLibrary
begin
  Event episodeModification(A,B);
  Event complementRelationship(A,B);
  Event scenarioRemoved(A);
  Event episodeRemoved(A,B)
    begin  isa episodeModification  end
  Event episodeAdded(A,B)
    begin  isa episodeModification  end
  Event episodeMoved(A,B,C)
  begin
    composedBy
      episodeRemoved(A,C) by source with weight = '1.0';
      episodeAdded(B,C) by target with weight = '1.0';
  end
  Event fusionOperation(A,B)  is EndEvent
  begin
    composedBy . . .
  end
end
```

B

```
01 Observations
02 begin
03   complementRelationship
       ("malfunction":"1" , "malfunction occurs":"2" ) [0.7] ;
04   scenarioRemoved("malfunction":"1");
05   episodeRemoved("malfunction":"1" , "epis3");
06   episodeAdded("malfunction occurs":"2" , "epis3");
07 end
```

Figura 10) A. Biblioteca de Planos e B. Descrição de Fatos Observados

A figura 10.B apresenta um exemplo de descrição de fatos observados para a biblioteca de planos da figura 10.A. Os fatos observados podem ser gerados por quaisquer componentes do mecanismo de rastreamento. As principais fontes de fatos observados são o gerenciador de configuração, o algoritmo de identificação de diferenças (*TreeDiff*), a própria aplicação anterior da técnica de reconhecimento de planos ou ainda a aplicação de transformações na Máquina Draco-PUC. O gerenciador de configuração é responsável pela geração dos fatos relacionados às mudanças de configuração do sistema, tais como a inserção ou remoção de artefatos. Um exemplo de fato gerado por este gerente é o que aparece na linha 4 da figura 10.B, que representa a remoção do cenário *malfunction* – Versão I da configuração que está sendo analisada. O algoritmo *TreeDiff* gera os fatos relacionados às diferenças entre os artefatos analisados, como os fatos das linhas 5 e 6 da figura 10.B que mostram, respectivamente, que o episódio 3 do cenário *malfunction* – Versão I foi removido e que foi inserido o episódio 3 no cenário *malfunction occurs* – Versão II. Fatos observados podem ser

criados pela aplicação anterior da técnica de reconhecimento de planos, como o mostrado na linha 3 da figura 10.B, que informa que os cenários *malfunction* – Versão I e *malfunction occurs* – Versão I possuem um relacionamento de complemento. O valor mostrado (0,7) representa um nível de certeza quanto a existência do relacionamento. A aplicação de transformações é uma importante fonte de fatos observados. Transformações podem ser escritas para reconhecer determinados padrões que caracterizam a ocorrência de um fato observado.

A figura 11 mostra uma visão geral da aplicação da técnica de reconhecimento de planos na identificação dos relacionamentos e das operações entre cenários, bem como os diversos conjuntos de fatos observados utilizados.

O resultado da aplicação da técnica de reconhecimento de planos é formado por vários conjuntos de eventos finais, onde cada conjunto explica todos os fatos observados. O conjunto a ser escolhido como resultado final pode ser aquele que possuir o maior grau médio de satisfação dos eventos ou aquele que possuir o menor número de eventos finais.

