Universidade Federal de Juiz de Fora
Instituto de Ciências Exatas
Bacharelado em Ciência da Computação

EDX: Uma Abordagem de Apoio ao Controle de Mudanças de Documentos XML

Plínio Antunes Garcia

JUIZ DE FORA
JULHO, 2012

EDX: Uma Abordagem de Apoio ao Controle de Mudanças de Documentos XML

Plínio Antunes Garcia

Universidade Federal de Juiz de Fora

Instituto de Ciências Exatas

Departamento de Ciência da Computação

Bacharelado em Ciência da Computação

Orientador: Alessandreia Marta de Oliveira

JUIZ DE FORA

JULHO, 2012

EDX: Uma Abordagem de Apoio ao Controle de Mudanças de Documentos XML

Plínio Antunes Garcia

MONOGRAFIA SUBMETIDA AO CORPO DOCENTE DO INSTITUTO DE CIÊNCIAS EXATAS DA UNIVERSIDADE FEDERAL DE JUIZ DE FORA, COMO PARTE INTEGRANTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE BACHAREL EM CIÊNCIA DA COMPUTAÇÃO.

Aprovada por:

Alessandreia Marta de Oliveira
M. Sc.

Luciana Conceição Dias Campos
D. Sc

Jairo Francisco de Souza
D. Sc.

JUIZ DE FORA

12 DE JULHO, 2012

À minha família.

À Flávia.

Aos meus amigos.

Resumo

O extensivo uso de documentos XML em projetos de desenvolvimento de software estimula o estudo de métodos capazes de controlar a evolução de tais documentos. Contudo, as particularidades deste tipo de documento, como a sua organização hierárquica, tornam a sua análise uma tarefa diferente das abordagens que utilizam comparação textual. Algumas linhas de pesquisas resolvem esta situação explorando as diferenças sintáticas entre versões de um documento XML. Porém, observa-se que é possível extrair informações implícitas entre as mudanças de um documento que são imperceptíveis a análises estruturais e textuais. Diante disso, uma alternativa para controlar as mudanças ocorridas em documentos XML é adaptar técnicas de Gerência de Configuração de Software no contexto do controle de mudanças. Para isto, este trabalho apresenta uma proposta de detecção e análise de mudanças semânticas entre as versões do documento. Esta abordagem é aplicada no protótipo EDX, que utiliza Prolog para realizar inferência de dados em documentos XML através de uma interface gráfica que permite a construção de regras em alto nível.

Palavras-chave: Controle de Mudanças, Inferência, XML, Prolog.

Agradecimentos

Agradeço à minha família por ter me ensinado que a educação é um patrimônio para toda a vida e por ter me dado o suporte necessário para seguir neste caminho.

Aos meus pais, Sônia e Antonio, pelo amor incondicional e pela dedicação empregada no meu crescimento pessoal e profissional. Ao meu irmão, Marcondes, pelos bons exemplos aos quais me baseio. Ao meu amor, Flávia, por estar sempre ao meu lado, por me ajudar a perseverar no curso nos momentos mais difíceis, pelo companheirismo e inspiração.

Aos meus colegas do Grupo de Educação Tutorial da Computação (GETComp UFJF), em especial à Carolina Cunha, Danúbia Dias, Guilherme Martins, Lenita Ambrósio pelas revisões e sugestões no trabalho, e ao colega de pesquisa Pedro Gazzola.

À professora e amiga, Alessandreia, pela motivação a cada aula, por me apresentar a área escolhida para este trabalho, pela confiança e pela dedicação na orientação desta pesquisa.

Agradeço ao Hugo Rezende e ao Thiago Valério por compreenderem minhas ausências no trabalho durante o período de estudo.

Aos professores do Departamento de Ciência da Computação pelos seus ensinamentos e aos funcionários do curso, que durante esses anos, contribuíram de algum modo para a minha evolução acadêmica.

"Não há nada permanente, a não ser a mudança".

Heraclito de Éfeso

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Abreviações

DCC	Departamento de Ciência da Computação
EDX	Evolução de Documentos XML
GCS	Gerência de Configuração de Software
UFJF	Universidade Federal de Juiz de Fora
XML	Extensible Markup Language

1 Introdução

Observa-se no cenário de desenvolvimento de software a grande utilização de dados semi-estrutrados, normalmente representados por documentos XML (*Extensible Markup Language*) (BRAY et al, 2008). Os documentos XML apresentam estrutura hierárquica e marcadores que são definidos pelo próprio desenvolvedor e, desta forma, permitem uma flexibilidade considerável na representação de dados.

Tal flexibilidade torna os documentos XML passíveis de alterações frequentes. Isto ocorre à medida que os projetos, cujos documentos fazem parte, evoluem ao longo do ciclo de desenvolvimento. As constantes alterações e as particularidades de um documento XML dificultam o gerenciamento dos projetos e o acompanhamento da sua evolução.

Com o objetivo de elaborar métodos e técnicas que auxiliam na gerência de projetos, a Gerência de Configuração de Software (GCS) se apresenta como uma atividade abrangente que é aplicada em todo o processo de Engenharia de Software e concentra pesquisas em soluções para este tipo de problema (PRESSMAN, 2002).

Um dos subsistemas que compõem a GCS é o controle de mudanças, que combina procedimentos humanos e ferramentas automatizadas com o objetivo de gerenciar as mudanças ocorridas durante o desenvolvimento de software. O controle de mudanças procura acompanhar uma alteração desde o momento em que a necessidade de mudança é reconhecida e solicitada, até a disponibilização da nova versão do item modificado. Durante todo o processo, cada alteração é documentada a fim de aumentar a compreensão da evolução de determinado item, e do projeto como um todo.

Algumas abordagens apresentam alternativas para controlar alterações em documentos XML baseadas em mudanças sintáticas (COBENA et al, 2002; ZHANG et al, 2004; ZHAO et al, 2004; ZHAO e SOURAV, 2005). Contudo, estas abordagens se limitam à estrutura do documento e não exploram informações implícitas nas alterações.

Para isto, a abordagem apresentada neste trabalho propõe a detecção e a análise de mudanças semânticas, visando auxiliar na compreensão das razões da modificação. Diante disso, o presente trabalho apresenta o protótipo EDX (Evolução de Documentos

XML), cuja funcionalidade principal é extrair informações implícitas presentes em versões de um documento XML. Com isto, a abordagem busca auxiliar o controle de mudanças em sistemas que utilizam documentos XML, adaptando técnicas e conceitos da GCS para adequar às particularidades desse tipo de documento. Tais informações são recuperadas através de inferências realizadas ao analisar as mudanças ocorridas no documento.

O processo de inferência deste trabalho utiliza a linguagem Prolog e propõe um módulo que semiautomatiza o processo de criação das regras de inferência sem a necessidade do conhecimento da linguagem. Para atingir este objetivo, o EDX apresenta uma interface gráfica para a construção de consultas em alto nível.

1.1 Motivação

É crescente a busca por meios de adaptar técnicas de GCS às características de documentos XML. Uma das abordagens para auxiliar o controle de mudanças neste tipo de documento, adotada neste trabalho, utiliza a linguagem Prolog para realizar inferências. Nesta abordagem, os documentos XML necessitam ser convertidos em fatos Prolog para que seja criada a base de conhecimento necessária para a inferência. Diante disso, as consultas devem ser feitas na mesma linguagem e, embora grande parte dos usuários interessados em realizar inferências em documentos XML seja de áreas tecnológicas, é possível encontrar dificuldades devido a pouca familiaridade com a linguagem Prolog.

Como uma alternativa para solucionar o problema mencionado, observa-se a necessidade de uma interface gráfica cuja usabilidade seja suficiente para a construção de consultas a partir de seleção de opções em alto nível, com a finalidade de gerar consultas em Prolog de forma transparente ao usuário. Assim, elimina-se a necessidade do usuário ser familiarizado com a linguagem usada na inferência.

1.2 Justificativa

A particularidade estrutural dos documentos XML exige abordagens específicas para a manipulação de dados semiestruturados. Algumas ferramentas tradicionais da GCS tratam os arquivos como documentos de texto, tornando-se inadequadas para trabalhar com

documentos XML.

Existem ferramentas específicas para documentos XML que extraem informações entre versões consecutivas de documentos através de cálculos de diferenças sintáticas. Estas abordagens se restringem à obtenção de mudanças estruturais.

É possível aproveitar as características específicas de documentos XML para extrair informações implícitas nas mudanças ocorridas. Isto pode ser feito aplicando uma ótica semântica nas diferenças encontradas entre versões consecutivas de um documento. No contexto do controle de mudanças, é importante conhecer o significado das alterações e analisar as razões que levaram a elas.

1.3 Objetivos

O objetivo geral deste trabalho é fornecer informações sobre as modificações detectadas entre duas versões de um documento XML.

Ainda que grande parte dos usuários que lidam com documentos XML sejam de áreas tecnológicas e possivelmente conheçam Prolog, este fato limita a abrangência de usuários. Diante disso, este trabalho propõe uma interface gráfica que permite a construção de consultas em alto nível.

A interface de construção de consultas possibilita ao usuário combinar elementos, operadores e outras consultas aninhadas. Este processo é feito através de menus de seleção, ou seja, permite ao usuário a construção de consultas sem a necessidade de escrever as regras em Prolog, pois esta etapa é realizada de forma transparente ao usuário.

1.4 Organização do Trabalho

Este trabalho está organizado em quatro capítulos além deste capítulo de introdução. No Capítulo 2 são apresentados os conceitos de documentos XML, gerência de configuração e controle de mudanças de documentos XML. Além de descrever a atuação da linguagem Prolog na realização da inferência.

O Capítulo 3 se caracteriza por explorar a temática de controle de mudança de documentos XML apresentando alguns trabalhos relacionados, bem como a abordagem

proposta nesta pesquisa. Este capítulo apresenta também um quadro comparativo entre os trabalhos estudados e a abordagem proposta.

No Capítulo 4 é descrito um estudo de caso que explica a utilização da interface gráfica projetada para a construção de regras, além dos resultados obtidos com as consultas geradas.

O Capítulo 5 finaliza este trabalho apresentando as considerações finais sobre a pesquisa, propostas de trabalhos futuros e otimizações da abordagem proposta.

2 Uso de Inferência na Evolução de Documentos XML

Este capítulo tem como foco a gerência de documentos XML e se inicia com a contextualização do tema na seção 2.1. Na seção seguinte a gerência de configuração de documentos XML é abordada. Em seguida, a seção 2.3 explora o controle de mudanças e o papel da inferência em documentos XML. Na seção 2.4 é apresentada uma explanação de alguns conceitos da linguagem Prolog e sua atuação na inferência. Por fim, encerrando o capítulo tem-se a conclusão na seção 2.5.

2.1 Documentos XML

Grande parte dos dados disponíveis eletronicamente não apresentam uma estrutura ou um esquema previamente definidos. Por exemplo, dados da *Web* que possuem estruturas heterogêneas e apresentam variados conteúdos, desde simples textos até registros que compõem informações estruturadas de um sistema. Dados com tais características são chamados semiestruturados. São dados que além da informação também possuem a própria estrutura explícita no documento, sendo assim auto-descritivos (MELLO et al, 2001).

A linguagem XML é muito utilizada na descrição e representação de dados semiestruturados. Documentos XML são muito utilizados para a publicação de informações na *Web*, configurações e integração entre bancos de dados e aplicações de diferentes plataformas.

O sucesso dessa linguagem é justificado pela sua flexibilidade na representação da informação. O conteúdo de um documento XML é definido pelo desenvolvedor desde as marcações, também conhecidas como *tags*, até seu conteúdo em uma única estrutura, normalmente, sob a forma de árvore. Em uma instância XML, por se tratar de dados semiestruturados, é necessária uma análise para extrair seu esquema de representação e

conhecer sua estrutura.

Instâncias de dados semiestruturados podem evoluir e sofrer modificações ao longo do tempo, considerando sua flexibilidade estrutural e sua heterogeneidade, gerenciar essas instâncias é um problema que vem sendo estudado.

Um documento XML pode sofrer alterações não somente em informações de conteúdo, mas também nas marcações que representam sua estrutura. Novos elementos podem ser introduzidos ou removidos, atributos podem ser modificados e atribuídos à diferentes elementos. Tais mudanças podem levar os dados do XML a um estado inconsistente ou incompatível com esquemas e definições anteriores. Por isso a necessidade de se acompanhar e controlar essas alterações.

2.2 Gerência de Configuração de Documentos XML

Em um ambiente de desenvolvimento de software é comum a geração de inúmeros artefatos ao longo do ciclo de vida de um projeto, tais como arquivos de código-fonte e documentações. Estes itens são constantemente acessados e modificados.

Neste contexto, surge a GCS, que é uma disciplina da Engenharia de Software cujo objetivo geral é prover um meio de controlar e gerenciar artefatos de projetos de software. Isto permite a evolução do projeto de forma organizada e consistente (ESTUBLIER, 2000).

Sob a perspectiva de desenvolvimento, a GCS é composta de três subsistemas. O controle de versões que combina procedimentos e ferramentas para identificar, armazenar e controlar os artefatos assim como a sua evolução ao longo do ciclo de vida do projeto para que ela ocorra de forma organizada. As ferramentas de controle de versões possibilitam que os envolvidos em um projeto trabalhem simultaneamente sobre os mesmos itens. As principais funcionalidades destas ferramentas são o armazenamento e recuperação de itens, identificação de conflitos e mesclagem de revisões. Desta forma, o controle de versões garante, em um ambiente de desenvolvimento de software, a evolução do projeto de forma distribuída e menos propenso a falhas.

