1
Introdução

1.1.
Motivação

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].

A importância da rastreabilidade na evolução de sistemas é um fato amplamente reconhecido:

a) Egyed [2] define a rastreabilidade como atividade essencial ao processo de desenvolvimento de software;

b) 4% dos gastos em tecnologia de informações do Departamento de Defesa americano são realizados em rastreabilidade [3];

c) a existência de um esforço para incorporar mecanismos de rastreamento no desenvolvimento de sistemas, como o previsto no padrão MIL-STD-2167A [4] e no modelo CMM [5];

d) Palmer [6] afirma que o rastreamento é essencial para mostrar aos desenvolvedores que os modelos atendem aos requisitos do sistema; e

e) Hamilton e Beeby [7] mostram a importância do rastreamento na identificação dos impactos oriundos de mudanças nos requisitos.

Apesar 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, estas informações de rastreamento freqüentemente não são confiáveis devido a inconsistência causada pela evolução distinta do sistema e dos seus requisitos. A principal conseqüência desta inconsistência é a perda efetiva do seu valor para o desenvolvedor, que passa a não utilizar as informações armazenadas. Cabe ressaltar que não é suficiente uma

acurácia parcial da informação: o desenvolvedor não tem condições de saber qual informação é correta e qual encontra-se desatualizada.

Além da desatualização das informações de rastreamento, uma série de outros fatores dificultam a obtenção da rastreabilidade: os padrões de desenvolvimento existentes não fornecem uma boa orientação de como obter a rastreabilidade; a não adoção de uma maneira sistemática de rastreabilidade pelas organizações de desenvolvimento; os mecanismos e modelos de rastreamento existentes variam muito e freqüentemente não são bem compreendidos[3]; a dificuldade de realizar o mapeamento entre os domínios do problema e da solução [8]; e o alto custo para obter, atualizar e validar as informações de rastreamento.

Garantir que um software atenda as necessidades dos fregueses e facilitar a manutenção e extensão do sistema durante seu tempo de vida são qualidades fundamentais do processo de desenvolvimento de software. As freqüentes mudanças nos sistemas tornam difícil manter o rastreamento entre o sistema implementado e os requisitos originais ou modificados.

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 a sua efetiva implantação, visualizamos o uso de sistemas transformacionais como uma alternativa viável na implementação destes mecanismos. Sistemas transformacionais são ferramentas de apoio ao engenheiro de software que permitem a manipulação estrutural e semântica de sistemas. Estas ferramentas têm sido aplicadas a diversos tipos de tarefa tais como síntese de programas e reengenharia. No contexto do reastreamento de requisitos, acreditamos que seu uso permitirá um maior nível de automatização das tarefas de captura e manutenção das informações de rastreamento entre artefatos.

Nesta tese, os artefatos que serão utilizados como estudo de caso serão os 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.

Cenários têm sido amplamente 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.

1.2. Objetivos

Conforme discorrido acima, coloca-se a confiabilidade e aderência das informações de rastreamento como um desafio para a pesquisa em engenharia de software. Nossa opinião é de que as soluções para obter uma maior confiabilidade e aderência destas informações passam, obrigatoriamente, pela automação das tarefas relacionadas à captura e manutenção das informações de rastreamento. Nossa tese é de que esta automatização pode ser conseguida através do uso de sistemas transformacionais. Portanto, o objetivo principal deste trabalho é o estudo de sistemas transformacionais na sistematização e automação do processo de rastreamento, através da definição, especificação e implementação de um mecanismo de rastreamento baseado na tecnologia transformacional. Para alcançar este objetivo cremos ser necessário:

- Especificar um mecanismo de rastreamento baseado na tecnologia transformacional centrado na automatização das tarefas de captura e manutenção das informações de rastreamento; e
- Estudar o uso particular deste mecanismo na evolução dos cenários de um sistema de software, procurando capturar as informações das operações sobre cenários definidas na taxonomia de evolução de cenários proposta por Breitman [10].