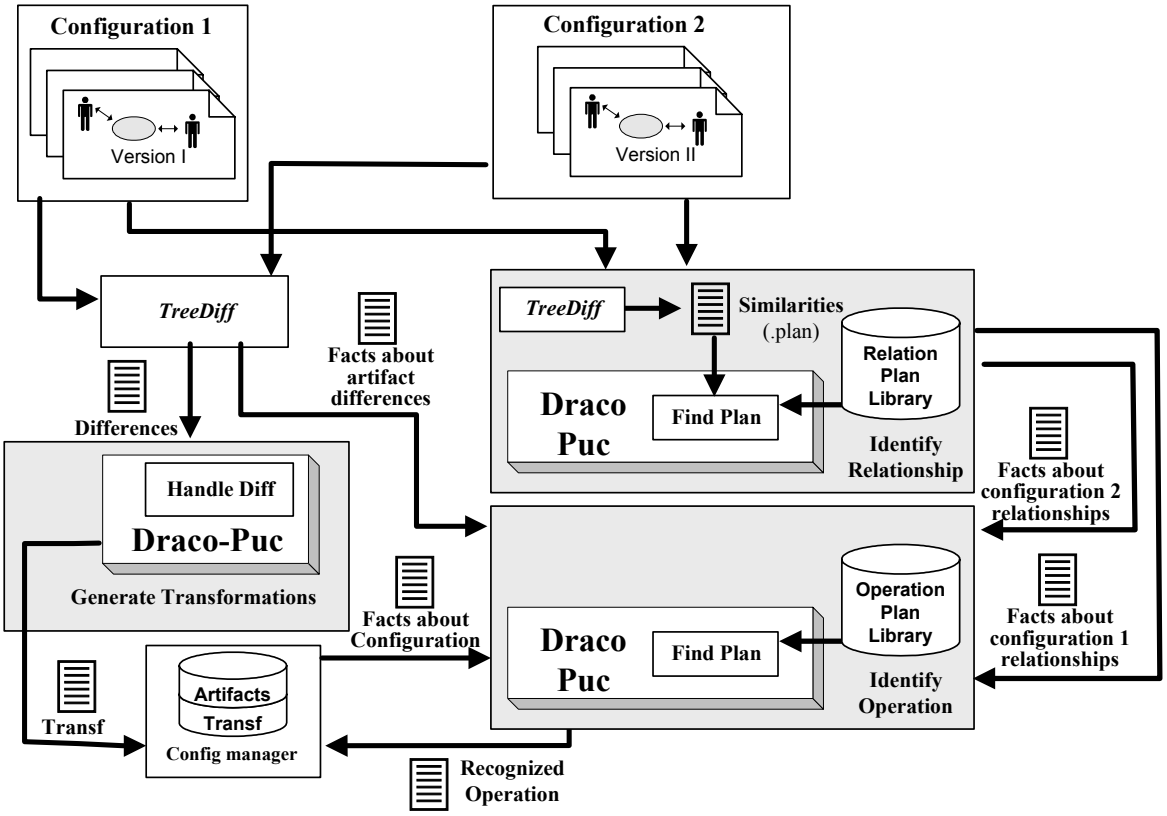

Figura 11 - Visão geral da aplicação do reconhecimento de planos

4 Estudo de Caso: Sistema de Controle de Luzes

O sistema escolhido para este estudo de caso é o que foi utilizado na tese de Breitman [12] na definição de sua taxonomia para a evolução dos cenários. Este sistema faz a simulação do controle da iluminação do Departamento de Informática da Universidade de Kaiserslautern e foi proposto inicialmente como estudo de caso em um workshop de engenharia de requisitos realizado no Schloss Daghstuhl, Alemanha em 1999 [18]. O motivo da escolha deste sistema para ser utilizado como estudo de caso é a existência de todo o histórico do seu desenvolvimento, bem como dos relacionamentos e operações entre cenários, e desta maneira poder ser utilizado na confrontação com os relacionamentos e operações identificados através do sistema descrito neste capítulo. Este histórico é composto por 8 configurações, totalizando

95 cenários, 100 relacionamentos entre eles e 73 operações aplicadas.

4.1 Resultados Obtidos

Os resultados obtidos a partir da execução do estudo de caso, utilizando o mecanismo de rastreamento apresentado neste artigo, foram os seguintes:

a) Foram geradas corretamente todas as transformações de rastreamento. Este fato foi comprovado através da aplicação do algoritmo *TreeDiff* entre o artefato que foi utilizado na geração da transformação e o artefato gerado pela aplicação da transformação, não tendo sido encontradas nenhuma diferença entre eles.

b) Quanto à técnica de reconhecimento de planos:

i) Foi notada uma baixa eficiência na implementação do reconhecimento de planos através de transformações.

ii) A linguagem utilizada para descrever os planos é adequada, permitindo que se capture todas as heurísticas necessárias à identificação das operações realizadas pelo desenvolvedor.

c) Quanto ao reconhecimento dos relacionamentos, ocorreu um número razoável de resultados falsos positivos, ou seja, o sistema identificou um relacionamento errado. O principal motivo que deu origem a estes erros foi o fato de que as heurísticas não permitem identificar com certeza qual o relacionamento existente.

d) Quanto à eficácia do reconhecimento das operações, a totalidade das operações intra-cenários foi identificada corretamente, bem como a maior parte das operações inter-cenários. A principal causa de erro na identificação das operações foi devido ao grande número de ações realizadas entre duas versões consecutivas dos artefatos, quando algumas ações foram atribuídas a outras operações. Uma causa de erro secundária foi devido às heurísticas utilizadas não poderem precisar exatamente qual a operação a ser reconhecida.

e) Quanto à eficiência do reconhecimento das operações, a quantidade de fatos gerados a serem tratados pelo reconhecimento de planos é muito grande.