O controle de mudanças que é responsável por acompanhar as alterações desde o momento em que elas são solicitadas até o momento em que elas forem disponibilizadas,

com o objetivo de organizar e garantir que tais mudanças ocorram de forma controlada permitindo-se conhecer o que foi alterado, por quem e quando a mudança ocorreu. As ferramentas de controle de mudanças documentam todos os dados gerados nas modificações de um projeto, a fim de disponibilizar informações aos interessados autorizados.

O gerenciamento de construção que é responsável pela disponibilização das versões executáveis de um software para os usuários, automatizando a transformação dos artefatos que compõem o projeto em distribuições executáveis (MURTA, 2006). Por exemplo, em um projeto de software desenvolvido na linguagem Java, é necessário o processo de construção, ou empacotamento, para converter os códigos fonte em um arquivo ".jar". As ferramentas de gerenciamento de construção permitem a automatização e a customização deste processo.

Tratando um documento XML como um Item de Configuração (IC), ou seja, um elemento de informação cuja evolução é passível de rastreamento (PRESSMAN, 2002), a GCS é uma alternativa para contribuir com o gerenciamento deste tipo de dados.

Diante deste cenário, surgem estudos com a proposta de adaptar técnicas de gerência de configuração para serem aplicadas a dados semiestruturados. Tais técnicas necessitam ser adaptadas pois existem particularidades no documento XML que demandam um tratamento específico.

Por se tratar de uma linguagem de marcação, ou seja, com uma estrutura própria definida pelo desenvolvedor, documentos XML podem ser inseridos entre arquivos de texto, necessitando de um tratamento diferenciado na resolução de problemas oriundos dessas particularidades e na manipulação dos dados contidos em tais arquivos (SILVA, 2011).

As ferramentas tradicionais de controle de versões analisam o documento como texto e utilizam comparações de caracteres para realizar a detecção de mudanças entre as versões analisadas. No contexto de dados semiestruturados, o controle de versões deve levar em consideração a estrutura peculiar de documentos XML embora nem sempre a estrutura da instância se apresenta de forma explícita.

O XPerseus (SILVA, 2011) é uma ferramenta de controle de versões desenvolvida para trabalhar especificamente com documentos XML e além disso, possui um módulo de

inferências cujo objetivo é apoiar o controle de mudanças neste tipo de dados (GAZZOLA, 2011).

2.3 Controle de Mudanças e Inferência em Documentos XML

Em grandes projetos de software, mudanças realizadas sem acompanhamento podem levar o projeto rapidamente ao caos. O controle de mudanças mescla procedimentos humanos e ferramentas que automatizam o controle das modificações de um item (PRESSMAN, 2002).

O controle de mudanças tem como função gerenciar os ICs de forma sistemática, coletando e armazenando as informações resultantes de alterações realizadas no artefato (OLIVEIRA, 2011). Espera-se de um sistema de controle de mudanças que ele forneça informações sobre a evolução de uma configuração para que os interessados possam analisar e compreender modificações realizadas em um projeto.

No contexto de dados semiestruturados, especificamente documentos XML, o controle de modificações deve fornecer apoio para a gerência das instâncias XML e informações sobre a evolução do item.

As principais ferramentas de GCS possuem sistemas de controle de mudanças com foco no histórico das modificações e, a princípio, não diferenciam documentos comuns de documentos semiestruturados.

Os sistemas mais comuns de controle de mudanças se concentram em registrar os dados que fornecem as informações de quem efetuou a solicitação ou a alteração propriamente dita, quando ela ocorreu e o que foi alterado. Contudo, é possível extrair resultados mais expressivos para o controle de mudanças de documentos XML aproveitando-se a organização hierárquica deste tipo de documento e as particularidades de uma linguagem de marcação.

Documentos XML, por serem auto-descritivos, carregam consigo informações úteis na compreensão das modificações ocorridas em um projeto. Contudo, são necessários métodos especializados em extrair conhecimento de dados semiestruturados e isso não

ocorre nos sistemas que utilizam métodos baseados em pesquisa textual.

O acesso ao conhecimento presente nos dados coletados durante a evolução de documentos XML permite obter informações de maior valor semântico implícito nas alterações dos documentos. Com isso, existem muitas pesquisas que visam automatizar a análise e o registro destas informações a fim de obter resultados mais elaborados no contexto de negócio onde se encontra a instância XML.

Entre os sistemas que realizam consultas em documentos XML, a maioria realiza as buscas baseadas na estrutura em árvore do documento, percorrendo desde o nodo raiz até o elemento mais interno descrito na consulta. Desta forma, a consulta leva em consideração somente os dados explícitos textualmente no documento e ignora as informações implícitas que poderiam ser extraídas destes dados (LIMA et al, 2011).

Por exemplo, um elemento "salário" que pertence ao elemento "funcionário" em uma estrutura XML, ilustrada na Listagem 2.1, tem seu valor alterado e é gerada uma segunda versão deste documento, representada na Listagem 2.2. Um sistema tradicional detecta que houve a alteração do valor de "salário", mas esta análise é superficial no que diz respeito à semântica. Um sistema mais elaborado poderia inferir que o salário aumentou ou diminuiu e explorar as razões da mudança através da análise de outros elementos do documento. Este exemplo simples ilustra como a inferência pode trazer informações de alto nível para o controle de modificações.

Listagem 2.1: Elemento funcionário em uma primeira versão

```
1  <funcionario>
2      <matricula>00044</matricula>
3      <nome>Antonio</nome>
4      <salario>3000</salario>
5      <cargaHoraria>36</cargaHoraria>
6      <cargo>Vendedor</cargo>
7      <depto>Vendas</depto>
8      <filial>Niteroi</filial>
9  </funcionario>
```

Listagem 2.2: Elemento funcionário em uma versão modificada

```
1  <funcionario>
2      <matricula>00044</matricula>
3      <nome>Antonio</nome>
4      <salario>3900</salario>
5      <cargaHoraria>36</cargaHoraria>
6      <cargo>Vendedor</cargo>
7      <depto>Vendas</depto>
8      <filial>Niteroi</filial>
9  </funcionario>
```

Um mecanismo que possa automatizar a busca e a recuperação de informações ricas em semântica é essencial em organizações que manipulam um grande volume de documentos, pois reduz consideravelmente a complexidade de tarefas neste cenário (FONTES, 2011).

Diante disso, existe a necessidade de extrair o conhecimento presente nos resultados, mas que não se apresenta de forma explícita nos dados retornados de uma consulta. A abordagem apresentada por GAZZOLA (2011) propõe a obtenção de resultados através da análise dos relacionamentos entre os dados de um documento semiestruturado, buscando pelo conhecimento que não é representado na forma de texto.

Para realizar a inferência de dados é necessário fornecer um conjunto de regras ao sistema para que ele seja capaz de processar estas regras e inferir o conhecimento implícito nos documentos XML. Tais regras analisam um grupo de declarações e fatos traduzidos para uma forma compatível com o funcionamento do sistema e que forneçam a base de conhecimento sobre o contexto em análise. Com os fatos extraídos das instâncias e as regras inerentes ao contexto do documento é possível que um raciocinador possa fornecer resultados expressivos dos documentos analisados.

De posse do conhecimento extraído de um documento XML, o controle de mudanças ganha informações que vão além do quem, quando e onde ocorreu a mudança, contribuindo para tomadas de decisão acerca do projeto e, obviamente, para o gerenciamento do documento.

2.4 A linguagem Prolog na Realização da Inferência

Prolog é uma linguagem de programação inserida no paradigma de programação declarativa e usa a lógica de predicados para provar teoremas e relações entre objetos. Como uma linguagem de programação lógica, trabalha sobre um subconjunto da lógica de primeira ordem, as Cláusulas de Horn (PALAZZO, 1997).

A linguagem teve como motivação em sua criação o interesse em utilizar a lógica como forma de representar o conhecimento. Uma das suas primeiras aplicações ocorreu na interpretação da linguagem natural (CUNHA et al, 2007).

Através de um programa em Prolog pode-se expressar o conhecimento utilizando cláusulas de dois tipos: fatos e regras. Um fato é uma afirmação verdadeira incondicional, já as regras definem as condições para que uma determinada declaração possa ser considerada verdadeira (PALAZZO, 1997). O mecanismo para a prova de teoremas nesta linguagem consiste em tentativas de satisfazer uma regra baseada nos fatos declarados na base de conhecimento do programa.

Um fato expressa uma sentença verdadeira, ou seja, uma afirmação. Os fatos também podem ser definidos como informações capazes de representar os dados existentes de um contexto (CUNHA et al, 2007). Como por exemplo, a informação "TCorp é uma empresa"pode ser representada pelo fato Prolog na primeira linha da Listagem 2.3. Uma série de fatos Prolog podem descrever os nomes dos funcionários de uma empresa, como exemplificado nas linhas 2 a 4.

Listagem 2.3: Fatos e regra Prolog

```
1 empresa(tcorp).
2 nome(flavia).
3 nome(jose).
4 nome(pedro).
5 funcionario(X,Y) :- empresa(X), nome(Y).
```

As cláusulas, comumente chamadas de regras, de um programa Prolog são compostas de cabeça e corpo, nesta ordem, sendo essas duas partes separadas pelo símbolo ":-"(lê-se "se e somente se"). A cabeça de uma regra pode ser tratada como a conclusão

da cláusula, enquanto o corpo da regra constitui a condição da cláusula. Se a condição representada pelo corpo de uma determinada regra é verdadeira, tem-se como consequência lógica que sua conclusão, ou seja, a cabeça da regra também é verdade (PALAZZO, 1997).

A regra representada anteriormente, na quinta linha da Listagem 2.3, pode ser interpretada como "Para todo X e para todo Y, se X é uma empresa e Y é o nome de um funcionário, então Y é funcionário da empresa X". Regras mais elaboradas podem ser feitas combinando mais condições no corpo da cláusula separadas por vírgulas, representando o operador *AND*, e separadar por ponto e vírgula, representando o operador *OR*. Na sintaxe da linguagem Prolog, argumentos que iniciam com letras maiúsculas são variáveis e, para representar um valor constante, utilizam-se argumentos iniciados por letras minúsculas (PALAZZO, 1997).

A utilização das regras é feita através de consultas, também chamadas de perguntas. Pode-se, por exemplo, perguntar ao programa Prolog quem é funcionário da empresa TCorp utilizando a notação descrita na Listagem 2.4.

Listagem 2.4: Consulta em Prolog

```
1 ?-funcionario(tcorp,Y).
```

Como não há nenhum fato que corresponde a esta pergunta, a linguagem Prolog busca a resposta na regra de mesmo predicado da pergunta "funcionario". A resposta para esta consulta é encontrada unificando as variáveis da regra com os valores presentes na pergunta. No exemplo citado a variável Y seria unificada com os valores "flavia", "jose"e "pedro", da Listagem 2.3, e as condições da regra seriam atendidas pois há um fato que satisfaz a condição empresa(tcorp) e há pelo menos um fato que satisfaz a condição nome(Y) no programa. Logo, a conclusão da regra também é satisfeita.

Baseada no poder de expressar o conhecimento, a linguagem Prolog é uma grande aliada na busca por relações entre os elementos de um documento XML. A utilização dessa linguagem permite a criação de regras baseadas nas informações explícitas, capazes de extrair as informações implícitas nos documentos.

Outras abordagens foram propostas para realizar inferências em documentos XML, algumas com o uso de ontologias. Os trabalhos que utilizam ontologias fazem uso de raciocinadores para realizar a inferência de dados representados por elas (CUNHA

et al, 2007).

Algumas abordagens trabalham com a linguagem de consulta em banco de dados chamada Datalog. Esta linguagem é baseada em Prolog e, como tal, consiste em um conjunto de fatos e regras. Porém, o uso de Datalog para a inferência de dados apresenta algumas desvantagens pois não permite utilizar termos complexos como argumentos de predicado além de restringir o uso de negação e recursividade (LIMA et al, 2011).

A proposta apresentada neste trabalho faz uso de Prolog para a realização da inferência, uma vez que, é continuidade do projeto XPerseus que conta com um módulo de inferência baseado na linguagem Prolog.

2.5 Conclusão

A grande variedade de aplicações que utilizam documentos XML e a estrutura peculiar destes documentos geram novos desafios para a engenharia de software. A adaptação de técnicas de gerência de configuração de software se faz necessária para contemplar as particularidades dos documentos XML.

Os sistemas de controle de mudanças tradicionais não levam em consideração a natureza das instâncias XML. Com isso, não entram no campo da extração de informações implícitas nos dados gerados a partir das alterações feitas nos documentos. Diante disso, a inferência em documentos XML visa dar suporte ao controle de mudanças no que diz respeito a agregar valor semântico às informações obtidas neste processo. Uma abordagem utilizada para atingir este objetivo é o uso de inferência através da linguagem Prolog.

3 Controle de Mudança de Documentos XML

Existem trabalhos sobre mineração de dados que apoiam o controle de mudanças de documentos XML. Diferentes abordagens são usadas para identificar as alterações e cada uma delas tem seu próprio conceito sobre a atuação da semântica no controle de mudança de documentos XML.

Diante disso, este capítulo aborda diferentes técnicas de controle de mudanças em documentos XML. Na seção 3.1 é descrita uma análise de trabalhos relacionados ao tema desta pesquisa com foco na utilização da semântica como forma de extrair conhecimento das alterações detectadas. A seção 3.2 detalha a abordagem do EDX passando pelo projeto que deu origem ao protótipo até suas funcionalidades. Na seção 3.3 é apresentado um quadro comparativo resumindo os principais parâmetros da análise dos trabalhos. Por fim, a seção 3.4 apresenta a conclusão.