1.3. Benefícios Esperados

A partir da realização dos estudos definidos na seção anterior, visualizamos como o principal benefício a ser obtido a verificação da viabilidade do uso de sistemas transformacionais na captura e manutenção das informações de rastreamento. Essencial a verificação desta viabilidade, um segundo benefício envolve a especificação e implementação de um mecanismo de rastreamento baseado na tecnologia transformacional.

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, devido a automatização do processo de captura e manutenção das informações de rastreamento; a sistematização do processo de rastreamento; e a diminuição do custo para se obter, atualizar e validar as informações de rastreamento, através do aumento da automatização do processo 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.

1.4. Revisão da Literatura

A revisão da literatura envolveu as diversas propostas existentes para obter e manter informações de rastreamento dos artefatos oriundos do processo de desenvolvimento de sistemas. Foram incluídos nesta revisão os trabalhos que, apesar dos autores não terem visualizado sua aplicação no rastreamento dos artefatos, apresentavam técnicas que poderiam ser estendidas, adaptadas ou até usadas diretamente na obtenção de informações de rastreamento.

A literatura apresenta diversas publicações que abordam a necessidade de se manter a rastreabilidade em todas as fases do desenvolvimento, enfatizando principalmente o rastreamento dos requisitos. Poucos trabalhos abordam a questão da geração, validação e uso das informações de rastreamento.

No restante desta seção, estaremos resumindo alguns trabalhos, agrupados segundo suas principais características, que acreditamos serem relevantes para o tratamento da rastreabilidade. Estes trabalhos, como dito anteriormente, podem não estar diretamente relacionados com rastreabilidade, porém apresentam conceitos e resultados que acreditamos serem importantes para uma melhor compreensão dos resultados já disponíveis.

1.4.1. Análise Léxica

Esta abordagem é baseada na especificação de expressões regulares que são utilizadas na procura de padrões nos artefatos. Sua principal vantagem é a versatilidade, pois pode ser aplicada em diversos tipos de artefatos tais como documentação e fonte. Suas principais desvantagens residem na pequena

granularidade dos padrões a serem localizados, dificultando a identificação de construções em um nível de abstração mais alto; e na impossibilidade de distinguir as construções sintáticas, como por exemplo se o trecho amarrado por uma expressão regular é um comentário ou código fonte.

Lexical Source Model Extraction (LSME)

Murphi & Notkin [11] utilizam uma linguagem de especificação própria baseada em expressões regulares e características. Estas características são utilizadas para facilitar a descrição do modelo a ser extraído, como por exemplo definindo uma hierarquia entre as expressões regulares. As expressões são definidas em termos de tokens e permitem a colocação de código a ser executado quando a expressão for amarrada a um trecho do artefato. O resultado final da aplicação da técnica deve ser analisado manualmente ou através da utilização de heurísticas pré-definidas.

1.4.2. Análise Sintática

Esta abordagem é baseada no reconhecimento das estruturas sintáticas existentes nos artefatos e na criação das árvores de sintaxe abstratas (AST) correspondentes. Necessitam da especificação formal da gramática utilizada. Suas principais desvantagens são a necessidade da correção e existência de todos os artefatos envolvidos, como por exemplo dos arquivos de cabeçalho, e a falta de flexibilidade, ou seja, a necessidade de um parser para cada linguagem utilizada.

Program Slivers

Gupta [12] utiliza o conceito de Slivers, que são linhas de código que contém algum padrão selecionado, a partir da utilização de uma técnica simples de casamento de padrões. Possui uma baixa granularidade, ou seja, não permite a identificação de construções em um nível de abstração maior.

Metrics for Clone Detection

Kontogiannis [13] procura identificar trechos de código clonado em vários pontos do programa. Aplica um conjunto de métricas em um ponto e tenta encontrar outros pontos que possuem o mesmo valor das métricas. O método utilizado não possui uma boa precisão, necessitando ser refinado manualmente.

