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

Uso de Datalog para Realizar Inferências em Documentos Semiestruturados

Carolina de Oliveira Cunha

JUIZ DE FORA
DEZEMBRO, 2011

Uso de Datalog para Realizar Inferências em Documentos Semiestruturados

Carolina de Oliveira Cunha

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 Julio

JUIZ DE FORA

DEZEMBRO, 2011

Uso de Datalog para Realizar Inferências em Documentos Semiestruturados

Carolina de Oliveira Cunha

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 Julio
M. Sc.

Edmar Welington Oliveira
M. Sc

Jairo Francisco de Souza
M. Sc.

JUIZ DE FORA

DEZEMBRO, 2011

Aos meus amigos, os irmãos que eu escolhi.

À minha família, meu porto seguro.

Resumo

Este trabalho apresenta uma abordagem de evolução de documentos semiestruturados baseada em inferência, utilizando a linguagem de consulta Datalog como instrumento para a criação das regras e descoberta de novas informações a partir dos dados presentes nesses documentos. A proposta apresentada é a de adaptar técnicas de gerência de configuração de software - no que diz respeito ao controle de modificações de dados semiestruturados - e, a partir da aplicação de regras de inferência apontar a intenção do usuário ao criar a segunda versão. As razões da mudança representam os motivos pelos quais ela ocorreu e refletem a evolução dos documentos. Para finalizar e exemplificar a proposta, um estudo de caso é apresentado considerando um sistema de gerenciamento de informações de funcionários para exemplificar a proposta.

Palavras-chave: Datalog, gerência de configuração, dados semiestruturados, inferência.

Agradecimentos

A Deus, por me guiar pelos caminhos mais difíceis, sempre me preenchendo com coragem e força.

À minha mãe, por me mostrar o caminho das pedras, por me dar forças para resistir às adversidades e mostrar que eu posso, sim, realizar meus sonhos.

Ao meu pai, meu porto seguro, minha razão para lutar por dias melhores, para mim e, principalmente, para ele.

Ao meu irmão, por ser meu guia, meu companheiro, meu mentor, meu conselheiro, meu amigo... meu irmão.

À minha orientadora Alessandreia, por ser mais que apenas uma orientadora, uma amiga, e ter me ensinado lições importantes que eu vou carregar comigo por onde eu for daqui para frente.

Aos meus amigos do GET Plínio Garcia, Pedro Gazzola, Guilherme Martins, Danúbia Dias e Lenita Ambrosio, pelo companheirismo, apoio, bons momentos proporcionados nesse tempo de convivência e pela ajuda na confecção deste trabalho.

Ao amigo Leonardo Costa, por me mostrar o sentido da palavra amizade.

Ao amigo Victor Fonseca, por ser o irmão que eu escolhi pra mim, e ser o melhor que eu podia ter escolhido.

A todos os meus amigos, por me ajudarem a passar pelas minhas dificuldades com leveza e alegria.

A todos os professores, por, de alguma forma, terem contribuído para o meu crescimento acadêmico e pessoal.

"Não te esqueças de que o arado, dilacerando o solo, acaba igualmente desmantelado e ferido, entretanto, desse choque de forças surge o pão que te supre a mesa".

Autor Desconhecido

Sumário

Lista de Figuras

Lista de Abreviações

DCC	Departamento de Ciência da Computação
FCS	Frequently Changed Structures
GC	Gerência de Configuração de Software
GCS	Gerência de Configuração de Software
H-DOM	Historical-Document Object Model
HMF	Hipótese do Mundo Fechado
HTML	Hyper Text Markup Language
IC	Item de Configuração
OWL	Web Ontology Language
SCD	Semantic Change Detection
SQL	Standard Query Language
SWRL	Semantic Web Rule Language
UFJF	Universidade Federal de Juiz de Fora
XML	eXtensible Markup Language

1 Introdução

Este trabalho apresenta uma estratégia para a Gerência de Configuração de Dados Semiestruturados e utiliza Datalog e Inferência para realizar o controle de modificações e extrair informações adicionais àquelas já presentes nos dados.

1.1 Contextualização

Geralmente, as grandes empresas não utilizam apenas um sistema para o gerenciamento de suas atividades. São usados vários sistemas em conjunto, cada um com foco em uma parte da organização e especializado em certas funções. É necessário, então, que haja a comunicação desses sistemas para que os dados fornecidos pelas diversas fontes sejam utilizados em conjunto.

A demanda pelo desenvolvimento de técnicas que auxiliem a troca de dados entre sistemas cresceu muito, também, a partir da popularização da Internet - que é um meio heterogêneo onde os dados se encontram representados de diversas formas.

Por serem gerados de diversas fontes, esses dados não possuem uma forma definida. No entanto, para que possa ser feita a comunicação entre os diversos sistemas, é necessário que as fontes forneçam os dados em algum padrão que facilite o seu processamento.

Para resolver esse problema, uma das soluções encontradas foi a disponibilização dos dados de forma que seja fácil o seu processamento por sistemas que não conhecem sua estrutura. Para isso, a estrutura dos dados passou a ser disponibilizada juntamente com seu conteúdo, sendo esses dados chamados de semiestruturados. Os dados semiestruturados são dados onde a informação sobre a estrutura está embutida nos valores dos dados, tornando-os autodescritivos (Elmasri e Navathe, 2005). Para a representação desses dados, é utilizada a linguagem XML (eXtensible Markup Language), uma linguagem simples baseada em texto (W3C, 1996) - hoje um padrão de fato.