3.1 Trabalhos Relacionados

Em COBENA et al (2002) é apresentado um algoritmo para detectar diferenças entre versões de documentos XML visando apoiar o controle de mudanças no contexto do Xyleme, um projeto que investiga *data warehouses* dinâmicos para armazenamento de grandes volumes de dados. O algoritmo, chamado de XyDiff, propõe eficiência em termos de velocidade e espaço de memória. Um dos seus objetivos é fornecer informações sobre as mudanças encontradas entre as versões dos documentos.

O XyDiff busca a maior subárvore sem modificações, e a partir desta subárvore, são procurados tanto nos nodos pais quanto nos filhos da mesma subárvore, aqueles que não sofreram modificações, obtendo separadamente os nodos modificados e os não modificados. Para aperfeiçoar o desempenho na comparação entre árvores, o XyDiff faz uso de heurísticas.

Ao realizar a associação entre os elementos de diferentes versões do documento XML em análise, o algoritmo calcula valores *hash*[1] e pesos para fornecer um identificador (XID) a cada nodo.

O algoritmo é dividido em cinco etapas. A primeira etapa consiste em buscar os elementos que possuem um atributo de ID previamente definido no DTD (Document Type Definitions) do documento. O algoritmo usa este atributo para encontrar os nodos correspondentes na versão do documento a ser comparado, e somente os elementos que não possuem um ID serão identificados pelo seu contexto.

Na segunda etapa, o algoritmo calcula o *hash* que será usado como assinatura e o peso de cada nodo, além de ordenar as subárvores pelo peso. O peso é calculado pelo tamanho do conteúdo do nodo e pelo peso dos nodos filhos. As primeiras posições da fila de subárvores serão ocupadas pelos nodos de maior peso e, portanto, serão os primeiros a terem sua equivalência verificada pelo algoritmo.

A terceira etapa consiste na busca por equivalências, respeitando a fila de prioridade criada na etapa anterior. De posse dos *hashs* de cada nodo, o XyDiff faz, primeiramente, a comparação das subárvores com mais diferenças entre si, e depois as que possuem poucas diferenças detectadas. Desta forma, a equivalência é definida através da comparação entre as assinaturas dos nodos.

Na quarta etapa, é feito na árvore um mapeamento *bottom-up* seguido de um *top-down*, procurando nodos cujos pais são equivalentes e que têm o mesmo nome, com o objetivo de evitar a detecção de inserções e deleções desnecessárias.

Na última etapa é calculado o *delta script*. Os resultados são classificados entre os nodos inseridos, removidos ou, caso haja alteração de conteúdo, os nodos são considerados atualizados. Além destas três operações básicas, o XyDiff destaca-se por calcular os nodos que mudaram de lugar, e a operação *move* se junta às operações *insert*, *delete* e *update* no *delta script* gerado.

A abordagem apresentada cumpre seu papel no contexto para o qual foi desenvolvida, trabalha de forma eficiente no que diz respeito ao volume de dados analisados e à velocidade, porém deixa a desejar na qualidade dos resultados. Seu foco consiste na

[1]Sequência de bits gerada por um algoritomo de dispersão

técnica para a detecção das alterações e não utiliza informações de valor semântico para explicar as alterações detectadas.

A proposta de ZHANG et al (2004) é detectar, através de sucessivas comparações entre versões de um documento XML, quais porções do documento foram alteradas e quais se mantiveram iguais, sem a utilização de um XML Schema.

Processos de detecção de mudanças baseados na estrutura ou no esquema do documento não são capazes de realizar a associação entre elementos cuja versão modificada apresenta a mesma informação da versão original, porém, com a disposição diferente dos elementos. Portanto, tais processos não são adequados em documentos que são estruturalmente diferentes e semanticamente iguais.

Ao contrário de outros trabalhos que têm foco na detecção de mudanças estruturais, esta abordagem apresenta um algoritmo para a detecção de mudanças baseada na análise semântica.

Através de exemplos, são mostradas as falhas que podem ocorrer quando uma mudança é atrelada somente à estrutura de um documento XML. Um elemento deve ser considerado alterado, sob a perspectiva de mudança estrutural, somente se seu conteúdo ou valor semântico se manteve o mesmo. Caso contrário a técnica de associar uma mudança a um elemento cuja estrutura foi alterada é falha.

O trabalho propõe um *framework* para a detecção de mudanças baseada em semântica. Os nodos de um documento XML que são semanticamente equivalentes não são considerados como uma mudança e então são associados independentemente do contexto estrutural. Esta associação permite o acompanhamento dos elementos ao longo das versões do documento.

A técnica de detecção de mudança semântica tem como base encontrar um identificador semântico para cada nodo do documento. Elementos semanticamente equivalentes possuirão o mesmo identificador. Basicamente, o identificador semântico é uma expressão regular de consulta usada para distinguir um elemento de outro. Um elemento pode ter estrutura idêntica a outro, mas sob a perspectiva semântica eles podem ser diferentes entre si e receberão identidades diferentes.

O algoritmo apresentado para calcular identificadores semânticos parte de dois

princípios: o primeiro diz que os nodos que são estruturalmente diferentes são semanticamente diferentes; o segundo diz que os nodos que são estruturalmente idênticos são semanticamente idênticos se e somente se seus pais são nodos raiz ou semanticamente idênticos.

Para realizar o casamento entre nodos semanticamente iguais o trabalho apresenta os conceitos *Type Territory* e *Node Territory*. *Type Territory* de um documento XML é o conjunto de todos os nodos tipo texto que são descendentes de pelo menos um ancestral em comum. Já o *Node Territory* é o *Type Territory* excluindo todos os nodos que são descendentes de outros nodos. Portanto, dentro de um território de um dado tipo existem territórios controlados por nodos deste tipo. O casamento entre nodos ocorre quando o identificador semântico de um nodo está contido no território de outro nodo.

Um ponto positivo deste trabalho é que o algoritmo leva em consideração não somente a estrutura de um nodo, mas também o contexto ao qual ele está inserido. Isto ocorre ao avaliar a equivalência semântica do nodo em análise e de seus pais e, posteriormente, ao realizar a associação entre os elementos em sucessivas versões do documento.

Diante disso, a abordagem favorece as consultas que podem envolver versões anteriores de um documento XML e também fornece apoio na avaliação de consulta incremental, pois pode reduzir o custo de avaliação da consulta.

A ideia de semântica no trabalho citado fica restrita às operações realizadas para detectar se um elemento de uma versão do documento XML é o mesmo em outra versão independente de sua estrutura ter mudado. Este trabalho não utiliza a semântica com o objetivo de explicar ou dar sentido a uma mudança.

A proposta de ZHAO et al (2004) é apresentar uma abordagem para descobrir estruturas (subárvores) que mudam com frequência a partir de um histórico de deltas estruturais de um documento XML.

O conhecimento extraído a partir desta abordagem pode ser utilizado na detecção de diferenças entre documentos XML, na indexação dos documentos, classificação e na mineração de regras de associação.

Visando a eficiência do processo de detecção, os autores apresentam um novo modelo de dados, *Historical-Document Object Model* (H-DOM). Trata-se de um modelo

compacto e com grande poder de expressão. Usando o H-DOM, o trabalho apresenta dois algoritmos que são capazes de descobrir todas as estruturas que sofreram mudanças frequentes.

Deleção e inserção de nodos em um documento são consideradas mudanças estruturais, enquanto a atualização denota uma mudança de conteúdo. Porém, esta abordagem não trata as operações de atualização. A proposta do trabalho traz como conhecimento os nodos que sofreram mudanças estruturais com mais frequência e também aqueles que ao longo do histórico de mudanças não foram encontradas alterações.

Além do conhecimento das estruturas que mais se modificam ao longo do tempo, é possível conhecer regras de associações entre os nodos que sofrem alterações. Por exemplo, sempre que um nodo N1 muda, a estrutura do nodo N2 também muda. As regras de associação podem ser muito úteis para, por exemplo, monitorar e prever as tendências de mudança em sites de comércio eletrônico e também para monitorar o comportamento e padrões de navegação de seus usuários.

Outro tipo de conhecimento extraído utilizando as técnicas deste trabalho é a detecção de padrões de mudança, como descobrir nodos onde frequentemente seus nodos filhos são deletados ou descobrir nodos cujas alterações são sempre de inserção.

O núcleo do trabalho está na detecção das estruturas que mudam com frequência. Diante disto, um ponto negativo da abordagem é que ela trata da mudança sintática do documento, ou seja, tem seu foco na estrutura do documento XML e não extrai conhecimento de possíveis mudanças de conteúdo. Esta abordagem se mostra útil em cenários onde as mudanças estruturais são críticas e as mudanças de conteúdo são raras ou podem ser ignoradas.

O trabalho citado trata a semântica sobre uma visão diferente da ótica desta pesquisa. Apesar de apresentar várias aplicações, o conhecimento extraído através da abordagem analisada nesta seção é superficial e não leva em consideração as informações que podem estar implícitas em alterações de conteúdo nos documentos XML.

O trabalho apresentado em ZHAO e SOURAV (2005) é uma extensão de ZHAO et al (2004) e sua principal diferença consiste em focar o processo de mineração de dados, em documentos XML, nas estruturas que mudam frequentemente, analisando as que possuem

maior valor semântico.

São propostos um novo modelo de dados chamado H-DOM+ e o algoritmo FASST *mining* para extrair as estruturas semânticas que mudam frequentemente (Frequently chAnging Semantic STructures - FASSTs). Diante disso, os autores propõem extrair mais conhecimento do domínio relacionado ao documento.

Com o intuito de tornar o processo de mineração das FASSTs interativo e flexível, o trabalho também apresenta uma linguagem declarativa para realização de consultas denominada FASSTQUEL (*FASST query language*).

Foi observado que em ZHAO et al (2004) nem todas as subestruturas detectadas eram dotadas de valor semântico significante. Com o objetivo de apresentar resultados mais interessantes em um domínio específico, restrições semânticas definidas pelo usuário são incorporadas ao processo de mineração de estruturas que mudam frequentemente. Esta função permite detectar mudanças de acordo com o conceito de semântica especificado pelo usuário.

Para auxiliar a detecção das FASSTs, algumas métricas são apresentadas. Uma delas é o valor de uma estrutura dinâmica. Quanto maior este valor mais significante é a subestrutura. O cálculo do valor de uma estrutura dinâmica envolve o percentual de nodos que sofreram mudanças de uma versão para outra do documento e o número de nodos envolvidos na subestrutura. De forma semelhante, outra métrica criada é o valor de versão dinâmica. Um valor alto significa que a subestrutura avaliada mudou mais frequentemente na sequência histórica de versões. E por fim, a métrica DoD utiliza os valores das versões e estruturas dinâmicas para calcular o quão significativas as alterações são.

O trabalho define como estrutura semântica aquela subestrutura que fornece informações que atendam a um conceito específico de um domínio. Os autores citam duas principais formas de obter o conceito de um domínio, a primeira delas é através de ontologias e a segunda é a construção de conceitos baseados em DTDs. Porém, é ressaltada a necessidade de personalização do conceito e, consequentemente, das estruturas semânticas obtidas de acordo com a preferência do usuário, uma vez que ele pode não estar interessado em todas as estruturas detectadas.

O conjunto de conceitos especificados pelo usuário é usado para orientar o processo de FASST *mining*. Os conceitos são representados sob uma forma hierárquica que especifica a relação entre eles. Os nodos da estrutura hierárquica podem ser classificados como primitivos, que representam os elementos básicos de um domínio, ou como não-primitivos aqueles que representam um conceito formado por um conjunto de conceitos primitivos. Os nodos não-primitivos se encontram em níveis mais altos na hierarquia.

O algoritmo de FASST *mining* tem como saída o modelo de dados H-DOM+ com todo o conhecimento extraído das estruturas semânticas que sofreram modificações significativas. Este conhecimento será usado nas consultas em FASSTQUEL com o intuito de buscar diferentes tipos de FASSTs.

Este trabalho traz como principal ponto positivo a possibilidade de personalização dos resultados de acordo com o conceito de semântica do usuário. A sintaxe da linguagem de consulta FASSTQUEL é apresentada por uma gramática relativamente pequena e seu poder de expressão é igualmente pequeno. O usuário é capaz de elaborar consultas que retornam as estruturas que foram inseridas ou removidas.

Para entrar com o conceito de domínio específico, o usuário deve criar um documento XML representando a hierarquia de conceitos a serem utilizados na consulta. A linguagem também permite consultas condicionais especificando os valores das métricas de estrutura, versão e DoD. Apesar da linguagem ter uma sintaxe de fácil aprendizado, nota-se que o usuário deve possuir uma noção básica do comportamento do algoritmo para usufruir das consultas. Isto torna o caráter interativo da abordagem deficiente.

Os resultados obtidos ainda se restringem a simples indicação de quais estruturas foram removidas e inseridas. A abordagem apresentada utiliza bem o domínio especificado pelo utilizador para a detecção e consulta das estruturas semanticamente mais significativas, porém, a forma de retorno do algoritmo tem pouco valor semântico, ou seja, não explica o significado dos resultados nem as razões das mudanças detectadas.

3.2 Controle de Mudanças no EDX

Esta seção apresenta um histórico da evolução do projeto que originou o protótipo deste trabalho, além de descrever suas funcionalidades e métodos utilizados para atingir o ob-

jetivo proposto.

3.2.1 Histórico

A ferramenta XPerseus (SILVA, 2011), foi criada com o intuito inicial de atuar no controle de versões de documentos XML. O sistema possui uma interface gráfica para a detecção e visualização de diferenças entre duas versões de um documento XML.

Esta aplicação, construída na linguagem de programação Java, utiliza o algoritmo JXyDiff para identificar e retornar as alterações realizadas entre as versões dos documentos comparados (TANI et al, 2012). A ferramenta permite que as diferenças detectadas sejam salvas em um arquivo XML chamado de delta.