1.4.3. Análise Dinâmica de Sistemas

Esta categoria agrupa as propostas que utilizam métodos dinâmicos para obter informações de rastreamento, ou seja, através de informações coletadas a partir da execução do sistema. As principais desvantagens destes métodos são: não podem ser utilizados durante todo o ciclo de vida do sistema, tendo seu uso possível apenas a partir do momento que o sistema entra em operação; não capturam as decisões tomadas pelo desenvolvedor e nem o seu contexto (ou razão pela qual foi tomada); não abrangem todo o sistema pois os cenários de utilização do sistema não cobrem todas as possibilidades; e o reuso das informações é muito reduzido ou inexistente. O uso das informações oriundas destas técnicas se restringe mais ao entendimento do sistema com vistas a sua manutenção do que ao processo de desenvolvimento propriamente dito.

Scenario-Driven Approach to Traceability

Egyed [2] apresenta um processo interativo para gerar informações de rastreamento a partir da observação de cenários de testes sendo executados por um sistema de software. O processo prevê inicialmente a criação de algumas hipóteses sobre rastreamento entre elementos dos modelos e o sistema, entre elementos dos modelos e cenários e entre elementos dos modelos. Em seguida, através da execução do sistema seguindo os diversos cenários, são coletadas informações sobre quais e quantas vezes as linhas de código são executadas em cada cenário. Por fim, estes dados são analisados e são feitas generalizações e refinamentos sobre o conjunto de informações.

URCA - A Use-Case Driven Method of Architecture Recovery for Program Understanding and Reuse Reengineering

O método proposto por Bojic & Velasevic [14] utiliza código fonte orientado a objetos, um conjunto de casos de teste e os casos de uso para agrupar as classes em módulos e obter o relacionamento entre os casos de uso e suas realizações, ou seja, os módulos e classes que implementam suas funcionalidades. O objetivo principal é facilitar a manutenção através da localização do código que implementa determinada funcionalidade.

ISVis / MORALE Project

Jerding & Rugaber [15] utilizam a análise estática para instrumentar o

código fonte. Em seguida o código instrumentado é executado usando uma série de cenários de uso e gerando eventos de rastreamento que são apresentados através de uma ferramenta de visualização, mostrando os atores e as interações envolvendo cada cenário. Cria ainda os modelos de arquitetura do sistema analisado e permite realizar diversas interações para refinar os modelos. A ênfase da proposta é no entendimento do programa para tratar a evolução de sistemas.

Object-Oriented Program Tracing and Visualization

Lange & Nakamura [16] propõem uma técnica que combina informações extraídas de maneira estática com informações extraídas da execução de programas. A captura das informações dinâmicas é realizada através de técnicas de debug. A ênfase da técnica é na visualização destas informações e não em seu uso durante o processo de evolução do sistema.

Recovering High Levels OO from Static and Dynamic Analysis

Richner & Ducasse [17] propõem um método que utiliza a análise estática e dinâmica para obter modelos de alto nível de sistemas a partir do código fonte Smalltalk. Este método não obtém os requisitos sendo ainda aplicável apenas a sistemas desenvolvidos em Smalltalk.

1.4.4. Reestruturação de Programas

Existem diversas propostas de técnicas de engenharia reversa para reestruturar sistemas existentes, objetivando primordialmente o entendimento dos sistemas. Apesar da grande maioria destas propostas não abordarem sua utilização no rastreamento dos elementos inferidos a partir das suas aplicações, as técnicas podem ser utilizadas nas fases finais do desenvolvimento para obter informações de rastreamento. Desta maneira, suas desvantagens são as mesmas da análise dinâmica.

DARE – Domain-Augmented Reengineering

A abordagem proposta por DeBaud [18] utiliza as informações sobre o domínio do sistema para dirigir o processo de entendimento do programa e de reengenharia. A partir da construção do modelo do domínio é realizado o mapeamento entre os conceitos do domínio e sua realização em um sistema legado. Estas informações podem ser então utilizadas no entendimento de outros sistemas, através da utilização de técnicas de reconhecimento de planos para

reconhecer a realização dos conceitos no código do novo sistema, e na reengenharia, através da definição de um novo mapeamento entre os conceitos e a nova plataforma.