Porém, assim como todo documento de texto, os documentos semiestruturados

evoluem ao longo do tempo, sendo importante controlar essa evolução. Uma das formas de se efetuar esse controle é a partir da aplicação de técnicas de uma área da Engenharia de Software conhecida como Gerência de Configuração de Software (GCS). A GCS é responsável por controlar as modificações dos dados, seus impactos e sua distribuição. É subdividida em controle de alterações, controle da construção e controle de modificações (Murta, 2006).

1.2 Motivação

Com a crescente demanda para a integração de dados de diversas fontes, aumentou-se consideravelmente a utilização de documentos semiestruturados, já que esse formato de dados facilita o processamento para sistemas que não conheçam sua estrutura. Além disso, sua utilização foi impulsionada com a popularização da Internet como meio para troca de informações. Esses dados evoluem com o tempo, já que representam informações que podem mudar de acordo com o momento em que foram geradas ou acessadas. Assim, se faz necessária a especialização de técnicas já existentes para o controle de modificações dos documentos que armazenam esses dados. A partir do controle de modificações, podem ser analisados os significados reais das mudanças que ocorreram, seu impacto, e sua importância para o conjunto de dados representados e, assim, inferir qual é a real intenção do usuário ao efetuar as alterações.

1.3 Justificativa

Os dados semiestruturados têm sido muito utilizados para a integração e troca de informações entre sistemas. Porém, possuem algumas particularidades que os diferem de documentos comuns. Um exemplo seria o fato de sua estrutura ser fornecer juntamente o seu conteúdo e sua representação, feita de forma hierárquica. Por esses motivos, torna-se necessário o desenvolvimento de estratégias para o tratamento desses documentos, levando em conta sua estrutura diferenciada.

Quando se fala em arquivos que compõem sistemas, torna-se importante controlar sua evolução, a fim de manter a consistência do sistema e mapear sua evolução ao longo

do tempo. Para isso, é necessária a aplicação de técnicas de Gerência de Configuração de Software - área da Engenharia de Software que visa acompanhar o desenvolvimento dos artefatos do projeto - as quais devem ser especializadas para a utilização com esses documentos. Assim, surge a demanda pelo desenvolvimento de novas estratégias para a Gerência de Configuração de Dados Semiestruturados, em especial, na parte que remete ao Controle de Modificações. É importante também considerando-se o Controle de Modificações, analisar o significado das alterações, permitindo inferir os motivos que levaram a elas.

1.4 Objetivos

O objetivo principal desse trabalho é apresentar a linguagem Datalog como estratégia para, a partir da inferência de dados, controlar as modificações feitas ao longo do tempo em documentos semiestruturados. Outro objetivo é apresentar os conceitos de Gerência de Configuração e Gerência de Configuração de Dados Semiestruturados. Também é realizada a discussão de trabalhos que estão sendo desenvolvidos na área e outras abordagens propostas para o controle de modificações.

1.5 Organização do Trabalho

Este trabalho, além deste capítulo, está organizado como a seguir. O Capítulo 2 apresenta a linguagem Datalog, seus conceitos, histórico, sintaxe e uma breve comparação com a linguagem Prolog. O Capítulo 3 expõe os conceitos de Gerência de Configuração e Gerência de Configuração de Dados Semiestruturados. Além disso apresenta as abordagens que estão sendo estudadas para o controle de modificações, a ferramenta XPerseus e alguns trabalhos relacionados a este. No Capítulo 4, é mostrado o estudo de caso. No Capítulo 5, são apresentadas as considerações finais.

2 Datalog

Neste capítulo, é introduzida a linguagem Datalog e seus conceitos principais: histórico, características, estrutura, sintaxe e aplicações. Após a apresentação dos conceitos, é feita uma breve comparação entre o Datalog e o Prolog, uma linguagem que também pode ser usada no auxílio do Controle de Modificações baseado em inferência de dados.

2.1 Histórico

A busca por uma linguagem de consulta baseada em regras fez parte dos objetivos dos pesquisadores de Bancos de Dados durante muito tempo. Na década de 80, surgiu a demanda por integração entre aplicações de Inteligência Artificial e tecnologias de Bancos de Dados capazes de gerenciar grandes bases de conhecimento. Para tal, uma das soluções encontradas foi a combinação de Programação em Lógica com Bancos de Dados. Nesse contexto surgiu o Datalog (Stefano et al., 1989).

Nesta época, já existia uma linguagem de programação lógica composta por fatos e regras, chamada Prolog (Cunha et al., 2007). Foram obtidos resultados interessantes com a utilização de sistemas Prolog para banco de dados relacionais. Porém uma maior eficiência foi alcançada com a integração desses sistemas à interfaces inteligentes. Essa solução indica que, dessa forma, os problemas imediatos podem ser resolvidos. Contudo, a longo prazo, é necessária uma integração mais forte. Em outras palavras, o que se espera, de fato, é que os sistemas de gerenciamento de banco de dados ofereçam acesso direto aos dados e suportem interação através de regras, como nos paradigmas de programação. O Datalog foi o primeiro passo nessa direção, mas que não alcançou muito sucesso na época (Campagna et al., 2011).

Nos últimos anos, o Datalog ressurgiu e vem sendo utilizado para sistemas onde um nível de abstração mais alto é necessário para a execução de consultas de estruturas relacionais e gráficas. Também vem sendo usado quando é necessária a execução eficiente de consultas recursivas, a implementação de técnicas de manutenção incremental baseadas

em modelos relacionais, raciocínio formal e análise. Além disso, a recursividade oferecida nas consultas pelo Datalog se faz útil dada a complexidade das aplicações existentes hoje em dia, onde é cada vez mais necessária a interligação e análise de dados buscados em diversas fontes (Huang et al., 2011).