A principal funcionalidade da ferramenta se concentra em uma função que recebe dois arquivos XML como entrada e retorna como saída o delta resultante da comparação entre estes arquivos. Na própria interface gráfica, o delta é exibido mostrando os nodos inseridos e removidos, além de informar a posição original de cada nodo, como no exemplo da Figura 3.1.

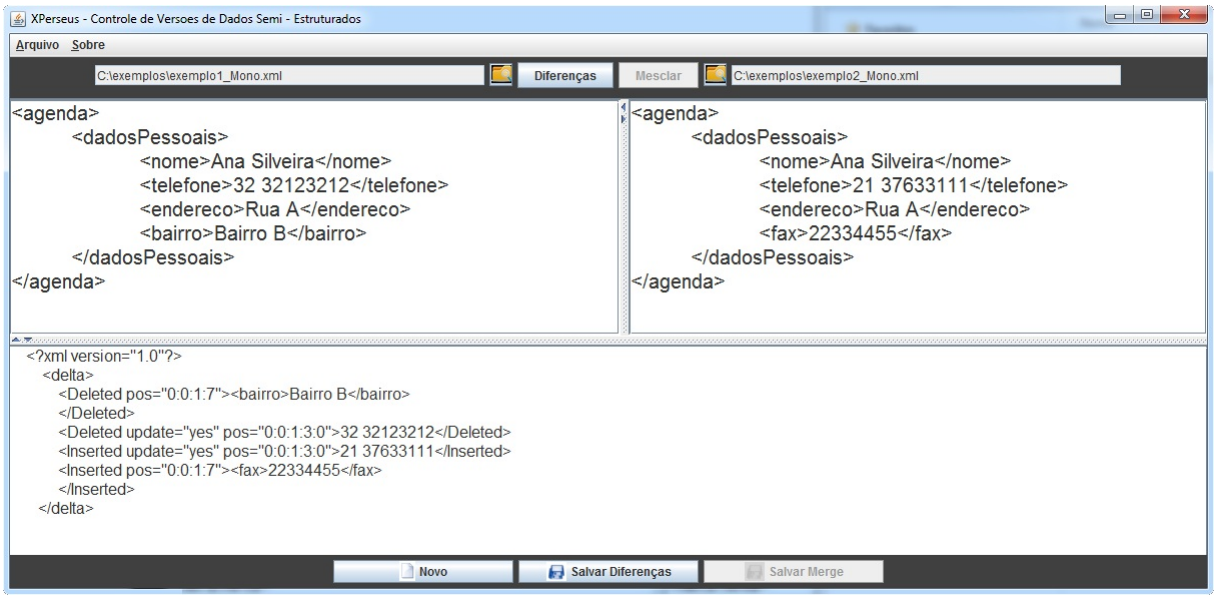

Figura 3.1: Diferenças detectadas pela ferramenta XPerseus

Na exibição do resultado da comparação, a representação textual da posição do elemento envolvido na operação apresenta certa deficiência com relação à legibilidade, por exemplo, o trecho de um delta script *<Deleted pos="0:0:1:7"> <bairro> Bairro B*

$< /bairro> < /Deleted>$ informa que o nodo alterado se encontra na posição "0:0:1:7". Esta representação exige que o usuário tenha conhecimento completo sobre a hierarquia do documento para localizar o nodo alterado.

Observando a necessidade de analisar a evolução de documentos XML de maneira abrangente, em um trabalho paralelo, GAZZOLA (2011), adiciona ao projeto XPerseus a alternativa de realizar inferências sobre a evolução dos documentos XML. O objetivo desta funcionalidade é fornecer informações de alto nível acerca das mudanças encontradas na comparação entre as versões.

A proposta de inferir informações da comparação entre duas versões de um documento insere a ferramenta no contexto do controle de mudanças de documentos XML, principal tema abordado nesta pesquisa.

O módulo de inferência da aplicação tira proveito da interface gráfica que coloca lado a lado duas versões de um documento XML. Este módulo utiliza a linguagem Prolog na realização da inferência e seu processo é dividido em três etapas.

A primeira etapa consiste na tradução das versões do documento em fatos Prolog através da biblioteca criada por LIMA et al (2011). Os fatos Prolog gerados a partir da tradução compõem a base de dados, ou a teoria, necessária para as consultas.

Na segunda etapa, são adicionadas regras Prolog à base de dados geradas na primeira etapa. Estas regras se dividem em duas categorias: as regras de inferência e as regras de negócio. As regras de inferência, são regras-base necessárias na execução do programa em Prolog. As regras de negócio são usadas para obter informações e exibir os resultados referentes a um domínio específico de conhecimento do usuário.

Na última etapa, de posse da teoria completa, o usuário seleciona as regras que serão exibidas como resultado e a execução da inferência ocorre utilizando a biblioteca tuProlog. Em seguida, os resultados são exibidos em uma área de texto da aplicação.

O uso da linguagem Prolog permite a representação do conhecimento de forma clara e, consequentemente, possibilita consultas e resultados de grande valor semântico. Esta abordagem vai além do conceito de semântica tratado pelos tabalhos descritos anteriormente, que retornam resultados limitados às operações realizadas nas alterações, como inserção e remoção.

O Prolog se encarrega de procurar soluções para as consultas na teoria extraída do documento XML. As regras são, na essência, relações entre os elementos do documento. Diante disso, os resultados gerados são informações de grande relevância no domínio de negócio especificado através das regras construídas pelo usuário e, de fato, apresentam uma explicação sobre a mudança ocorrida. É esta a ótica sobre semântica desta pesquisa, visando tomar como base o projeto XPerseus na construção de um protótipo que atue no controle de mudanças de documentos XML.

Um ponto que é passível de aperfeiçoamento no módulo de inferência da ferramenta é a etapa onde o usuário insere as regras que são utilizadas na execução da inferência. Apesar da interface gráfica permitir a criação das regras no próprio ambiente da aplicação, é exigido que o usuário conheça a linguagem Prolog e o padrão de construção das regras-base para que elas façam corretamente as referências aos fatos gerados pelo tradutor. Em síntese, o usuário fornece um arquivo Prolog com as regras-base que serão carregadas na ferramenta e as regras de negócio são redigidas em uma caixa de texto da aplicação.

3.2.2 Abordagem Proposta

Observa-se que o módulo de inferência da ferramenta XPerseus evoluiu ao ponto de poder se tornar uma ferramenta independente do módulo de controle de versões. O protótipo EDX apresentado neste trabalho, reúne todas as funcionalidades do módulo de inferência da ferramenta XPerseus somadas às novas funcionalidades propostas nesta abordagem.

O módulo de inferência da ferramenta XPerseus deixa implícito que para a sua utilização o usuário necessita ser especialista em Prolog. Isto limita o uso da função de inferência além de criar barreiras para a criação da regra-base uma vez que o usuário necessita compreender o modelo de regra que faz o vínculo com os fatos de cada versão do documento XML.

Diante disso, esta pesquisa apresenta o protótipo EDX com um módulo de construção de regras para a inferência de documentos XML. Essa abordagem tem dois objetivos principais: tornar a criação da regra-base um processo semiautomático e fazer com que a construção de regras de negócio seja uma tarefa de grande usabilidade, sem a ne-

cessidade de conhecer a linguagem Prolog. A Figura 3.2 ilustra o processo de construção de regras no EDX.

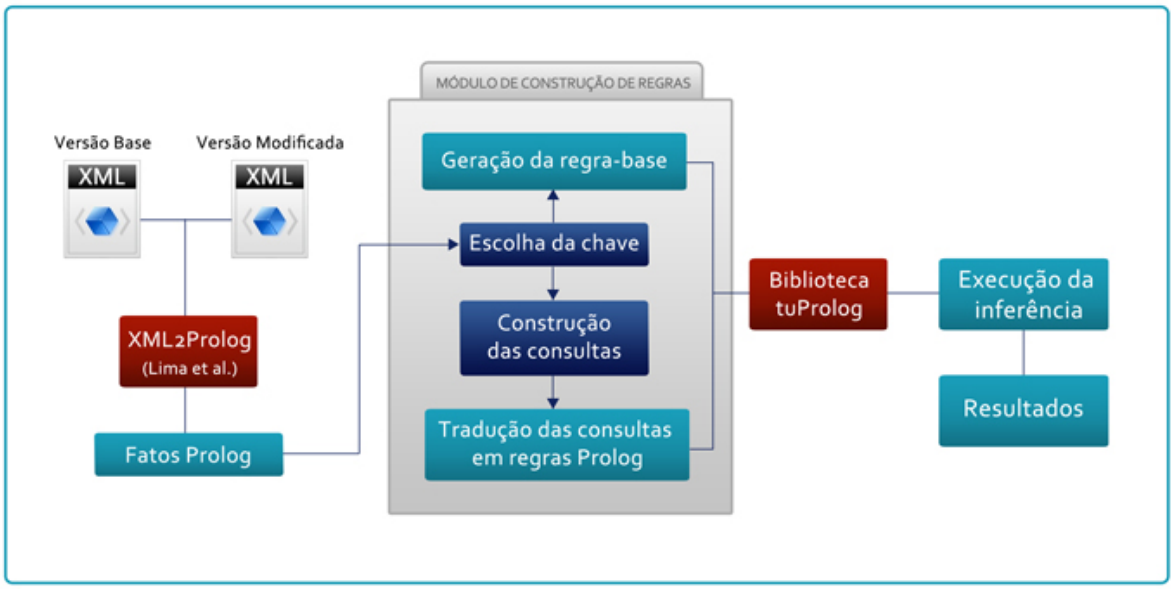

Figura 3.2: Processo da abordagem proposta

O processo se inicia na tradução dos elementos de duas versões do documento XML em fatos Prolog (LIMA et al, 2011). Cada elemento dá origem a um fato que descreve o seu conteúdo, sua posição hierárquica no documento e seu identificador, que é gerado sequencialmente.

Ao trabalhar com duas versões de um documento torna-se necessário criar um mecanismo de identificação dos elementos da versão base do XML e seus correspondentes na versão modificada do mesmo documento. A regra-base é essencial para fazer o vínculo entre os fatos gerados na tradução de XML para Prolog. Em outras palavras, a principal função de uma regra-base é garantir que as comparações serão feitas entre um mesmo elemento em ambas as versões.

O primeiro passo para a criação da regra-base é definir um atributo chave do contexto de negócio representado no documento XML. Esta chave caracteriza um identificador único do elemento e trata-se de um nodo pertencente ao elemento que se deseja realizar a inferência. Ao optar por construir uma regra, é apresentada ao usuário uma tela com todos os elementos do documento XML e cabe ao utilizador escolher um deles como atributo chave. Espera-se que o usuário, conhecedor do domínio de negócio em questão,

escolha um atributo cujo valor é único em cada elemento. Por exemplo, um atributo "CPF"poderia ser usado como chave para identificar elementos "pessoa".

A regra-base tem a função de identificar um mesmo elemento nas versões do documento e, por isso, não é exibida ao usuário. Esta importante regra segue a forma descrita na Listagem 3.1.

Listagem 3.1: Forma da regra-base

```
1 mesmoElemento(IdBase, IdModificado, Chave) :-
2 elemento(base, IdBase), elemento(modificado, IdModificado),
3 chave(IdBase, Chave), chave(IdModificado, Chave).
```

A regra pode ser lida da seguinte forma: para ser considerado um mesmo elemento em ambas as versões, este elemento deve possuir a mesma chave nos dois documentos XML.

A ação do usuário na geração semiautomática da regra-base se limita a escolha do atributo chave do contexto. O segundo passo é realizado pelo sistema e consiste na identificação do elemento principal do documento XML, que é o nodo que possui a raiz como nodo pai. Desta forma é criado um fato para cada versão do XML, chamado fato-base. Na Listagem 3.1, os fatos-base são exibidos na linha 2. O terceiro passo consiste na busca do fato Prolog cujo nome do functor é a chave escolhida. Este fato é exibido na linha 3 da Listagem anterior. Concluindo, os dois fatos-base de cada versão do XML compõem a regra-base.

A segunda contribuição desta pesquisa para o módulo de inferência da ferramenta consiste em uma interface que permite a construção de regras de negócio em alto nível, ou seja, sem a necessidade do usuário ser especialista em Prolog. Diante disso, após a escolha do atributo chave do contexto, é exibida a tela de construção de regras. Essa tela é composta de um campo onde o utilizador dá nome à sua regra e este nome será usado no texto de saída da consulta que utilizar a regra construída. Após a definição do nome da regra, o usuário deve selecionar um operador binário que atua sobre dois fatos extraídos do documento XML. Tais fatos são escolhidos pelo usuário de acordo com a regra que desejar montar. Os operadores disponíveis são "maior", "menor", "e", "igual", "diferente", "novo_elemento"e "elemento_excluído".

Os operadores "novo_elemento"e "elemento_excluído"não exigem fatos (operandos) definidos pelo usuário. Tais funcionalidades montam, em segundo plano, regras capazes de definir se determinado elemento existe em uma das versões do documento. A forma destas regras são descritas na Listagem 3.2 onde as linhas de 1 a 4 apresentam as regras que estabelecem as condições para que um elemento faça parte da versão original e da versão modificada, respectivamente. As linhas 5 a 8 mostram o modelo das regras que, através de negações, permitem inferir elementos que estão exclusivamente em uma das versões.

Listagem 3.2: Forma das regras de existência de um elemento em uma versão do documento

```
1 existe_modificado(SAIDA):-
2         funcionario(modificado,Fm), saida(Fm,SAIDA).
3 existe_base(SAIDA):-
4         funcionario(base,Fb), saida(Fb,SAIDA).
5 novo_elemento(SAIDA):-
6         existe_modificado(SAIDA), not(existe_base(SAIDA)).
7 elemento_excluido(SAIDA):-
8         existe_base(SAIDA), not(existe_modificado(SAIDA)).
```

Com o intuito de possibilitar a criação de regras mais complexas, é permitido a utilização de regras criadas pelo usuário na construção de novas regras. Esta funcionalidade torna o módulo de construção de regras flexível o suficiente para que o especialista no contexto de negócio possa realizar inferências sem limitação de quantidade de fatores. Todo o processo de construção de regras será exemplificado no estudo de caso do capítulo seguinte.