4.2 Análise dos Resultados

Quanto aos resultados obtidos pelo estudo de caso, algumas considerações devem ser feitas quanto à eficiência e à eficácia do sistema.

Considerações sobre a eficiência

A utilização do algoritmo *TreeDiff* se mostrou adequada e eficiente, podendo ser utilizada para a identificação das diferenças existentes entre dois artefatos quaisquer descritos em XML.

A utilização do paradigma transformacional na implementação do processo de reconhecimento de planos se mostrou pouco eficiente, não sendo adequada para tratar sistemas grandes onde o número de fatos observados é elevado. Duas alternativas podem ser utilizadas para resolver este problema. A primeira mantendo o enfoque transformacional e otimizando aspectos específicos do transformador, e a segunda abandonando o enfoque transformacional e adotando uma abordagem mais eficiente. Para a segunda alternativa, o enfoque transformacional para o reconhecimento de planos poderia ser substituído por algoritmos que utilizam técnicas baseadas em inferências diretas ou na teoria de grafos, tais como os propostos por Lesh & Etzioni [19] ou Lin & Goebel [20]. Neste caso, devem ser mantidas a mesma linguagem para descrição dos planos atualmente utilizada e o mesmo mecanismo de seleção de planos baseado no seu nível de satisfação.

Considerações sobre a eficácia

Existe a necessidade de refinar e estender o conjunto de heurísticas utilizadas na identificação dos relacionamentos e operações. Para o conjunto de heurísticas utilizadas atualmente torna-se necessária a intervenção do usuário do sistema para escolher, a partir de

uma lista priorizada de planos reconhecidos, qual o plano que realmente pode explicar a situação em análise.

5 Conclusão

5.1 Benefícios

A utilização da tecnologia transformacional na construção do mecanismo de rastreamento da evolução de cenários apresentada neste artigo permite que sejam incorporadas informações sobre a semântica associada a cada ligação, adicionadas as informações de rastreamento de evolução capturadas e armazenadas pelos sistemas de rastreamento usuais. A obtenção das informações de rastreamento relacionadas com a evolução de um artefato original para um novo artefato é basicamente um problema resolvido, realizado normalmente através da utilização de matrizes de rastreamento. Nosso diferencial reside na disponibilização das informações que explicam de que maneira se deu esta evolução, ou seja, além do estado inicial e final de um artefato, são capturadas as informações sobre como foi realizada esta evolução. A nomeação de qual a operação realizada pelo desenvolvedor sobre os cenários, em conjunto com os relacionamentos existentes entre os cenários da configuração inicial, permite identificar os cenários da nova configuração que sofrerão o impacto pelas operações realizadas. A existência de uma série de heurísticas pré-definidas associadas a cada operação, que guiam o desenvolvedor na escolha de qual operação usar na evolução dos artefatos, permite ainda que, a partir do momento em que uma determinada operação foi identificada, seja inferido o porquê de sua aplicação no contexto atual, fornecendo desta maneira a justificativa para sua aplicação.

As transformações geradas podem ser ainda utilizadas na geração de uma série de informações úteis ao gerenciamento dos requisitos, tais como a verificação de dependências entre os artefatos e o impacto das modificações. A geração destas informações pode ser realizada através da aplicação de novas transformações sobre o conjunto formado pelas transformações de rastreamento e pelos planos que foram reconhecidos, identificando, por exemplo, as referências existentes entre os artefatos, inferindo sobre a dependência existente entre eles; as mudanças da configuração do sistema, inferindo os artefatos que foram reunidos em um só; e o conjunto de operações que foram aplicados sobre um artefato até que o produto final do sistema tenha sido obtido, concluindo sobre o impacto das mudanças

O maior nível de automatização do processo de captura das informações sobre a evolução dos cenários permite minimizar o problema da falta de atualização das informações de rastreamento, aumentando o valor destas informações para o desenvolvedor, bem como diminuir o custo de obtenção, atualização e validação das informações de rastreamento.

5.2 Trabalhos futuros

Os trabalhos futuros estão relacionados ao refinamento e extensão do mecanismo de rastreamento e a otimização de sua implementação. Quanto ao refinamento e extensão do mecanismo, devem ser realizados estudos que permitam determinar qual a freqüência de aplicação do processo de captura e como tratar a questão da evolução das transformações de rastreamento. Os trabalhos de otimização referem-se ao refinamento das heurísticas utilizadas para reconhecer relacionamentos e operações entre cenários, bem como a otimização da técnica de reconhecimento de planos.