2.2 Características

O Datalog é uma linguagem não procedural de consulta baseada em Prolog. Sua essência está na capacidade que possui de definir e trabalhar com relações, que são suas unidades básicas (Bravenboer e Smaragdakis, 2009). Sua estratégia de execução, parecida com a estratégia *bottom-up*, permite a utilização de recursão, negação e implementação de dependências funcionais (Hajiyev et al., 2006). É uma linguagem quen nasceu no meio acadêmico como opção à linguagem de consulta SQL (Standard Query Language). Como vantagens, podem ser citadas sua capacidade de realizar consultas recursivas e sua semântica limpa e clara (Stefano et al., 1989).

Datalog oferece, também, suporte à inferência direta de fatos. Realizar inferência consiste na extração de informações extras a partir de outras já conhecidas. Utilizando-se das regras, é possível gerar novos fatos, relacionados aos já existentes (Stefano et al., 1989). Além disso, o fato de ser uma linguagem declarativa a torna uma boa opção para a construção de algoritmos complexos (Hajiyev et al., 2006).

2.3 Estrutura Básica

O Datalog é formado por fatos e regras. Regras são sentenças que permitem que fatos sejam deduzidos a partir de fatos já existentes (Stefano et al., 1989). As regras se assemelham a visões relacionais, especificando relações virtuais que não são armazenadas. Porém, ao contrário das visões relacionais, as regras podem utilizar a recursividade, rendendo relações virtuais, que não podem ser definidas em termos de visões relacionais básicas. Já um fato é uma afirmação sobre uma parte relevante do mundo, como: “Tiago é pai de Pedro”. Os fatos são especificados de modo semelhante à especificação das relações, exceto pelo fato de a inclusão dos nomes dos atributos não ser necessária. A máquina de

inferência tem a função de deduzir fatos novos do banco de dados interpretando as regras (Elmasri e Navathe, 2005).

Tanto as regras quanto os fatos Datalog são cláusulas do tipo *Horn*. Cláusulas *Horn* são cláusulas que podem conter, no máximo, um literal positivo - ou seja, uma fórmula atômica que não esteja precedida por *not* (Elmasri e Navathe, 2005).

Uma cláusula de *Horn* possui uma das seguintes formas:

$$\text{not}(P_1) \text{ OR not}(P_2) \text{ OR ... OR not}(P_n) \text{ OR Q} \qquad (1)$$

ou da forma

$$\text{not}(P_1) \text{ OR not}(P_2) \text{ OR ... OR not}(P_n) \qquad (2)$$

A cláusula (1) pode ser escrita como:

$$P_1 \text{ AND } P_2 \text{ AND ... AND}(P_n) => Q$$

que, em Datalog, representa a seguinte regra:

$$Q :- P_1, P_2, ..., P_n.$$

Já a cláusula (2) pode ser escrita em Datalog da seguinte forma:

$$P_1, P_2, ..., P_n.$$

2.4 Sintaxe

Segundo (Korth e Silberschartz, 2006), as regras de um programa Datalog possuem seguinte a forma geral:

$$L_0 \text{ :- } L_1, ..., L_n$$

onde cada L_i é um literal da forma $p_i(t_1, ..., t_k i)$ sendo p_i um predicado e $t_1, ..., t_k i$ os termos, que podem ser tanto constantes (valores numéricos por exemplo) como variáveis. O lado esquerdo *(LHS - left-hand side)* de uma cláusula Datalog é chamado de cabeçalho, e o lado direito *(RHS - right-hand side)* é chamado de corpo.

Os fatos nada mais são do que cláusulas com o corpo vazio, já regras são cláusulas com pelo menos um literal (fórmula atômica) presente no corpo. O exemplo a seguir, representando a relação entre avós e netos, ilustra a sintaxe dos fatos e regras.

Para definir a relação entre os avós e seus respectivos netos em Datalog, a regra seria:

avo(X,Z) :- pai(X,Y), pai(Y,Z)

Os fatos representam quais são os indivíduos com os quais se deseja fazer a relação. Os fatos são os seguintes:

pai(João, Pedro)

pai(Pedro, Tiago)

As relações indicadas pelos fatos acima são: João é pai de Pedro, e Pedro é pai de Tiago.

Dado o processamento dos fatos pela regra apresentada, será inferido que João é avô de Tiago.

avo(João, Tiago) :- pai(João, Pedro), pai(Pedro Tiago)

Para a escrita dos fatos e regras, é aconselhado o uso da seguinte convenção: constantes e predicados (setenças com significado implícito, sugerido pelo nome ou pelo número fixo de argumentos) devem ser iniciados com letras minúsculas, enquanto variáveis devem ser iniciadas com letras maiúsculas (Stefano et al., 1989).

Um predicado em que os argumentos são representados por constantes é chamado de fundamentado ou instanciado. As regras, geralmente, são formadas por um conjunto de variáveis, embora possam receber constantes como argumento (Elmasri e Navathe, 2005).

No Datalog, é utilizada a Hipótese do Mundo Fechado(HMF), onde apenas os fatos conhecidos são considerados verdadeiros. Isto implica diretamente na negação de fatos, uma vez que a afirmação de que um fato é falso passa a depender dos fatos conhecidos. No momento da avaliação da negação então passa a ser necessário o conhecimento de todos os fatos verdadeiros, mesmo que eles sejam derivados de outros. A principal vantagem da utilização da HMF é evitar a criação de *loops* (Nardon, 1996).

2.5 Segurança em Programas Datalog

A segurança em programas Datalog é muito importante, pois a utilização de recursão em regras mal formadas pode trazer muitos problemas, sendo o principal deles a formação de *loops* infinitos.

Um programa Datalog é considerado seguro quando gera uma quantidade finita de fatos. Porém, a definição da segurança de um conjunto de regras é tido como um problema aberto, já que é muito difícil definir com exatidão se uma regra irá gerar ou não fatos infinitos.