Após a construção das regras, a base de conhecimento necessária para a realização da inferência está formada, ou seja, os fatos, juntamente com a regra-base e as regras geradas pelo usuário no construtor de regras constituem a teoria que alimenta o processo de inferência.

O módulo de inferência exibe como resultados os valores dos elementos que tornaram válida uma consulta (regra). Uma vez que o módulo de construção de regras

possibilita a nomeação das mesmas, a saída do módulo utiliza este nome como cabeçalho das respostas das consultas. Isto permite a exibição dos resultados em uma linguagem textual próxima ao contexto de negócio facilitando, portanto, a compreensão das mudanças detectadas entre as versões do documento XML.

Desta forma, o EDX é capaz de retornar informações ricas em conhecimento. Ao contrário das outras abordagens analisadas que se limitam a informar a posição ou a estrutura de nodos inseridos e removidos, a abordagem proposta exibe resultados mais compreensíveis a um determinado contexto de negócio.

O objetivo geral desta abordagem é apresentar uma ferramenta de fácil utilização na busca por informações ricas em significado dentro de um domínio específico. Esse objetivo é alcançado automatizando o processo de criação de regras e fornecendo uma interface para a construçao de consultas em alto nível.

3.3 Quadro Comparativo

Com o intuito de resumir as principais características dos trabalhos analisados neste capítulo e relacioná-los de acordo com o foco desta pesquisa, a Figura 3.3 apresenta um quadro comparativo entre as abordagens estudadas.

	Cobena et al. (2002)	Zhang et al. (2004)	Zhao et al. (2004)	Zhao & Sourav (2005)	EDX
Principal função	Diferença baseada na estrutura	Diferença baseada no contexto	Mineração de mudanças frequentes	Mineração de mudanças significantes	Inferência de dados XML
Conhecimento extraído	Delta script (inserção, remoção, movimentação e atualização)	Delta script (inserção e remoção)	Padrões de mudanças frequentes	Padrões de mudanças frequentes dado um domínio específico	Semântica das mudanças encontradas entre duas versões de um documento XML
Interface	Não se aplica	Não se aplica	Não se aplica	Linha de comando	Interface gráfica
Permite consultas	Não	Não	Não	Sim	Sim
Uso de semântica	Não se aplica	Na associação entre os elementos de diferentes versões	Na dedução de regras de associação	No uso de conceito de domínio específico das consultas	Na associação entre os elementos, construção de regras e exibição dos resultados

Figura 3.3: Quadro comparativo entre os trabalhos relacionados

Esta comparação deixa evidente o uso da semântica na abordagem proposta desde a associação dos elementos entre as versões até a exibição dos resultados. Além disso,

o EDX se mostra como a única abordagem capaz de fornecer a razão das mudanças detectadas.

3.4 Conclusão

É possível observar que o conceito e a utilização de semântica no controle de mudanças variam em cada abordagem. A maioria das abordagens relacionam a semântica à estrutura do documento XML em técnicas que utilizam da hierarquia dos nodos para encontrar os nodos correspondentes nas versões do documento XML.

O conceito de semântica desta pesquisa diz respeito ao contexto de negócio representado no documento XML e possibilita a aproximação das mudanças detectadas dos reais motivos que levaram a alteração do documento. Por exemplo, em um cenário que descreve os funcionários de uma empresa, é de interesse desta abordagem saber se um funcionário recebeu aumento salarial, não simplesmente se o valor do salário foi alterado.

Dentre os trabalhos analisados, nenhum apresenta uma alternativa para atribuir significado aos resultados extraídos das mudanças. Este fato torna a abordagem apresentada neste trabalho a única que fornece informações ricas em significado, com o objetivo de explicar as alterações detectadas entre os documentos XML.

4 Exemplo de Utilização

Este capítulo tem o objetivo de exemplificar a utilização da ferramenta EDX, no contexto de documentos XML que descrevem os funcionários de uma empresa. Através de um exemplo prático, são apresentadas as ações que o usuário do protótipo EDX deve realizar para operar o módulo de inferências sem a necessidade de escrever regras na linguagem Prolog. Com isto, o usuário pode obter informações implícitas nas modificações detectadas entre as versões de um documento XML.

Na seção 4.1 é apresentado o contexto de negócio abordado no exemplo, a seção 4.2 apresenta a utilização do protótipo criado para a abordagem proposta e, por fim, as conclusões do estudo de caso na seção 4.3.

4.1 Contextualização

O contexto escolhido para o estudo de caso desta abordagem envolve as atividades de controle de recursos humanos de uma determinada empresa. As informações deste contexto são organizadas em um documento XML, e periodicamente a empresa pode gerar novas versões do documento com a finalidade de fazer comparações e ter uma visão geral das mudanças no quadro de funcionários.

Neste exemplo, a primeira versão do documento XML, e chamada de versão base ou original, é descrita na Listagem A.1 do apêndice. O documento descreve os funcionários da empresa e seus respectivos atributos: número de identificação na empresa (matrícula), nome, salário, carga horária, cargo, departamento e a filial onde o funcionário está alocado, conforme o fragmento representado na Listagem 4.1.

Listagem 4.1: Fragmento da versão base do documento exemplo

```
1  <funcionario>
2      <matricula>00010</matricula>
3      <nome>Carlos</nome>
4      <salario>1600</salario>
5      <cargaHoraria>36</cargaHoraria>
6      <cargo>Analista de Sistemas</cargo>
7      <depto>Tecnologia da Informacao</depto>
8      <filial>Juiz de Fora</filial>
9  </funcionario>
```

Suponha um cenário onde a empresa passou por mudanças em sua equipe e, consequentemente, o documento XML gerado com o quadro de funcionários passou por uma evolução dando origem a uma segunda versão, ou versão modificada, cujo fragmento é representado na Listagem 4.2. A versão modificada pode ser vista na Listagem A.2.

Listagem 4.2: Fragmento da versão modificada do documento exemplo

```
1  <funcionario>
2      <matricula>00010</matricula>
3      <nome>Carlos</nome>
4      <salario>2500</salario>
5      <cargaHoraria>40</cargaHoraria>
6      <cargo>Analista de Sistemas Pleno</cargo>
7      <depto>Tecnologia da Informacao</depto>
8      <filial>Juiz de Fora</filial>
9  </funcionario>
```

Dentre as modificações feitas entre as versões do documento estão a exclusão do funcionário Roberto e a contratação de dois novos funcionários, Junior e Eduardo. Estas modificações, especificamente, sugerem o uso das regras que fazem inferência de um novo elemento ou de um elemento excluído do documento XML.

Também foram feitas alterações nas informações dos funcionários já existentes, como por exemplo, o funcionário Carlos que teve seu salário, cargo e carga horária mo-

dificados como pode ser visto a partir da observação das listagens 4.1 e 4.2. Já a programadora Andrea teve seu salário mantido e sua carga horária alterada, enquanto Maria teve o salário e o cargo modificados. O vendedor Antonio teve somente seu salário modificado e Tiago mudou de filial. Tais modificações permitem uma série de inferências que auxiliam na compreensão da razão das mudanças. As diferenças entre as versões deste exemplo de utilização estão sintetizadas na Tabela 4.1

Tabela 4.1: Diferenças entre as versões do documento XML exemplo

Funcionário	Diferenças
Carlos	Salário, carga horária e cargo alterados
Pedro	Filial alterada
Roberto	Elemento excluído na versão modificada
Andrea	Carga horária alterada
Maria	Salário, cargo e departamento alterados
Antonio	Salário alterado
Tiago	Filial alterada
Junior	Elemento inserido na versão modificada
Eduardo	Elemento inserido na versão modificada

Este exemplo baseia-se na utilização do módulo de construção de regras por parte de um profissional da área de recursos humanos que tem como objetivo detectar e compreender as mudanças ocorridas no quadro de funcionários da empresa.

4.2 Utilização do Protótipo EDX

A Figura 4.1 representa a interface principal do protótipo EDX. São observados três setores, onde o setor inferior é destinado à exibição dos resultados da inferência e da tradução dos documentos XML para fatos Prolog. Os dois setores superiores, que dividem a tela ao meio, são áreas que exibem as versões do documento ao selecionar os arquivos utilizando os botões cujos ícones fazem alusão à pesquisa em diretórios. Cada botão de pesquisa em diretório é usado para carregar uma versão do documento que será exibido lado a lado nos setores mencionados anteriormente.

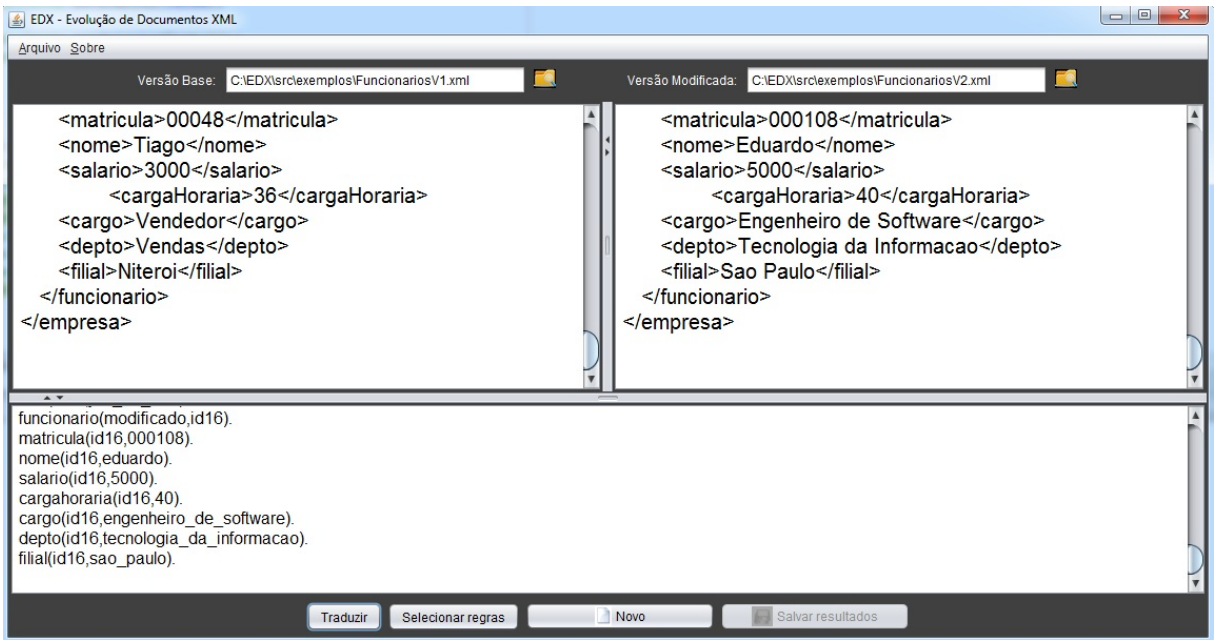

Figura 4.1: Interface principal do EDX

4.2.1 Tradução de XML em Fatos Prolog

A etapa seguinte à seleção das versões do documento é a tradução dos documentos XML para fatos Prolog. Este processo é realizado ao acionar o botão **Traduzir**. O resultado da tradução é apresentado no setor inferior, como observado na Figura 4.1. O resultado completo da tradução é apresentado na Listagem A.3.

4.2.2 Criação das Regras de Inferência

Para iniciar o processo de criação de regras, deve-se clicar no botão **Selecionar regras** e então é exibida uma janela, representada na Figura 4.2.

Figura 4.2: Interface de seleção de regras para inferência

Como no módulo de inferência do projeto de GAZZOLA (2011), se mantém a área onde o usuário pode digitar as regras em Prolog ou carregar um arquivo contendo as regras. Portanto, o usuário pode carregar regras prontas em um arquivo e executá-las diretamente, ou aproveitá-las utilizando-as na criação de novas regras. Além disso, o usuário pode construir novas regras sem utilizar nenhuma outra.

A abordagem proposta neste trabalho apresenta a opção de construir tais regras abstraindo a linguagem Prolog. Ao acionar o botão **Criar Regras**, uma nova janela é exibida e onde o usuário deve escolher qual é o atributo chave do seu contexto de negócio. Os atributos disponíveis no contexto escolhido para este exemplo são ilustrados na Figura 4.3.

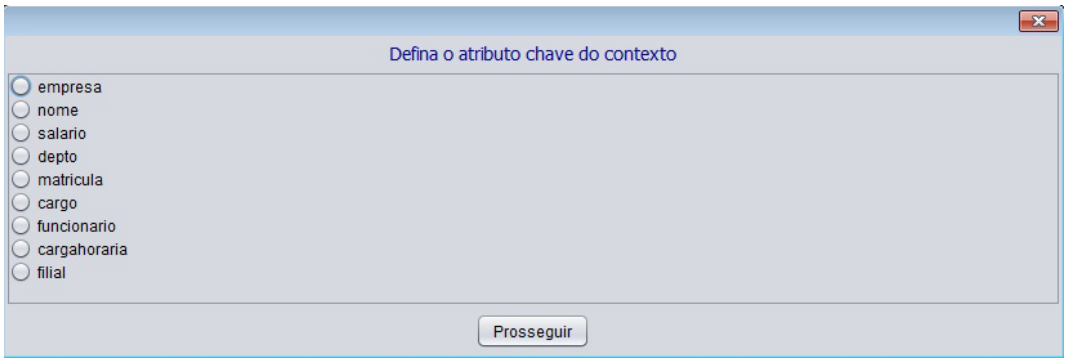

Figura 4.3: Interface de escolha do atributo chave do contexto

Como descrito na seção 3.2.1, a escolha do atributo chave do contexto é essencial na automação da criação da regra-base. No exemplo deste estudo, o atributo que caracteriza um identificador único do elemento funcionário é a sua matrícula. Portanto, a chave escolhida neste contexto é o atributo matrícula e no instante em que o usuário aciona o botão **Prosseguir**, é criada a regra-base descrita na Listagem 4.3. A regra permanece em segundo plano e, portanto, não é exibida ao usuário.