Finalmente, devem ser realizados estudos envolvendo a evolução de outros artefatos, tais como os Diagramas de Classes ou cartões de CRC, com o objetivo de definir uma taxonomia para sua evolução, nos moldes da taxonomia de cenários utilizada neste trabalho.

5.3 Comparação com Trabalhos Existentes

A principal característica que difere o mecanismo de rastreamento proposto neste trabalho das ferramentas existentes reside na maneira pela qual as ligações relacionadas ao

processo (Evolui-Para e Justifica) são capturadas, armazenadas e manipuladas. Enquanto que na maior parte dos mecanismos existentes estas ligações são armazenadas na forma de relacionamentos entre o artefato original e o novo artefato, o nosso mecanismo armazena a própria ação executada pelo desenvolvedor na forma de transformações que podem ser aplicadas ao artefato original para que se obtenha o novo artefato.

Egyed [2] apresenta uma proposta de rastreamento dirigida por cenários, onde as informações são capturadas a partir da observação de cenários de teste sendo executados pelo sistema. Esta proposta necessita que tanto os modelos quanto uma versão executável do sistema estejam disponíveis. A informação gerada mostra apenas qual os componentes do sistema que são utilizados na execução dos cenários de teste (ligação Satisfaz). Nossa proposta pode ser utilizada em qualquer fase do desenvolvimento pois não depende da análise dinâmica do sistema, sendo ainda utilizada para tratar as ligações de Evolui-Para e Justifica.

Pinheiro e Goguem [21] utilizam uma abordagem formal para manter a rastreabilidade de um sistema na forma de relacionamentos entre objetos. Estas informações precisam ser informadas manualmente pelo desenvolvedor. Nossa proposta permite que grande parte da informação seja capturada automaticamente.

Uma abordagem que apresenta algumas similaridades com a nossa é apresentada por Antoniol et. al. [22] que calcula a diferença entre versões para auxiliar o desenvolvedor a tratar inconsistências entre as versões, apontando para regiões do código onde as diferenças estão concentradas. É utilizada uma representação intermediária (AOL) e computado o delta analisando este código. A principal diferença em relação a nossa proposta reside nesta característica, pois utilizamos diretamente as árvores de sintaxe abstratas e desta maneira não perdemos nenhuma informação ao fazermos a transformação entre o programa original e sua representação. Outra diferença é que, enquanto a proposta de Antoniol trata somente as informações existentes no Diagrama de Classes, nossa proposta pode ser aplicada em qualquer tipo de modelo. O uso das informações capturadas também difere, pois nosso objetivo é utilizá-las para manter os diversos tipos de ligações e não somente mostrar inconsistências entre versões.

Uma proposta que também utiliza transformações é apresentada por Baxter & Mehlich [23], que propõe que os refinamentos executados durante o desenvolvimento devem ser armazenadas na forma de transformações. Estas transformações poderiam então ser utilizadas no processo de engenharia reversa de outros sistemas a partir do código fonte, com o objetivo de obter os modelos do sistema. A implementação desta proposta depende da existência de uma especificação formal do sistema sobre a qual são aplicadas transformações para derivar o código do sistema, obrigando que o desenvolvedor utilize métodos formais no desenvolvimento. Nossa proposta permite a utilização de qualquer método ou ferramenta já conhecidos pelos desenvolvedores, tendo em vista que a geração das transformações é realizada diretamente sobre os resultados das ações realizadas pelo desenvolvedor. Finalizando, o enfoque de Baxter & Mehlich é na engenharia reversa de sistemas não tratando a questão da rastreabilidade.

5.4 Extensão da Abordagem para outros artefatos

A abordagem apresentada neste artigo é utilizada na captura de informações de rastreamento relacionadas à evolução de cenários. A utilização desta abordagem em outros artefatos depende da definição do conjunto de operações possíveis de serem realizadas sobre cada tipo de artefato, nos moldes da taxonomia para evolução de cenários apresentada na seção 2.2 e da especialização das atividades e produtos da abordagem proposta que são dependentes do tipo do artefato. Para cada tipo de artefato deve ser criada uma biblioteca de planos correspondente, função do conjunto de operações específicas de cada artefato, e os

fatos observados a serem gerados devem ser aqueles relacionados a cada biblioteca. As demais atividades permanecem inalteradas, podendo ser aplicadas diretamente a qualquer tipo de artefato.