Para que uma regra, seja, então considerada segura, é aconselhado que todas as suas variáveis sejam limitadas. Uma variável limitada é uma variável que aparece no cabeçalho e no corpo de um predicado; aparece no corpo de um predicado onde seja relacionado com constantes; e aparece no corpo de um predicado da forma X=Y ou Y=X, sendo Y uma variável limitada (Elmasri e Navathe, 2005).

2.6 Aplicação

Segundo Huang et al. (2011), a aplicação do Datalog é adequada especialmente em três aplicações, descritas a seguir:

2.6.1 Integração de Dados

Um dos objetivos da integração de dados é reunir várias bases de dados, com esquemas e representações diferentes entre si, e utilizá-los como uma unidade integrada. No cenário clássico, são informados aos esquemas fonte o esquema alvo, esquemas de mapeamento entre a fonte e o alvo, além de uma instância do banco de dados fonte. Na integração de dados chamada virtual, é passada uma consulta sobre o esquema alvo, com o objetivo de que ela seja reformulada e respondida ao longo dos bancos de dados fonte. Na troca de dados, o objetivo é computar uma instância do banco de dados alvo para que as consultas sejam respondidas diretamente.

Os mapeamentos de esquemas, geralmente, são feitos a partir da geração de dependências entre tuplas, o que pode ser traduzido em regras Datalog, que podem ser

enriquecidas com outras tecnologias para representar as variáveis do mapeamento. Tanto a integração de dados materializada quanto a virtual se tornam casos especiais da evolução de consultas em Datalog. Porém, é necessário se certificar de que os programas Datalog sejam finitos.

2.6.2 Rede Declarativas

As redes declarativas são baseadas no fato de que as consultas recursivas são ótimas formas para representação de protocolos de rede, sendo esse protcolos baseados em relações recursivas entre os nós da rede. As tabelas de informações geradas pelos protocolos podem ser vistas como consultas recursivas distribuídas entre as mudanças de estado da rede de entrada, e seus resultados devem ser mantidos consistentes a cada mudança de estado da rede.

2.6.3 Análise de Programas

A área de análise de programas abrange desde fluxo de dados, fluxo de controle, à análise de pontos de função e até código estático. Os resultados dessas análises são usados, por exemplo, para otimização, descoberta de problemas, detecção de erros e aplicação de padrões. Essas análises são naturalmente recursivas, proporcionando uma possibilidade para a utilização do Datalog.

2.6.4 Sistemas Acadêmicos e Comerciais

Entre os sistemas comercias que utilizam Datalog destaca-se o LogicBlox, um sistema que oferece suporte para detectar prováveis erros de programação, otimização de consultas, modularização, e meta-programação - técnica de reuso que se provou bastante útil em segurança e processamento distribuído de consultas. O LogicBlox suporta, também, uma linguagem de atualização, termos *Skolem*, funções definidas pelo usuário, e restrições de integridade.

É importante ressaltar dois sistemas acadêmicos muito relevantes no auge da pesquisa com Datalog: o Coral e o LDL++. Ambos suportam formas avançadas de recursão utilizando negação. Além disso, o LDL++ suporta uma sintaxe para a definição

de agregação pelos usuários e uma forma limitada de meta-programação.

2.7 Datalog x Prolog

O Prolog, assim como o Datalog, é uma linguagem de programação baseada em lógica matemática, formada por fatos e regras. O Prolog implementa algumas estruturas de dados, como listas. Já o Datalog apenas executa consultas aos dados e os recupera da mesma forma em que eles foram armazenados nos fatos, além de ser diferente do Prolog por não implementar estruturas de dados. Do ponto de vista da sintaxe, o Datalog pode ser considerado como um subconjunto da linguagem Prolog, já que toda cláusula Datalog pode ser transformada e executada por um interpretador Prolog (Stefano et al., 1989).

Diferentemente do Prolog, o Datalog não permite regras que contenham operações aritméticas com os argumentos (Elmasri e Navathe, 2005).

Enquanto Datalog possui apenas semântica declarativa, os programas Prolog são construídos a partir de uma semântica operacional, onde seu significado reflete como ele deve ser executado. O processamento de um programa Prolog é feito utilizando-se a abordagem com *backtracking* em profundidade para a construção das árvores, respeitando a ordem em que as cláusulas e os literais devem aparecer. O grande problema dessa abordagem é que ela não garante que o programa será finito (Stefano et al., 1989).

Já os programas Datalog possuem uma estratégia de execução parecida com a estratégia *bottom-up*, o que permite que as regras e fatos não tenham que estar em sequência para que sejam interpretados corretamente. É essa a principal característica que permite a existência de consultas recursivas (Elmasri e Navathe, 2005).

O Datalog possui, também, suporte à negação utilizando a HMF e à recursão, o que não acontece no Prolog. A recursão se mostra muito importante para a implementação de pesquisas e para a realização de inferência sobre os dados (Stefano et al., 1989).

Outra diferença entre as duas linguagens é a disposição das regras no programa: para o Datalog, a ordem das cláusulas é irrelevante - podem ser colocadas em qualquer ordem e o funcionamento do programa não será alterado. Porém, em um programa Prolog, as cláusulas devem ser colocadas na sequência em que elas devem ser executadas (Elmasri e Navathe, 2005).

Por fim, ao se conectar o Prolog a uma base de dados, a abordagem de acesso a esses dados por parte do interpretador Prolog é de "uma tupla por vez", ineficiente quando se trata de uma base que armazena um grande volume de dados. O Datalog, por sua vez, é uma linguagem de consulta própria para bancos dados - sendo então mais eficiente especialmente quando há uma grande quantidade de dados armazenada (Stefano et al., 1989).