Listagem 4.3: Regra-base do estudo de caso

```
1 mesmo_elemento(Fb,Fm,MATRICULA):-
2 funcionario(base,Fb),funcionario(modificado,Fm),
3 matricula(Fb,MATRICULA),matricula(Fm,MATRICULA).
```

Após a escolha do atributo chave, é exibida a interface onde o usuário pode construir suas regras. O primeiro campo observado na Figura 4.4 é destinado ao nome da

regra. A regra construída é válida se, e somente se, as condições estabelecidas nos campos seguintes também forem válidas.

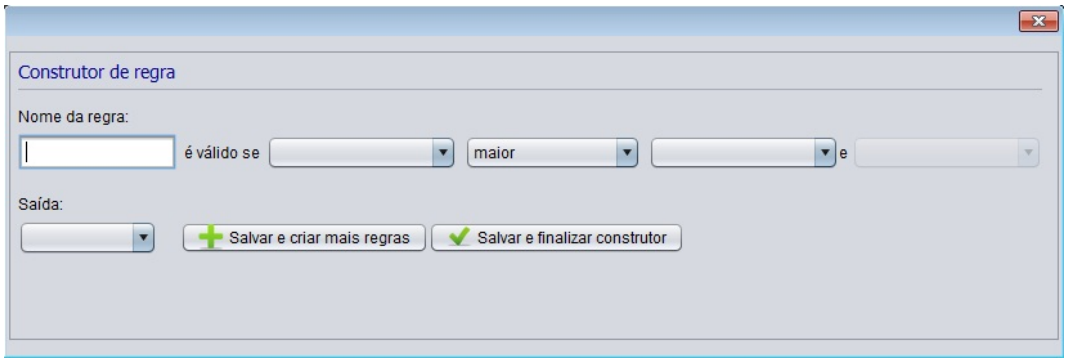

Figura 4.4: Interface do construtor de regras

São observados em um primeiro momento, quatro menus de seleção disponíveis, onde o primeiro e o terceiro são considerados como termos de uma operação, enquanto o segundo lista as opções de operadores. O quarto menu encontra-se desabilitado pois ele lista as regras já criadas pelo usuário, portanto, durante a primeira interação este se encontra vazio. A Figura 4.5 mostra as opções do primeiro menu, que são as mesmas do terceiro menu.

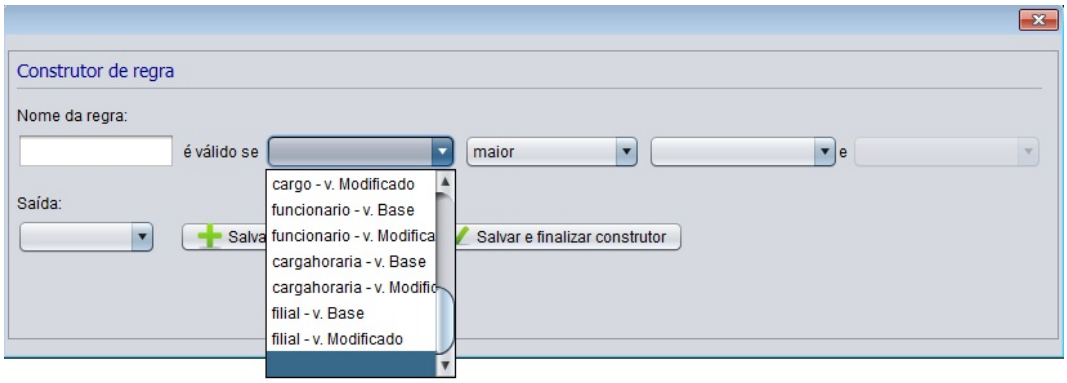

Figura 4.5: Opções de termos usados para a construção da regra

A Figura 4.6 exibe os operadores que o módulo de construção de regras oferece.

O quinto menu de seleção é responsável pela saída da execução da inferência. Deve-se escolher qual a informação desejada na exibição dos resultados. Por exemplo, se o usuário deseja obter como resultado da regra construída o nome dos funcionários que atendem às condições da regra, deve-se optar pelo item "nome"no menu de seleção.

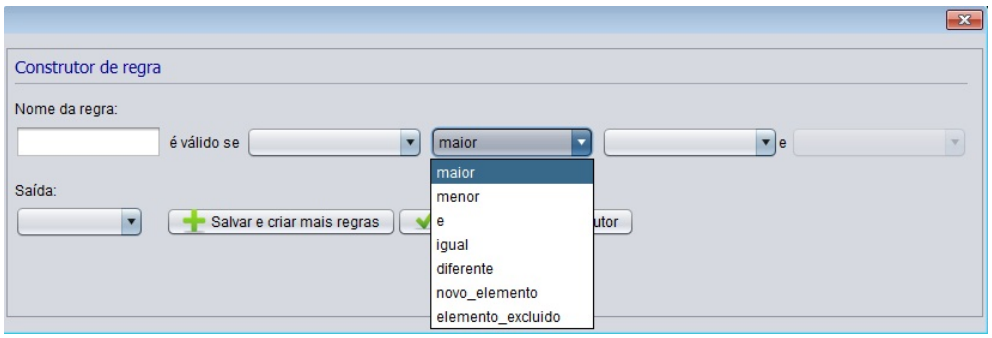

Figura 4.6: Opções de operadores usados para a construção da regra

O botão **Salvar e criar mais regras** armazena a regra criada e possibilita a criação de uma nova regra com a opção de utilizar a regra criada anteriormente. Já o botão **Salvar e finalizar construtor** armazena as regras criadas e encerra o módulo de construção para que seja feita a seleção das regras construídas e a execução da inferência.

A Figura 4.7 exibe a construção de uma regra que tem o objetivo de inferir todos os funcionários que foram transferidos.

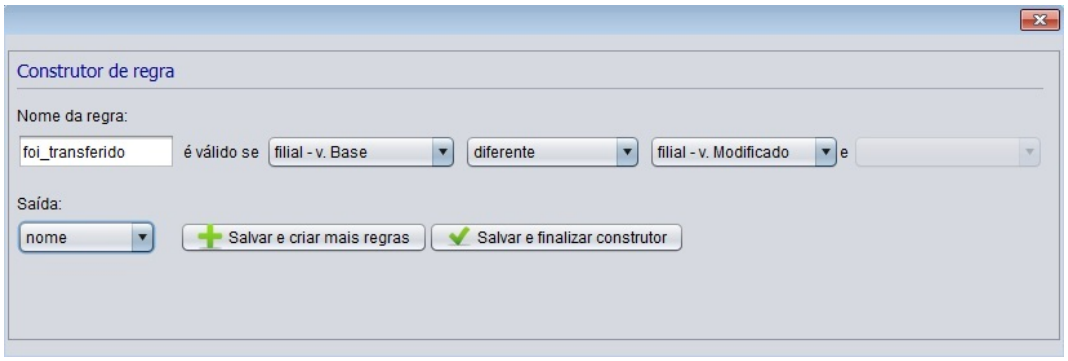

Figura 4.7: Construção da regra que define funcionários transferidos

A Figura 4.8 apresenta a construção de uma regra que permite consultar todos os funcionários que receberam aumento de salário, ou seja, aqueles que possuem o valor de salário na versão modificada do documento maior que o valor da versão base.

A Figura 4.9 ilustra como utilizar uma regra já construída na criação de outra que permite inferir aqueles funcionários que mudaram de cargo e receberam aumento, ou seja, esta informação traduz o conhecimento de funcionário promovido.

Ao finalizar a criação de regras clicando no botão **Salvar e finalizar o construtor**, é exibida a tela do módulo de inferência com as regras criadas já traduzidas para Prolog, conforme a Figura 4.10. Neste ponto, é permitido ao usuário criar novas regras

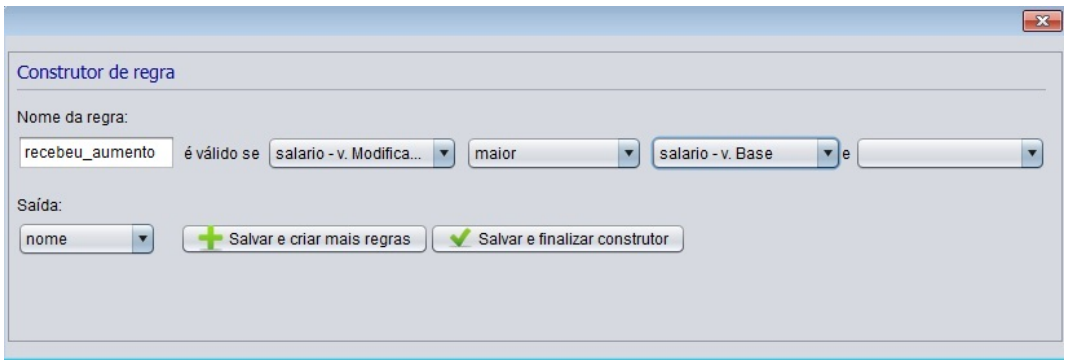

Figura 4.8: Construção da regra que define funcionários que receberam aumento salarial

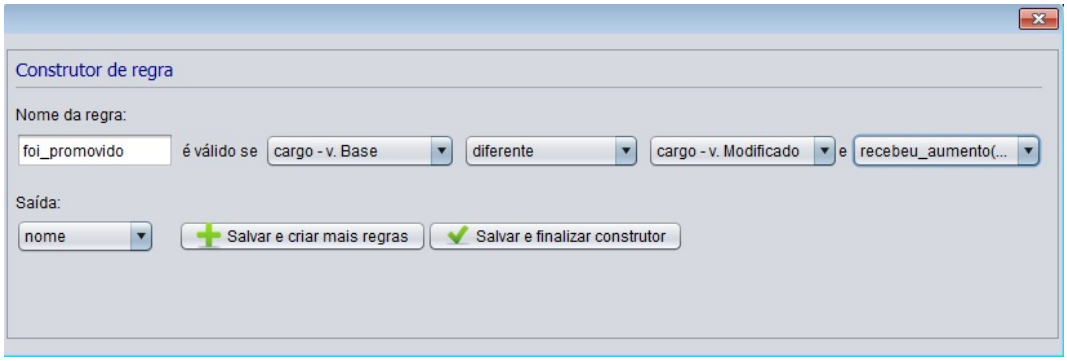

Figura 4.9: Construção da regra que define funcionários que foram promovidos

sem perder aquelas já construídas e, ainda, pode-se utilizá-las na criação das regras novas.

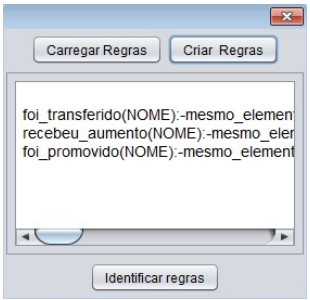

Figura 4.10: Regras criadas

As regras construídas anteriormente dizem respeito às modificações de conteúdo feitas nos nodos dos documentos XML. No entanto, o construtor de regras também é capaz de fornecer meios para detectar mudanças na estrutura de instâncias XML, através das opções novo_elemento e elemento_excluído presentes no segundo menu de seleção do módulo de construção de regras.

O operador novo_elemento permite criar consultas para obter os elementos que

não estavam presentes na primeira versão do documento e que foram criados na segunda versão. Caso o usuário deseje obter os elementos que foram removidos do documento, ou seja, existem na versão original e não existem na versão modificada, deve-se utilizar o operador elemento_excluído.

A Figura 4.11 ilustra a construção da regra que permite obter os funcionários contratados utilizando o operador que detecta os elementos incluídos na versão modificada do documento

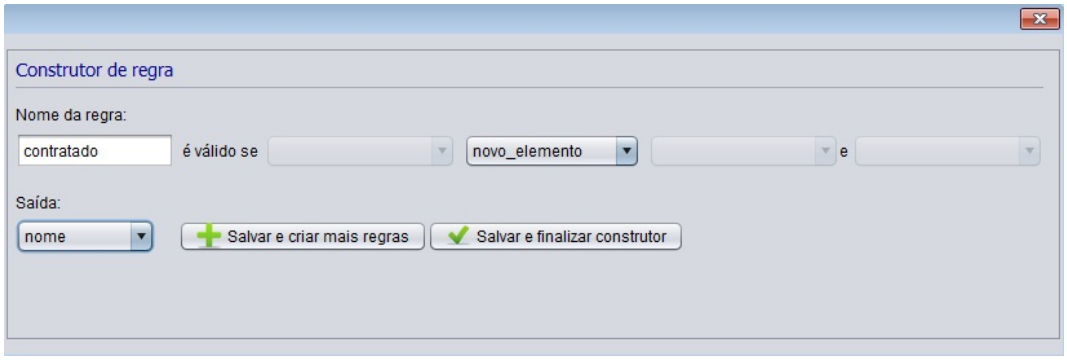

Figura 4.11: Construção da regra que define funcionários que foram contratados

A Figura 4.12 ilustra uma regra para consultar os funcionários demitidos utilizando o operador que detecta os elementos excluídos do documento.