1.4.4.1. Identificação de Tipos Abstratos de Dados (ADT)

Este conjunto de técnicas é utilizado na identificação de tipos abstratos de dados (objetos no contexto da orientação a objetos), a partir de código fonte procedural. Girard et al. [19] apresentam uma comparação de diversas técnicas para obtenção de ADT. As principais abordagens utilizadas nesta identificação são as seguintes:

- Dirigido a Funções: identifica os objetos a partir das chamadas de funções existentes procurando minimizar o acoplamento entre os objetos
- Dirigido a Dados: utiliza as estruturas de dados do sistema como ponto inicial para definição dos objetos. As rotinas que acessam as estruturas de dados se tornam métodos dos objetos
- Dirigido a Objetos: utiliza o conhecimento do desenvolvedor sobre o domínio da aplicação

Scenarios for the Identification of Objects in Legacy Systems

Wiggerts et al. [20] propõem uma forma incremental para realizar a identificação de objetos do negócio em código legado, através da utilização das 3 abordagens (funções, dados e objetos).

OBAD

Yeh et al. [21] propõem uma ferramenta para identificação de objetos a partir da AST do programa. As estruturas de dados e sub-rotinas são os nós e as referências existentes em cada sub-rotina para as estruturas são os arcos. Os candidatos a objetos são os conjuntos de componentes interconectados.

CANTO

Antoniol et al. [22] utilizam as métricas de sistemas orientados a objetos para identificar os candidatos a objetos. Agrupa as funções e dados em objetos de maneira a conseguir um melhor valor para as métricas.

Type Inference

Liwu [23] identifica os objetos através da procura de variáveis que

compartilham uma mesma representação. Está técnica analisa as atribuições existentes entre as diversas variáveis e infere o tipo de cada uma pelo relacionamento que a mesma possui com outras, como por exemplo no trecho de código a seguir é inferido que as variáveis *a* e *x* possuem o mesmo tipo:

```
int a , b ;
Funcao f(int x , int y) {
  x = a;
}
```

1.4.4.2. Decomposição

Esta abordagem procura obter a decomposição do sistema em um conjunto de módulos estruturados. Esta decomposição pode ser obtida pelo particionamento do sistema em seus componentes ou pelo agrupamento dos componentes em unidades maiores.

Slicing

São técnicas utilizadas para extrair as partes do programa (slices) relacionadas com alguma determinada computação. As partes selecionadas do programa consistem das partes que podem, direta ou indiretamente, afetar os valores computados para um determinado ponto de interesse (slicing criterion). De acordo com este critério, podemos ter o Static Slicing [24]; que localiza as porções do programa que preservam o comportamento de uma determinada variável para todos os valores possíveis de entrada; Dynamic Slicing [24], que leva em consideração apenas um subconjunto de entradas do programa; Architectural Slicing [25], utilizado para extrair componentes arquitetônicos reusáveis a partir de especificações formais de arquiteturas de sistemas existentes; e Interprocedural Slicing [26], que procura dividir o sistema analisando trechos que implementam determinado componente de interface com o usuário.

Clustering [27][28][29]

São técnicas utilizadas para agrupar os componentes de um sistema baseado nos relacionamentos ou similaridades existentes entre eles. Estas técnicas podem utilizar teoria dos grafos, onde são utilizadas representações em grafo para o programa; algoritmos de otimização, onde é criada uma partição inicial do sistema e são realizadas tentativas de melhora da solução inicial através de adaptações

interativas; critérios estruturais para agrupar os componentes em subsistemas minimizando o acoplamento entre eles e maximizando o acoplamento dos componentes de um mesmo subsistema; abstração de dados, que utiliza os dados do mesmo tipo como critério para agrupar componentes.

Architectural Query Language (AQL)

Sartipi et al. [30] apresentam uma técnica que procura obter o arranjo ótimo das estruturas do sistema, a partir da AST do programa fonte, que atendam a um conjunto de hipóteses feitas pelo usuário sobre a arquitetura do sistema,. Estas hipóteses são descritas utilizando uma linguagem de consulta própria (AQL - Architectural Query Language) e o processo de otimização utiliza técnicas de Data Mining e Clustering.