2.8 Conclusão

Datalog se mostrou uma linguagem com sintaxe simples mas com características interessantes que lhe conferem um grande poder de processamento.

O Datalog é uma linguagem própria para consultas em banco de dados e dada a crescente utilização de documentos semiestruturados para a troca de informações entre bases de dados diferentes, esses dados podem ser melhor aproveitados se forem tratados por uma linguagem que possa explorar todas as suas propriedades.

A seguir, será descrita a utilização do Datalog para consultas em documentos semiestruturados, bem como o uso de inferência para mapear a evolução desses documentos ao longo do tempo.

3 Evolução de Documentos Semiestrutrados

Nesse capítulo, são definidos os conceitos de Gerência de Configuração: como se deu seu surgimento, suas perspectivas de análise, sua aplicação para dados semiestruturados, trabalhos relacionados e qual será a estratégia utilizada para a realização de inferências sobre esses dados.

3.1 Introdução

A Gerência de Configuração (GC) surgiu na década de 50, quando houve a necessidade de controle das modificações na documentação de aviões de guerra e naves espaciais pela indústria aeroespacial norte-americana (Estublier, 2000). Nas décadas de 60 e 70, foram incorporados os artefatos de software aos artefatos de softwares já estabelecidos, surgindo a Gerência de Configuração de Software (GCS) (Thayer e Christenser, 2002).

Quando foi criada, a GCS era aplicada apenas a aplicações militares e espaciais. A partir da década de 80, com o surgimento da primeira norma IEEE Std 828 (IEEE, 2005) e o aparecimento de softwares cada vez mais complexos, a GCS foi incorporada ao processo de desenvolvimento de softwares não-militares.

Com o aumento da complexidade dos sistemas, a GCS é cada vez mais necessária para o acompanhamento de todas das etapas do desenvolvimento, além de auxiliar na qualidade final do produto.

3.2 Gerência de Configuração de Software

Há diversos conceitos considerados adequados para a GCS, sendo um deles "uma disciplina que aplica procedimentos técnicos e administrativos para identificar e documentar as características físicas e funcionais de um item de configuração (IC), controlar alterações nessas características, armazenar e relatar o processamento das modificações e o estágio da implementação e verificar a compatibilidade com os requisitos especificados"(IEEE,

1990).

A GCS atua em todo ciclo de vida do software, acompanhando sua evolução, e documentando os motivos e impactos causados pelas ações tomadas durante essa evolução. Suas metas incluem o aumento da produtividade e a diminuição de erros na produção de software. A aplicação de técnicas eficientes de GCS ao longo do processo de desenvolvimento promove o acompanhamento sistemático da evolução do sistema, além possibilitar a análise e a manipulação de soluções direcionadas e refinadas, agregando qualidade ao processo de desenvolvimento. Porém, é importante se ter em mente que o objetivo da GCS não é propor maneiras de se executar as modificações, mas apenas acompanhar e controlar os motivos que levaram às mudanças e os impactos por ela gerados (Cunha e Garcia, 2010).

A análise da GCS pode ser efetuada a partir de duas perspectivas diferentes: a perspectiva gerencial e a perspectiva de desenvolvimento (Murta, 2006).

Na perspectiva geral, a GCS é subdividida em: identificação da configuração, onde são definidos, nomeados, numerados e descritos os ICs; função de controle da configuração, onde é acompanhada a evolução dos ICs; função de contabilização da situação da configuração, onde são armazenadas e disponibilizadas informações geradas pelas outras funções; função de revisão e avaliação da configuração, onde é feita a auditoria funcional e física da linha-base; e a função de gerenciamento de liberação e entrega, responsável pela construção e liberação dos ICs (Murta, 2006).

Já na perspectiva de desenvolvimento, a GCS é subdividida em três subsistemas: o Sistema de Controle de Versões, o Sistema de Controle de Modificações e o Sistema de Controle da Construção (Murta, 2006).

3.2.1 Sistema de Controle de Versões

O Sistema de Controle de Versões tem a função de, através da combinação entre procedimentos e ferramentas, identificar, armazenar e controlar os IC e sua evolução ao decorrer do projeto. É também responsável por conduzir as modificações de forma distribuída, concorrente e disciplinada, de forma a não permitir a corrupção do sistema como um todo (Murta, 2006). Exemplos de sistemas de Controle de versões mais utilizados podem

ser citados o Subversion (Collins-Sussman, 2002), o GIT (Chacon, 2010) e o Mercurial (Selenic, 2011).

3.2.2 Sistema de Controle da Construção

O Sistema de Controle da Construção é responsável pela disponibilização de versões do software para os usuários. Para isso, ele utiliza ferramentas para a automatização do processo de agrupamento dos itens de configuração para sua posterior transformação em executáveis. As ferramentas para automatizar a distribuição de *releases* são especialmente úteis quando se há a disponibilização de pequenas *releases* por vez (Murta, 2006).

Como exemplos desses sistemas podem ser citados o Maven (Apache, 2011) e o Ant (Apache, 2000).

3.2.3 Sistema de Controle de Modificações

O Sistema de Controle de Modificações é responsável por documentar as mudanças que ocorreram nos itens de configuração. Devem ser documentados o motivo pelo qual a mudança foi realizada, a data em que ocorreu, qual membro da equipe foi o responsável, e outras informações que forem julgadas como necessárias para que se tenha total controle dos ICs envolvidos no projeto (Murta, 2006).

O foco inicial dos sistemas de controle de modificações era o gerenciamento de modificações corretivas. Porém, com o crescimento e amadurecimento da área, eles passaram a ser utilizados em todas as etapas do desenvolvimento, especialmente quando o processo de desenvolvimento privilegia pequenas liberações, como em metodologias ágeis de desenvolvimento de software (Murta, 2006).