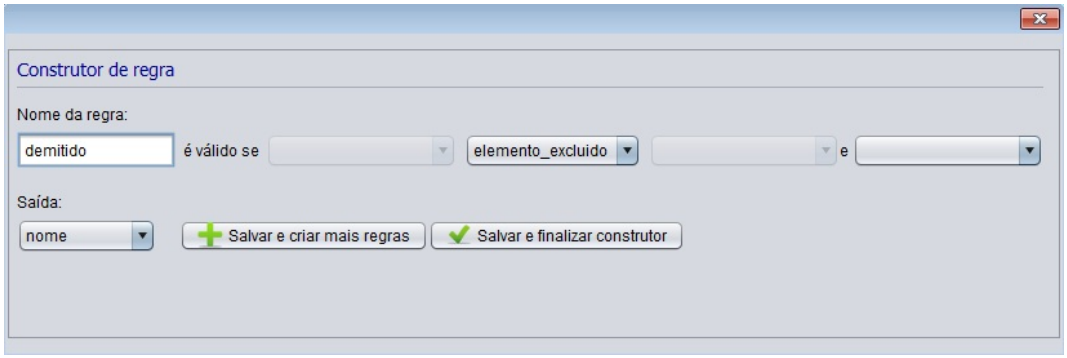

Figura 4.12: Construção da regra que define funcionários que foram demitidos

As regras geradas quando se utiliza estes operadores, no contexto deste exemplo, são apresentadas na Listagem 4.4. Nota-se que tais consultas sobre mudanças estruturais não exigem a seleção de termos, somente o operador e a opção de saída.

Listagem 4.4: Regras que permitem o uso dos operadores de mudança estrutural

```
1 existe_modificado(NOME):-funcionario(modificado,Fm), nome(Fm,NOME).
2 existe_base(NOME):-funcionario(base,Fb), nome(Fb,NOME).
3 contratado(NOME):- existe_modificado(NOME), not(existe_base(NOME)).
4 demitido(NOME):- existe_base(NOME), not(existe_modificado(NOME)).
```

O contexto de negócio trabalhado neste exemplo permite criar outras regras que auxiliam na compreensão da razão das mudanças. Além de regras que determinam os funcionários que receberam aumento, aqueles que foram promovidos, que mudaram de cargo ou que foram transferidos, pode-se também construir regras que extraem a informação de funcionários que possuem salários incompatíveis, por exemplo. Funcionários que tiveram a carga horária aumentada e seus salários mantidos podem ser considerados funcionários com salário incompatível. Esta regra, representada em Prolog na Listagem 4.5, faz uso de uma regra auxiliar que determina os salários que se mantiveram inalterados.

Listagem 4.5: Regra usada para determinar funcionários com salário incompatível

```
1 salario_incompativel(NOME):-
2         mesmo_elemento(Fb,Fm,MATRICULA),
3         cargahoraria(Fm,CARGAHORARIAModificado),
4         cargahoraria(Fb,CARGAHORARIABase),
5         CARGAHORARIAModificado > CARGAHORARIABase,
6         nome(Fb,NOME),mesmo_salario(NOME).
```

4.2.3 Execução da Inferência e Resultados

Nesta etapa deve-se acionar o botão **Identificar regras** para filtrar aquelas que serão exibidas no resultado da inferência. Como observado na Figura 4.13, as regras são selecionadas antes de acionar o botão **Concluir** que dará início ao processo de inferência.

Os resultados são exibidos no setor inferior, como na Figura 4.14. A ferramenta também permite salvar os resultados em arquivo clicando no botão **Salvar resultados**.

Todas as regras criadas através do módulo de construção da abordagem proposta neste trabalho estão descritas em Prolog na Listagem A.4 do apêndice. Os resultados

Figura 4.13: Regras identificadas e selecionadas

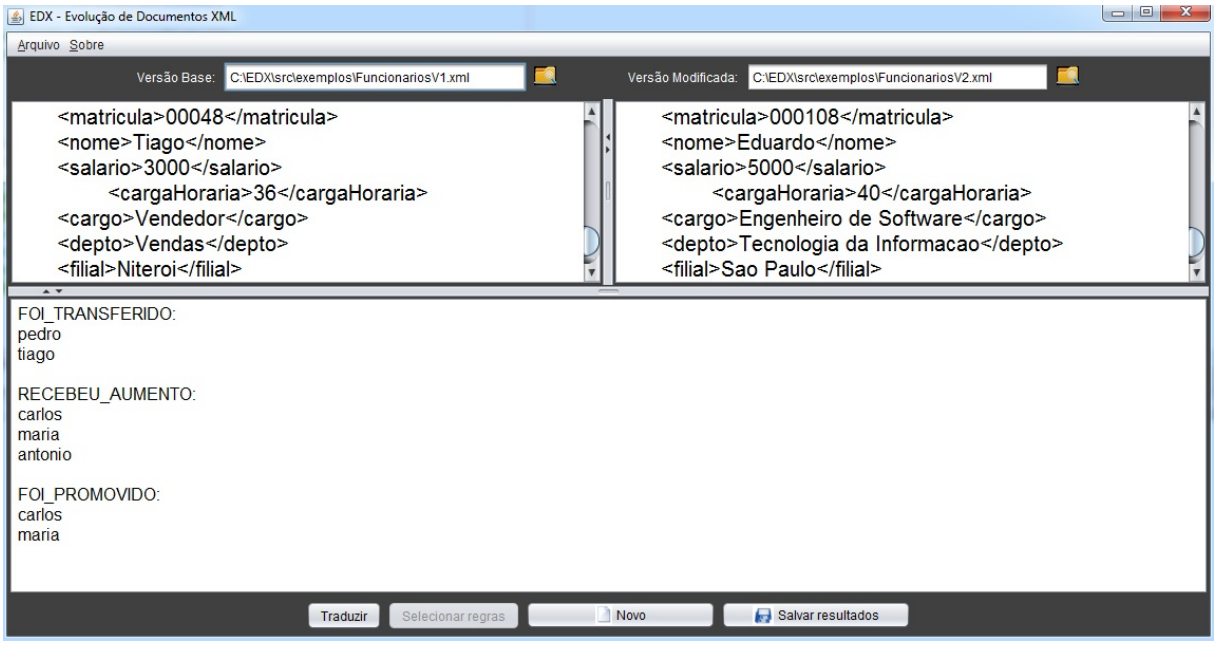

Figura 4.14: Resultados da inferência

obtidos são observados na Listagem A.5. A Tabela 4.2 sintetiza os resultados obtidos nas principais regras do exemplo de utilização.

4.3 Conclusão

O exemplo de utilização apresentado neste capítulo apresentou a abordagem proposta nesta pesquisa. É possível observar que o módulo de construção de regras cumpre seu objetivo ao permitir diversas consultas sem a necessidade de conhecer a linguagem Prolog.

Observa-se que a construção de regras semiautomáticas não limita o poder de consulta da ferramenta e torna a tarefa de criação de regras uma operação de fácil manipulação por um usuário conhecedor do contexto de negócio representado nos documentos.

Tabela 4.2: Resultados inferidos no exemplo de utilização

Regra	Resultado
foi_transferido(NOME)	pedro, tiago
recebeu_aumento(NOME)	carlos, maria, antonio
foi_promovido(NOME)	carlos, maria
contratado(NOME)	junior, eduardo
demitido(NOME)	roberto
mudou_cargo(NOME)	carlos, maria
mesmo_salario(NOME)	pedro, andrea, tiago
salario_incompativel(NOME)	andrea

Os resultados obtidos nas inferências trazem informações que permitem conhecer os motivos das modificações detectadas na evolução do documento XML. O conhecimento extraído através da inferência é um fator que contribui para a gerência de mudanças em um determinado projeto.

5 Considerações Finais

A Engenharia de Software enfrenta desafios na aplicação de GCS em projetos que utilizam dados semiestruturados. As características típicas de documentos XML, tais como sua organização hierárquica e suas marcações definidas pelo próprio desenvolvedor, inviabilizam a utilização de algumas ferramentas tradicionais na gerência de configuração de software. Neste cenário pode-se observar pesquisas que buscam adaptar as técnicas de GCS a documentos XML, visando tirar proveito de suas particularidades para atuar no controle de mudanças.

É possível observar que grande parte das pesquisas se concentram em detectar as diferenças sintáticas de documentos XML e, com isto, obtém-se como resultados as modificações de caráter estrutural. Através do estudo dos trabalhos relacionados, concluiu-se que algumas abordagens consideram as técnicas de associação de elementos de diferentes versões de um documento como uma abordagem que utiliza semântica.

A proposta apresentada neste trabalho segue a ótica de diferenças semânticas entre documentos XML. Esta abordagem faz uso de inferência de informações através da conversão dos documentos em fatos Prolog permitindo, desta forma, a criação de regras que funcionam como consultas à base de conhecimento extraída das versões XML. Os resultados obtidos nesta abordagem auxiliam na compreensão das modificações detectadas através da inferência de informações implícitas entre as versões.

Contudo, foi observada a necessidade do usuário conhecer a linguagem Prolog para operar o módulo de inferência. Além disso, era necessária a criação de regras fundamentais (regras-base) para associar um mesmo elemento nas versões do documento. Diante disso, este trabalho apresenta o protótipo EDX com interfaces gráficas que possibilitam a criação semiautomática de uma única regra-base além de permitir a criação de regras em alto nível, ou seja, dispensa a obrigatoriedade de conhecer Prolog para utilizar a ferramenta.

Para exemplificar a proposta do trabalho, foi apresentado um exemplo de utilização onde pôde-se extrair informações implícitas entre duas versões consecutivas de um

documento XML. O exemplo mostrou a eficácia do módulo construtor de regras permitindo a criação de consultas sem utilizar a sintaxe Prolog.

Considerar diferentes tipos de dados como datas, números e valores lógicos pode agregar mais poder de consultas ao módulo de inferência. Trabalhos futuros podem explorar a integração da ferramenta com *XML Schemas*, visando explorar as definições de tipo e estrutura presentes nestes documentos.

Outras formas de associação de elementos entre as versões podem contribuir para comparações entre documentos XML cujas alterações afetam sua estrutura. Por exemplo, a detecção de similaridade semântica abordada em OLIVEIRA (2012), daria maior flexibilidade ao módulo de inferência, permitiria a comparação de documentos com estruturas diferentes, além de eliminar a etapa de escolha do atributo chave.

Trabalhos futuros podem adicionar ao módulo de construção de regras do EDX, campos editáveis para possibilitar a construção de consultas com termos diferentes dos elementos do documento. Melhorias na interface e na navegabilidade do protótipo também podem contribuir para a evolução do projeto. Por exemplo, a inclusão de um mecanismo capaz de adicionar mais menus de seleção na interface de construção de regras, visando a utilização de mais de uma regra aninhada na criação de outras, além de possibilitar a criação de regras com mais de uma condição. Outra otimização a ser feita é a possibilidade do usuário dar nomes aos documentos XML e utilizá-los na tradução dos elementos para fatos Prolog.

Os elementos dos documentos XML cujos valores possuem espaçamento ou caracteres em caixa alta, foram editados visando atender à sintaxe da biblioteca utilizada para a inferência. No entanto, é possível tratar estes valores para que os resultados exibam exatamente o mesmo texto dos elementos. Além disso, trabalhos futuros podem trabalhar a ideia de realizar comparações com mais de dois documentos.

Outra forma de incrementar o presente trabalho, seria criar mecanismos onde o sistema sugere regras básicas ao usuário ao detectar relações entre os elementos modificados. Desta forma, o usuário tem a tarefa de analisar as regras detectadas e realizar ações como renomeá-las ou adaptá-las, caso deseje utilizá-las, ou excluí-las.

Referências Bibliográficas

Bray, T.; Paoli, J.; Sperberg-McQueen, C. M.; Maler, E. ; Yergeau, F. Extensible markup language (XML) 1.0. 2008.

Cobena, G.; Abiteboul, S. ; Marian, A. **Detecting changes in XML documents**. In: ICDE, p. 41–52, 2002.

Cunha, A.; Albrecht, F. ; Fernandes, R. Q. A. Inferência sobre ontologias: o reencontro com Prolog. **Instituto Militar de Engenharia**, 2007.

Estublier, J. **Software configuration management: a roadmap**. In: Proceedings of the Conference on The Future of Software Engineering, ICSE '00, p. 279–289, New York, NY, USA, 2000. ACM.

Fontes, C. A. Explorando inferência em um sistema de anotação semântica. **Instituto Militar de Engenharia**, 2011.

Gazzola, P. O. L. Inferência em documentos XML utilizando Prolog. **Universidade Federal de Juiz de Fora (UFJF)**, 2011.

Lima, D. M. G.; Delgado, C.; Murta, L. ; Braganholo, V. Consultando documentos XML utilizando inferência. **SBBD**, 2011.

Mello, R. S.; Dorneles, C. F.; Kade, A.; de Paula Braganholo, V. ; Heuser, C. A. Dados semi-estruturados. **Instituto de Informática da Universidade do Rio Grande do Sul (UFRGS)**, 2001.

Murta, L. G. P.; Werner, C. M. L. Gerência de Configuração no Desenvolvimento Baseado em Componentes. **UFRJ Rio de Janeiro RJ Brasil**, 2006.

Oliveira, A. M. Uso de inferência na evolução de documentos semi-estruturados no contexto de controle de modificações. **Universidade Federal Fluminense (UFF)**, 2011.

Oliveira, D. V. D. Similaridade em documentos XML. **Universidade Federal de Juiz de Fora (UFJF)**, 2012.

Palazzo, L. A. M. **Introdução à Programação Prolog**. Pelotas: Universidade Católica de Pelotas, 1997.

Pressman, R. S. **Engenharia de Software**. 5. ed., Rio de Janeiro: McGraw-Hill, 2002.