Concept Analysis

O método proposto por Siff & Reps [31] procura agrupar artefatos computando uma função que mede a similaridade existente entre pares de artefatos. O método prevê a construção de um grafo de conceitos (coleção máxima de entidades similares) agrupando estes artefatos em unidades maiores. Estas unidades maiores são consideradas como uma abstração do grupo, inferida através da semelhança entre seus membros. Permite tratar tanto as informações positivas quanto as negativas, ou seja, tanto agrupar por possuírem alguma característica comum como também por não a possuírem.

Informal Information

Biggerstaff et al. [32] utilizam uma técnica baseada nas informações existentes em identificadores e comentários como heurística na escolha entre várias alternativas de decomposição de um sistema.

Dominance Analysis

A técnica proposta por Girard & Koschke [33] constrói o grafo de chamadas de sub-rotinas identificando a hierarquia de componentes arquitetônicos (sub-rotinas de suporte, módulos e sub-sistemas) através da aplicação da análise de dominância no grafo.

1.4.5. Técnicas Baseadas em Conhecimento

Este conjunto de técnicas utiliza uma biblioteca de planos de programação pré-definidos que são comparados com porções do código, com o objetivo de

identificar os planos que podem ser utilizados para explicar o trecho selecionado. Cada plano possui um padrão de comparação e um conceito associado. O padrão é uma combinação de componentes, itens da linguagem ou sub-planos que precisam ser reconhecidos, e restrições, relacionamentos que precisam existir entre os componentes para que o plano seja reconhecido. O conceito associado ao plano descreve as características inferidas sobre o trecho de código a partir da identificação do plano.

Estas técnicas tipicamente requerem uma extensa base de dados de planos e garantem apenas a recuperação de modelos parciais. Outros problemas destas técnicas são a existência de vários padrões de código que implementam o mesmo plano e um problema de escala para sistemas grandes.

Proper Decomposition

Esta técnica foi proposta por Hartman [34] para realizar o reconhecimento de estruturas de controle. Inicialmente o programa é transformado em uma representação independente de linguagem, na forma de um grafo hierárquico de controle de fluxo / fluxo de dados. Em seguida este grafo é decomposto hierarquicamente em *propers*, grafos simples de entrada / saída. As estruturas de controle (planos) são descritas utilizando o mesmo conceito de propers associados a um conjunto de restrições.

Constraint-Based Program Plan Matching

A técnica proposta por Quilici et al. [35] utiliza componentes e restrições na definição dos planos. Os planos são identificados através da utilização de algoritmos de satisfação de restrições.

Clichés

Clichés são planos utilizados para descrever grupos de expressões do programa relacionados a um objetivo do usuário. São compostos por algoritmos e pelas estruturas de dados comuns que são acessadas por eles. A técnica proposta por Fiutem et al. [36] utiliza algoritmos de casamento de padrões para localizar partes do código que correspondem aos clichés.

1.4.6. Métodos Formais

As técnicas baseadas em métodos formais tornam as especificações mais precisas e desta maneira facilitam o armazenamento e processamento das

informações de rastreamento. O problema inerente a sua utilização prática deriva da falta de conhecimento dos desenvolvedores na utilização do formalismo para especificar os artefatos.

TOOR: Traceability of Object Oriented Requirements

A proposta apresentada por Pinheiro & Goguem [37] utiliza uma abordagem formal para tratar o rastreamento de requisitos. Armazena as ligações através de relacionamentos entre elementos. Os relacionamentos podem ser de vários tipos diferentes, podendo ainda ser criados novos relacionamentos pelos usuários da ferramenta (TOOL). Desta forma, cada elemento está relacionado de diversas maneiras diferentes com outros elementos, criando assim uma rede de dependências, o que fornece uma maior flexibilidade na sua definição e no seu conseqüente uso. O usuário pode navegar ao longo dos relacionamentos, bem como executar pesquisas utilizando expressões regulares. Esta abordagem é melhor aplicada para rastrear requisitos ao seu contexto, ignorando a questão do rastreamento de design.

Software Reflexion Model

A abordagem proposta por Murphy et al. [38] [39] baseia-se na construção de um modelo de alto nível por um especialista e na subseqüente comparação deste modelo com os modelos obtidos através da análise estática do código fonte, obtendo pontos de concordância e de discrepância entre os modelos de alto nível e o código fonte. O objetivo deste método é verificar a consistência entre os modelos e o código fonte.