Dentre os sistemas de controle de modificações mais conhecidos e utilizados para desenvolvimento de software livre está o Bugzilla (Mozilla, 2011). Através do Bugzilla, é possível realizar busca detalhada de modificações, criar vínculos entre modificações, controlar o estado das modificações, o relacionamento entre modificações e os artefatos alterados propriamente ditos. Além disso, uma preocupação constante no projeto é a necessidade de um alto desempenho da ferramenta. Para atingir esse objetivo, o Bugzilla faz uso de banco de dados relacional e interfaces HTML (Hypertext Markup Language)

(W3C, 1996).

Além do Bugzilla, existem alguns outros sistemas de controle de modificações, como o Scarab (Silveira, 2007) cujo diferencial é o fato de ter sido implementado na linguagem Java e o ClearQuest (White, 2000) que faz parte da iniciativa da IBM Rational de prover um ambiente integrado de GCS.

Apesar da diversidade de sistemas de controle de modificações já existentes, essa é uma área ainda carente em pesquisas que façam uso da infraestrutura existente com o objetivo de atingir maior qualidade e produtividade no desenvolvimento de software (Murta, 2006).

3.3 Gerência de Configuração de Dados Semiestruturados

Apesar do crescimento de dados eletronicamente compartilhados, nem todos são coletados e inseridos em bancos de dados estruturados cuidadosamente projetados. As grandes empresas, em geral, gerenciam vários repositórios heterogêneos de dados, e as várias aplicações que necessitam utilizar esses dados executam requisições utilizando diferentes modelos e mecanismos de acesso (Saccol e Heuser, 2003).

Para o acesso a este tipo de informação, foram propostos novos modelos de dados, surgindo os dados semiestruturados. Esses dados podem ser coletados sem que se saiba como serão armazenados e gerenciados. Além disso, apresentam uma estrutura diferente, composta por atributos heterogêneos. São portanto mais adaptáveis a situações reais (Elmasri e Navathe, 2005).

Nos dados semiestruturados, o esquema de representação está presente (de forma explícita ou implícita), juntamente com o dado - ou seja, o mesmo é autodescritivo. Atributos podem ser compartilhados entre várias entidades ou existir apenas em algumas. Atributos adicionais podem ser introduzidos em itens de dados mais novos a qualquer momento, e não há nenhum esquema predefinido (Elmasri e Navathe, 2005).

Assim como os dados armazenados em bancos de dados estruturados, os dados semiestruturados evoluem ao longo do tempo. As modificações realizadas podem gerar

inconsistência, já que as instâncias podem se tornar incompatíveis com as definições mais recentes dos esquemas. Em bancos de dados estruturados, modificações que levem a base a um estado inconsistente são bloqueadas pelo sistema gerenciador. Nos modelos de dados semiestruturados, onde não há um sistema gerenciador, modificações no esquema não podem ser bloqueadas em função das instâncias existentes (White, 2000).

Existem diversas formas de representação para dados semiestruturados. Para o desenvolvimento desse trabalho, a representação utilizada foi a oferecida pela linguagem XML (eXtensible Markup Language). O padrão XML é baseado em texto simples para a representação de informações estruturadas, como documentos e configurações. Arquivos XML são muito utilizados para trocas de dados entre aplicações de naturezas diferentes, o que é facilitado pela sua portabilidade (W3C, 1996).

Além disso, o padrão XML oferece uma melhor qualidade no serviço de trocas de dados, permite a utilização de linguagens de consulta e facilita a integração de dados semânticos. Por esses motivos, é largamente utilizado na Internet quando há necessidade de se fazer o transporte de dados pela rede (Cobena et al., 2002).

Porém, assim como artefatos de um ciclo de desenvolvimento de software, os dados representados em arquivos XML também sofrem alterações ao longo do tempo. Muitas vezes, se faz necessário analisar essas mudanças, não apenas descobrindo onde elas ocorreram, mas, também, o significado delas para o conjunto de dados representado (Elmasri e Navathe, 2005).

Mesmo após os dados semiestruturados se tornarem populares, especialmente com a utilização de arquivos XML para representá-los, a maioria das ferramentas existentes para GCS não oferecem suporte a esses dados, tratando arquivos XML da mesma forma como são tratados arquivos de texto comuns. Por esse motivo, são perdidas muitas informações relevantes presentes nesses arquivos, que poderiam resultar em mapeamentos importantes a respeito do comportamento dos dados e do motivo pelo qual eles sofreram alterações.

3.4 Gerência de Modificações de Dados Semiestruturados

A detecção de mudanças é uma área de estudo que tem impacto em diversas aplicações, especialmente aquelas que lidam com grandes volumes de dados semiestruturados que sofrem muitas alterações ao longo do tempo, como por exemplo sistemas de folha de pagamento ou dados médicos experimentais de um novo medicamento.

Além disso, sabe-se que, frequentemente, os usuários estão mais interessados em entender o motivo pelo qual os dados sofreram modificações do que com o valor atual que foi atribuído ao dado. Com a Internet trazendo a utilização crescente de documentos HTML (Hypetext Markup Language) (W3C, 1996) e a popularização do padrão XML para a troca de informações, houve a motivação necessária para se estudar formas de versionamento e controle desse tipo especial de dado (Cobena et al., 2002).

A descrição das modificações sofridas por dados é de suma importância para o acompanhamento de sua evolução. Dada a grande capacidade de representação obtida ao se usar dados semiestruturados, é possível, a partir desse acompanhamento, obter uma gama de diferentes análises úteis a vários contextos.