Silva, R. B. Uma Interface Gráfica para Detecção de Diferenças entre Documentos XML. **Universidade Federal de Juiz de Fora (UFJF)**, 2011.

Tani, R.; Molli, P. ; Jouille, F. **JXyDiff LibreSource**. https://github.com/tanob/jxydiff, 2012.

Zhang, S.; Dyreson, C. ; Snodgrass, R. T. **Schema-less, semantics-based change detection for XML documents**, 2004.

Zhao, Q.; Sourav, S. B.; Mohania, M. ; Kambayashi, Y. **Discovering frequently changing structures from historical structural deltas of unordered XML**. In: Proceedings of the thirteenth ACM international conference on Information and knowledge management, CIKM '04, p. 188–197, New York, NY, USA, 2004. ACM.

Zhao, Q.; Sourav, S. B. **Fasst mining: discovering frequently changing semantic structure from versions of unordered XML documents**. In: Proceedings of the 10th international conference on Database Systems for Advanced Applications, DASFAA'05, p. 724–735, Berlin, Heidelberg, 2005. Springer-Verlag.

A Apêndice

Listagem A.1: Versão original de um documento XML

```
<?xml version="1.0" encoding="UTF-8"?>
<empresa>
   <funcionario>
      <matricula>00010</matricula>
      <nome>Carlos</nome>
      <salario>1600</salario>
      <cargaHoraria>36</cargaHoraria>
      <cargo>Analista de Sistemas</cargo>
      <depto>Tecnologia da Informacao</depto>
      <filial>Juiz de Fora</filial>
   </funcionario>
   <funcionario>
      <matricula>00014</matricula>
      <nome>Pedro</nome>
      <salario>1900</salario>
      <cargaHoraria>40</cargaHoraria>
      <cargo>Suporte Help Desk</cargo>
      <depto>Tecnologia da Informacao</depto>
      <filial>Juiz de Fora</filial>
   </funcionario>
      <funcionario>
      <matricula>00019</matricula>
      <nome>Roberto</nome>
      <salario>1600</salario>
      <cargaHoraria>36</cargaHoraria>
      <cargo>Analista de Sistemas</cargo>
      <depto>Tecnologia da Informacao</depto>
      <filial>Juiz de Fora</filial>
   </funcionario>
   <funcionario>
      <matricula>00024</matricula>
      <nome>Andrea</nome>
      <salario>2000</salario>
      <cargaHoraria>40</cargaHoraria>
      <cargo>Programador</cargo>
      <depto>Tecnologia da Informacao</depto>
      <filial>Sao Paulo</filial>
   </funcionario>
   <funcionario>
      <matricula>00034</matricula>
      <nome>Maria</nome>
      <salario>3200</salario>
      <cargaHoraria>44</cargaHoraria>
      <cargo>Gerente de Vendas</cargo>
      <depto>Vendas</depto>
      <filial>Rio de Janeiro</filial>
   </funcionario>
```

```
48      <funcionario>
49          <matricula>00044</matricula>
50          <nome>Antonio</nome>
51          <salario>3000</salario>
52          <cargaHoraria>36</cargaHoraria>
53          <cargo>Vendedor</cargo>
54          <depto>Vendas</depto>
55          <filial>Niteroi</filial>
56      </funcionario>
57      <funcionario>
58          <matricula>00048</matricula>
59          <nome>Tiago</nome>
60          <salario>3000</salario>
61          <cargaHoraria>36</cargaHoraria>
62          <cargo>Vendedor</cargo>
63          <depto>Vendas</depto>
64          <filial>Niteroi</filial>
65      </funcionario>
66  </empresa>
```

Listagem A.2: Versão modificada de um documento XML

```
1   <?xml version="1.0" encoding="UTF-8"?>
2   <empresa>
3       <funcionario>
4           <matricula>00010</matricula>
5           <nome>Carlos</nome>
6           <salario>2500</salario>
7           <cargaHoraria>40</cargaHoraria>
8           <cargo>Analista de Sistemas Pleno</cargo>
9           <depto>Tecnologia da Informacao</depto>
10          <filial>Juiz de Fora</filial>
11      </funcionario>
12      <funcionario>
13          <matricula>00014</matricula>
14          <nome>Pedro</nome>
15          <salario>1900</salario>
16          <cargaHoraria>40</cargaHoraria>
17          <cargo>Suporte Help Desk</cargo>
18          <depto>Tecnologia da Informacao</depto>
19          <filial>Sao Paulo</filial>
20      </funcionario>
21      <funcionario>
22          <matricula>00024</matricula>
23          <nome>Andrea</nome>
24          <salario>2000</salario>
25          <cargaHoraria>44</cargaHoraria>
26          <cargo>Programador</cargo>
27          <depto>Tecnologia da Informacao</depto>
28          <filial>Sao Paulo</filial>
29      </funcionario>
30      <funcionario>
31          <matricula>00034</matricula>
32          <nome>Maria</nome>
```

```
33          <salario>3900</salario>
34          <cargaHoraria>44</cargaHoraria>
35          <cargo>Analista</cargo>
36          <depto>Marketing</depto>
37          <filial>Rio de Janeiro</filial>
38      </funcionario>
39      <funcionario>
40          <matricula>00044</matricula>
41          <nome>Antonio</nome>
42          <salario>3900</salario>
43          <cargaHoraria>36</cargaHoraria>
44          <cargo>Vendedor</cargo>
45          <depto>Vendas</depto>
46          <filial>Niteroi</filial>
47      </funcionario>
48      <funcionario>
49          <matricula>00048</matricula>
50          <nome>Tiago</nome>
51          <salario>3000</salario>
52          <cargaHoraria>36</cargaHoraria>
53          <cargo>Vendedor</cargo>
54          <depto>Vendas</depto>
55          <filial>Rio de Janeiro</filial>
56      </funcionario>
57      <funcionario>
58          <matricula>00058</matricula>
59          <nome>Junior</nome>
60          <salario>3000</salario>
61          <cargaHoraria>36</cargaHoraria>
62          <cargo>Vendedor</cargo>
63          <depto>Vendas</depto>
64          <filial>Juiz de Fora</filial>
65      </funcionario>
66      <funcionario>
67          <matricula>000108</matricula>
68          <nome>Eduardo</nome>
69          <salario>5000</salario>
70          <cargaHoraria>40</cargaHoraria>
71          <cargo>Engenheiro de Software</cargo>
72          <depto>Tecnologia da Informacao</depto>
73          <filial>Sao Paulo</filial>
74      </funcionario>
75  </empresa>
```

Listagem A.3: Fatos traduzidos

```
1  empresa(base).
2  funcionario(base,id2).
3  matricula(id2,00010).
4  nome(id2,carlos).
5  salario(id2,1600).
6  cargahoraria(id2,36).
7  cargo(id2,analista_de_sistemas).
8  depto(id2,tecnologia_da_informacao).
```

```
9  filial(id2,juiz_de_fora).
10 funcionario(base,id3).
11 matricula(id3,00014).
12 nome(id3,pedro).
13 salario(id3,1900).
14 cargahoraria(id3,40).
15 cargo(id3,suporte_help_desk).
16 depto(id3,tecnologia_da_informacao).
17 filial(id3,juiz_de_fora).
18 funcionario(base,id4).
19 matricula(id4,00019).
20 nome(id4,roberto).
21 salario(id4,1600).
22 cargahoraria(id4,36).
23 cargo(id4,analista_de_sistemas).
24 depto(id4,tecnologia_da_informacao).
25 filial(id4,juiz_de_fora).
26 funcionario(base,id5).
27 matricula(id5,00024).
28 nome(id5,andrea).
29 salario(id5,2000).
30 cargahoraria(id5,40).
31 cargo(id5,programador).
32 depto(id5,tecnologia_da_informacao).
33 filial(id5,sao_paulo).
34 funcionario(base,id6).
35 matricula(id6,00034).
36 nome(id6,maria).
37 salario(id6,3200).
38 cargahoraria(id6,44).
39 cargo(id6,gerente_de_vendas).
40 depto(id6,vendas).
41 filial(id6,rio_de_janeiro).
42 funcionario(base,id7).
43 matricula(id7,00044).
44 nome(id7,antonio).
45 salario(id7,3000).
46 cargahoraria(id7,36).
47 cargo(id7,vendedor).
48 depto(id7,vendas).
49 filial(id7,niteroi).
50 funcionario(base,id8).
51 matricula(id8,00048).
52 nome(id8,tiago).
53 salario(id8,3000).
54 cargahoraria(id8,36).
55 cargo(id8,vendedor).
56 depto(id8,vendas).
57 filial(id8,niteroi).
58
59 empresa(modificado).
60 funcionario(modificado,id9).
61 matricula(id9,00010).
```

```
62  nome(id9,carlos).
63  salario(id9,2500).
64  cargahoraria(id9,40).
65  cargo(id9,analista_de_sistemas_pleno).
66  depto(id9,tecnologia_da_informacao).
67  filial(id9,juiz_de_fora).
68  funcionario(modificado,id10).
69  matricula(id10,00014).
70  nome(id10,pedro).
71  salario(id10,1900).
72  cargahoraria(id10,40).
73  cargo(id10,suporte_help_desk).
74  depto(id10,tecnologia_da_informacao).
75  filial(id10,sao_paulo).
76  funcionario(modificado,id11).
77  matricula(id11,00024).
78  nome(id11,andrea).
79  salario(id11,2000).
80  cargahoraria(id11,44).
81  cargo(id11,programador).
82  depto(id11,tecnologia_da_informacao).
83  filial(id11,sao_paulo).
84  funcionario(modificado,id12).
85  matricula(id12,00034).
86  nome(id12,maria).
87  salario(id12,3900).
88  cargahoraria(id12,44).
89  cargo(id12,analista).
90  depto(id12,marketing).
91  filial(id12,rio_de_janeiro).
92  funcionario(modificado,id13).
93  matricula(id13,00044).
94  nome(id13,antonio).
95  salario(id13,3900).
96  cargahoraria(id13,36).
97  cargo(id13,vendedor).
98  depto(id13,vendas).
99  filial(id13,niteroi).
100 funcionario(modificado,id14).
101 matricula(id14,00048).
102 nome(id14,tiago).
103 salario(id14,3000).
104 cargahoraria(id14,36).
105 cargo(id14,vendedor).
106 depto(id14,vendas).
107 filial(id14,rio_de_janeiro).
108 funcionario(modificado,id15).
109 matricula(id15,00058).
110 nome(id15,junior).
111 salario(id15,3000).
112 cargahoraria(id15,36).
113 cargo(id15,vendedor).
114 depto(id15,vendas).
```

```
115 filial(id15,juiz_de_fora).
116 funcionario(modificado,id16).
117 matricula(id16,000108).
118 nome(id16,eduardo).
119 salario(id16,5000).
120 cargahoraria(id16,40).
121 cargo(id16,engenheiro_de_software).
122 depto(id16,tecnologia_da_informacao).
123 filial(id16,sao_paulo).
```

Listagem A.4: Regras geradas

```
1  foi_transferido(NOME):-mesmo_elemento(Fb,Fm,MATRICULA),filial(Fb,
       FILIALBase),filial(Fm,FILIALModificado),FILIALBase\=
       FILIALModificado,nome(Fb,NOME).
2
3  recebeu_aumento(NOME):-mesmo_elemento(Fb,Fm,MATRICULA),salario(Fm,
       SALARIOModificado),salario(Fb,SALARIOBase),SALARIOModificado>
       SALARIOBase,nome(Fb,NOME).
4
5  foi_promovido(NOME):-mesmo_elemento(Fb,Fm,MATRICULA),cargo(Fb,
       CARGOBase),cargo(Fm,CARGOModificado),CARGOBase\=CARGOModificado,
       nome(Fb,NOME),recebeu_aumento(NOME).
6
7  contratado(NOME):- existe_modificado(NOME),not(existe_base(NOME)).
8
9  demitido(NOME):- existe_base(NOME),not(existe_modificado(NOME)).
10
11 mudou_cargo(NOME):-mesmo_elemento(Fb,Fm,MATRICULA),cargo(Fb,CARGOBase
       ),cargo(Fm,CARGOModificado),CARGOBase\=CARGOModificado,nome(Fb,
       NOME).
12
13 mesmo_salario(NOME):-mesmo_elemento(Fb,Fm,MATRICULA),salario(Fb,
       SALARIOBase),salario(Fm,SALARIOModificado),SALARIOBase==
       SALARIOModificado,nome(Fb,NOME).
14
15 salario_incompativel(NOME):-mesmo_elemento(Fb,Fm,MATRICULA),
       cargahoraria(Fm,CARGAHORARIAModificado),cargahoraria(Fb,
       CARGAHORARIABase),CARGAHORARIAModificado>CARGAHORARIABase,nome(Fb,
       NOME),mesmo_salario(NOME).
16
17 mesmo_elemento(Fb,Fm,MATRICULA):-funcionario(base,Fb),funcionario(
       modificado,Fm),matricula(Fb,MATRICULA),matricula(Fm,MATRICULA).
18
19 existe_modificado(NOME):-funcionario(modificado,Fm),nome(Fm,NOME).
20
21 existe_base(NOME):-funcionario(base,Fb),nome(Fb,NOME).
```

Listagem A.5: Resultados da inferência

```
1 FOI_TRANSFERIDO:
2 pedro
3 tiago
4
```

```
5  RECEBEU_AUMENTO:
6  carlos
7  maria
8  antonio
9
10 FOI_PROMOVIDO:
11 carlos
12 maria
13
14 CONTRATADO:
15 junior
16 eduardo
17
18 DEMITIDO:
19 roberto
20
21 MUDOU_CARGO:
22 carlos
23 maria
24
25 MESMO_SALARIO:
26 pedro
27 andrea
28 tiago
29
30 SALARIO_INCOMPATIVEL:
31 andrea
```