1.4.7. Técnicas Manuais

Técnicas de rastreamento manuais não são aplicáveis em sistemas grandes e complexos, por necessitarem de uma grande quantidade de recursos tanto para obter as informações quanto para mantê-las atualizadas. A maior parte dos métodos propostos aborda fases ou aspectos específicos do desenvolvimento, diminuindo desta forma os recursos necessários, mas restringindo sua aplicabilidade e abrangência.

Extended Traceability

Haumer et al. [40] apresentam uma abordagem baseada no registro do relacionamento entre a utilização concreta do sistema pelo usuário (através de

vídeo, som, etc.) com os modelos conceituais do sistema. Estes relacionamentos, que o autor chama de *Extended Traceability*, são utilizados para melhorar a eficiência das revisões.

System for Automated Validation of Embedded Software

Lingamarla et al. [41] apresentam a arquitetura de um sistema que utiliza um banco de dados relacional para armazenar as informações de rastreamento entre requisitos, testes e configurações de controladores embarcados. Seu objetivo principal é gerenciar os requisitos e criar produtos que serão colocados em operação em múltiplas configurações, com requisitos funcionais similares, mas diferenças na implantação junto ao ambiente do usuário. Este método não rastreia os produtos intermediários do desenvolvimento, apenas os requisitos, testes e módulos do sistema. Em conseqüência, sua aplicação fica restrita as fases de testes, distribuição e implantação do sistema.

AMBOLS

A proposta de Liu et al. [42] adota a semiótica organizacional como fundamento teórico: vê a organização como um sistema onde sinais são criados para permitir a comunicação necessária ao negócio e as regras do negócio governam o comportamento e as operações. Propõem a obtenção dos requisitos do sistema através da observação do ambiente e do sistema como uma caixa preta. Utiliza a análise dinâmica para realizar o mapeamento entre os requisitos e as funções do sistema.

1.4.8. Paradigma Transformacional

Estas técnicas utilizam um sistema transformacional como ferramenta para tratar gerar e manter as informações de rastreamento.

Design Maintenance System (DMS)

O sistema proposto por Baxter et al. [43] [44] [45] utiliza linguagens específicas de domínio (DSL) para escrever uma especificação formal do sistema, aplicando a seguir um conjunto de transformações para obter o código correspondente. Procura armazenar as decisões de modelagem como transformações capturadas automaticamente. Enfatiza a questão de que o desenvolvedor deve trabalhar nos modelos e gerar a implementação

automaticamente, como forma de manter os diversos artefatos atualizados e sincronizados.

1.5. Organização da Tese

No capítulo 2 apresentamos a Máquina Draco-PUC que irá fornecer o ferramental básico necessário ao estudo do tema proposto nesta tese. Serão apresentados o seu uso, extensões e ferramentas desenvolvidas para dar suporte a esta tese.

O capítulo 3 fornece uma visão geral de cenários abordando sua conceituação, descrição, os aspectos envolvidos em sua evolução e questões referentes ao rastreamento.

No capítulo 4 apresentamos nossa proposta para um mecanismo de rastreamento baseado em transformações. Inicialmente, definimos a notação interna utilizada para descrever os artefatos e, em seguida, apresentamos suas atividades.

O capítulo 5 apresenta um sistema de rastreamento que utiliza o mecanismo proposto nesta tese e voltado para o desenvolvimento baseado em cenários. É apresentada, inicialmente, uma descrição do sistema seguido por um protótipo de ferramenta. Finalizando o capítulo, é apresentado um estudo de caso utilizado para exemplificar o uso do sistema e verificar a viabilidade de implementação do mecanismo proposto.

Nossas conclusões são apresentadas no capítulo 6, onde ressaltamos as contribuições da tese e apresentamos uma comparação com outras propostas de rastreamento existentes na literatura. Por último, acenamos com alguns trabalhos futuros que podem ser realizados para dar prosseguimento a nossa tese.

Os apêndices contêm uma descrição completa dos diversos artefatos criados na definição do mecanismo de rastreamento e na implementação do sistema correspondente.