Nesse contexto, a proposta deste trabalho é propor uma forma de gerenciamento de modificações em arquivos XML. Para isso, a estratégia utilizada será realizar inferências sobre os dados descritos pelo arquivo para, então, extrair novas informações além das já disponíveis.

Estão sendo realizadas pesquisas com outras formas de inferência para a realização da gerência de configuração de modificações. Além do Datalog, que será a estratégia sugerida nesse trabalho, estão sendo estudadas estratégias de inferência com o uso de Prolog e Ontologias.

3.4.1 Prolog

O Prolog é uma linguagem de programação, baseada em lógica matemática, onde programas são escritos como regras para verificar relações entre objetos. Uma de suas características mais importantes é sua maturidade com relação às novas linguagens para

inferência (Cunha et al., 2007).

Gazzola (2011) desenvolveu um módulo onde é utlizado o Prolog para realizar inferência em dados semiestruturados e extrair informações a partir desses dados. Essa funcionalidade foi adicionada a ferramenta XPerseus, que é uma ferramenta acadêmica desenvolvida por (Silva, 2011) que já realiza o controle de versão de dados semiestruturados. O módulo implementado realiza de forma automatizada a transformação dos arquivos XML de entrada em fatos e o processamento de regras de inferência sobre esses fatos.

Sobre a utilização de Prolog, um ponto negativo é a disposição das regras no programa as quais devem ser colocadas na sequência em que elas devem ser executadas. Outro ponto é que a abordagem de acesso a dados por parte do interpretador Prolog é de uma tupla por vez, o que se torna ineficiente quando a base em questão armazena um grande volume de dados.

3.4.2 Ontologias

Na área da computação, ontologias são uma forma concreta com a qual se torna possível a representação do conhecimento. Assim como dados semiestruturados, modificações em ontologias ao longo do tempo acarretam sua evolução. Para realizar inferência em ontologias, umas das possibilidades é a utilização a SWRL (Semantic Web Rule Language), juntamente com *reasoners*, que são responsáveis por processar os dados representados na ontologia de acordo com as regras escritas em SWRL.

Está sendo realizado um estudo por membros do Grupo de Educação Tutorial do curso de Ciência da Computação da UFJF em que regras SWRL são utilizadas para a inferência de dados representados por ontologias. Para a criação e edição de ontologias, é usado o Protegé, um editor que permite a manipulação de ontologias e suas instâncias. Foi incorporado ao Protegé[1] o *reasoning* Jess[2], que é responsável por processar as regras de inferência escritas em SWRL e recuperar e extrair informações relevantes presentes nas ontologias. Outro *reasoning* que, como o Jess, pode ser utilizado em conjunto com o

[1] http://protege.stanford.edu/
[2] http://www.jessrules.com/

Protegé é Pellet[3].

Um ponto negativo sobre a utilização de ontologias é que estas novas linguagens (OWL, SWRL) e seus raciocinadores ainda não possuem a mesma maturidade alcançada por Prolog e Datalog. A OWL possui construtores que permitem a escrita de predicados unitários (pertencer a uma classe) e predicados binários (relações). No entanto, a OWL somente permite inferir predicados unitários a partir de seus axiomas. Semanticamente, isto significa que permite apenas a inferência para classificação. Esta capacidade limitada não explora toda a riqueza do conhecimento do domínio, uma vez que o trabalho de classificar já está pronto - e por isso não foi considerada neste trabalho.

3.4.3 Datalog

Como mencionado anteriormente, tanto Datalog quanto Prolog são linguagens de programação matemática, formada por fatos e regras. A vantagem do primeiro em relação ao segundo reside no fato de ser possível a realização de consultas recursivas e a implementação de sentenças de negação. Além disso há mais garantias de se construir programas finitos (Huang et al., 2011).

Na seção seguinte, será detalhada a forma de uso de Datalog como estratégia para inferência de dados e sua utilização na gerência de modificação de dados semiestruturados.

3.5 Uso de Inferência com Datalog

Realizar inferência sobre um conjunto dados significa analisar esses dados e, a partir dessa análise, extrair novas informações relevantes para o contexto em que eles estão inseridos. Ela se faz muito útil ao se analisar a evolução de dados, pois, ao se construir regras de inferência, é possível se fazer uma análise direcionada a algum ponto ou característica que se deseja destacar (Elmasri e Navathe, 2005).

A inferência se dá através de regras, que devem ser escritas de modo que seu retorno expresse qual é a informação que se deseja descobrir.

O objetivo deste trabalho é, através da utilização regras de inferência, propor

[3]http://clarkparsia.com/pellet/

uma estratégia para a realização do controle de modificações em dados semiestruturados. As operações necessárias para a aplicação das regras de inferência são executadas sobre os dados puros, sem nenhuma manipulação prévia.

Para a realização da inferência, são utilizados dois arquivos contendo dados com a mesma estrutura. Contudo cada um desses arquivos conterá valores diferentes para esses dados. É feita, então, uma comparação entre os valores a partir de regras de inferência pré-definidas. O resultado será a recuperação de novas informações, construídas a partir da interpretação dos valores presentes nos dados.

A linguagem Datalog foi utilizada para a construção das regras de inferência. Por ser uma linguagem de consulta baseada em programação lógica, Datalog oferece uma grande expressividade para essas regras, tornando possível várias combinações entre os elementos dos dados semiestruturados para a extração de informações (Stefano et al., 1989).

Porém, antes que as regras de inferência possam ser aplicadas aos dados, eles são convertidos em fatos - afirmações sobre as quais regras são aplicadas. Fatos também serão retornados após a aplicação de regras (Korth e Silberschartz, 2006).

A inferência sobre as diferentes versões dos dados resulta na descoberta de novas informações. É possível descobrir o motivo pelo qual as modificações nos dados foram realizadas e qual foi o impacto que elas causaram.