6 Referencias Bibliográficas

[1] Gotel, O., Finkelstein, A., *An Analysis of the Requirements Traceability Problem*, in Proc. of the First International Conference on Requirements Engineering, p 94-101, 1994.

[2] Egyed, A., *A Scenario-Driven Approach to Traceability*, in Proc. of the 23rd International Conference on Software Engineering, p 123-134, 2001.

[3] Palmer, J., *Traceability, Software Requirements Engineering*, R.H. Thayer and M. Dorfman, eds., p. 364-374, 1997.

[4] Hamilton, V., Beeby, M., *Issues of Traceability in Integrating Tools*, in Proc. of the IEE Colloquium on Tools and Techniques for Maintaining Traceability during Design, p 4/1-4/3, Dec 1991.

[5] Neighbors, J., *The Draco Approach to Constructing Software from Reusable Components*, IEEE Transactions on Software Engineering, SE-10, p 564-573, Sep. 1984.

[6] Freeman, P., *A Conceptual Analysis of the Draco Approach to Constructing Software Systems*, IEEE Transactions on Software Engineering, SE-13(7), p 830-844, July 1987.

[7] Ramesh, B., Jarke, M., *Toward Models for Requirements Traceability*, IEEE Transactions on Software Engineering, p 58-93, Vol 27, No 1, 2001.

[8] Breitman, K., Leite, J., *Scenario-Based Software Process*, in Proc. of the 7th International Conference and Workshop on the Engineering of Computer Based Systems, p 375-381, 2000.

[9] Leite, J., C., et al., *Enhancing a Requirements Baseline with Scenarios*, in Proc. of the Third IEEE International Symposium on Requirements Engineering (RE'97) – Annapolis, USA – IEEE Computer Society Press, p 44-53, 1997.

[10] Carroll, J.M., Scenario Based Design: Envisioning Work and Technology in System Development, John Wiley and Sons, 1995.

[11] Jacobson, I., *Object Oriented Software Engineering: a Use Case Driven Approach*, Addison Wesley, 1994.

[12] Breitman, K., *Evolução de Cenários*, Tese de Doutorado, PUC/RJ, Maio, 2000. O histórico completo dos exemplos encontra-se disponível em http://stones.les.inf.puc-rio.br/Karin/exemplo/index.html

[13] Bergmann, U., *Evolução de Cenários Através de um Mecanismo de Rastreamento Baseado em Transformações*, Tese de Doutorado, PUC-Rio, 2002.

[14] Kautz, H.A., Allen, J.F., *Generalized Plan Recognition*, in Proceedings of the 5th Nat. Conf. AI, p 32-37, 1986.

[15] Bergmann, U., Leite, J.C., *Domain Networks in the Software Development Process*, in Proceedings of the 7th International Conference on Software Reuse, p 194-209, 2002.

[16] Wang, J., et al., *An Algorithm for Finding the Largest Approximately Common Substructures of Two Trees*, IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 20, No. 8, p 889-895, Aug 1998.

[17] Quilici, A., Yang, Q., *Applyng Plan Recognition Algorithms to Program Understanding*, in Proceedings of the 11th Knowledge-Based Software Engineering Conference (KBSE), p 96-103, 1996.

[18] Requirements Capture, Documentation and Validation – Dagstuhl- Seminar Report 242 – 13.06.99 – 18.06.99 (99241) - Schloss Dagstuhl, 1999. Disponível em http://rn.informatik.uni-kl.de/~recs/problem.

[19] Lesh, N., Etzioni, O., *A Sound and Fast Goal Recognizer*, in Proc. 14th Int. Joint Conf. AI, p 1704-1710, 1995.

[20] Lin, D., Goebel, R., *A Message Passing Algorithm for Plan Recognition*, in Proc. 12th Int. Joint Conf. AI, volume 1, p 280-285, 1990.

[21] Pinheiro, F., Goguem, J., *An Object Oriented Tool for Tracing Requirements*, IEEE Software, 13(2), p 52-64, 1996.

[22] Antoniol, G., Canfora, G., De Lucia, A., *Maintaining Traceability During Object-Oriented Software Evolution: a Case Study*, in Proceedings of the International Conference on Software Maintenance, p 211-219, 1999.

[23] Baxter, I. D., Mehlich, M., *Reverse Engineering is Reverse Forward Engineering*, Proceedings of the 4th Working Conference on Reverse Engineering, IEEE Computer Press, p 104-113, 1997.