Para o usuário, essas novas informações tem, por exemplo, a capacidade de auxiliar análises mais robustas do funcionamento de uma aplicação. Através da interpretação dos dados torna-se possível entender o que de fato está acontecendo, diagnosticar problemas, gerar estatísticas, acompanhar modificações, entre outras coisas.

A seguir, são apresentados trabalhos relacionados a esse, que também estudam formas para realizar o controle de modificações em dados semiestruturados.

3.6 Trabalhos Relacionados

Existem várias pesquisas conduzidas por diferentes áreas com o objetivo de propor estratégias para a detecção de mudanças em dados semiestruturados.

Cobena et al. (2002) desenvolveram um projeto chamado Xyleme, que propõe a criação de um *data warehouse* que seja capaz de armazenar um grande volume de dados no padrão XML. O grande problema a ser resolvido nesse trabalho foi a criação de um algoritmo que calculasse a diferença entre os arquivos XML com grande eficiência. O algoritmo que representa a solução encontrada para o cálculo desse delta tem o seguinte funcionamento: incialmente, ele procura pelas árvores que não sofreram alterações; depois, trata caso a caso onde ocorreram mudanças - por exemplo, árvores inteiras, apenas filhos ou atributos. Com a aplicação de técnicas de otimização, foi possível ainda aumentar a eficiência do algoritmo. A partir do delta já calculado, foi implementado um sistema que permite monitorar modificações e extrair informações sobre essas modificações. As principais vantagens oferecidas por esse trabalho são: possibilidade de consultas a versões antigas de dados, a utilização do algoritmo para a descoberta e descrição das modificações para auxiliar na resolução de conflitos, gerenciamento das modificações e indexação dos dados.

Porém, o citado trabalho não levou em consideração o significado semântico das modificações, nem seu impacto no conteúdo dos dados. Se, ao se ter o controle sobre a modificação de dados, fosse possível avaliar as mudanças do ponto de vista do seu significado,seria possível possibilitar aos usuários diversas análises e relatórios sobre a evolução de um sistema, e a parte do mundo real que ele representa.

Já o trabalho de Zhao et al. (2004) propõe um modelo para a descoberta de modificações frequentes em arquivos XML, H-DOM (Historical-Document Object Model), e apresenta dois algoritmos básicos que, percorrendo apenas duas vezes uma sequência XML, é capaz de descobrir todas as mudanças que ocorreram nessa sequência. O foco desse trabalho é analisar as FCS *(Frequently Changed Structures)* e investigar o histórico das modificações que ocorreram nessas estruturas para extrair dados relevantes acerca das modificações que ocorreram ao longo desse histórico. Ao se considerar cada delta como uma transação, e numa escala maior se associar regras a essas transações é possível a partir da associação entre os diferentes deltas contendo FCS monitorar e prever as modificações que estão ocorrendo, além de se aumentar a escalabilidade dos sistema de controle de modificações para dados semiestruturados.

No trabalho citado, há uma junção do controle de versões com o controle de modificações, sendo o foco dessas operações as partes dos documentos que mais sofrem mudanças. Assim como no trabalho aqui desenvolvido, é levado também em conta o aspecto semântico das alterações. Porém, informações importantes podem ser perdidas ao se considerar apenas parte dos dados, já que uma alteração em outro local pode ser essencial para se entender o significado do conjunto de modificações.

Há, também, o trabalho desenvolvido por Lim e Ng (2004) cujo foco está em dados representados no formato HTML. É apresentado um algoritmo semântico de detecção de diferenças, chamado SCD (Semantic Change Detection), que é baseado nas modificações semânticas entre os dados e no conceito de hierarquia semântica. A hierarquia semântica é construída da seguinte forma: primeiro são removidas dos dados as *tags* de formatação e depois, é definida uma precedência dos dados. Essa abordagem, além de ser a primeira a considerar a hierarquia semântica entre os dados, é muito útil para os usuários cuja principal preocupação é detectar as modificações nos documentos. Além disso, para a execução do SCD não é necessário nenhum tipo de pré-processamento nem o conhecimento da estrutura interna dos documentos a serem analisados, e podem ser utilizados quaisquer dois arquivos que sofrem alterações frequentes e cujas alterações se deseja monitorar.

O trabalho citado propõe um algoritmo de detecção de mudanças que além de controlar as alterações ocorridas, se preocupa em descobrir o seu significado. Porém, esse algoritmo é voltado para dados descritos em HTML, deixando de lado outros padrões que se mostram igualmente importantes para a representação de dados semiestruturados.

3.7 Conclusão

A GCS sempre foi uma parte importante no processo de desenvolvimento de software auxiliando na manutenção dos artefatos. Com o crescente aumento na complexidade dos sistemas, ela passou a ser indispensável para o bom andamento da evolução dos sistemas.

Com a popularização da Web, houve um aumento exponencial da utilização de dados semiestruturados representados na Web, em sua maioria, por pelos padrões HTML e XML. Juntamente com essa popularização, surgiu a necessidade da aplicação das técnicas de GCS já existentes a esse tipo especial de dado e suas particularidades.

4 Estudo de Caso

Neste capítulo, é apresentada uma abordagem para a inferência de dados em documentos semiestruturados, onde as regras de inferências são escritas na linguagem Datalog, e os dados semiestruturados são transformados em fatos Datalog. O interpretador LogicBlox processa as regras sobre os fatos e retorna as respostas na forma de afirmações.

4.1 Visão Geral

Os documentos semiestruturados armazenam mais informações que apenas as especificadas explicitamente por seus elementos e atributos - e talvez até mais valiosas. São essas informações as possíveis de serem obtidas através da utilização de inferência.

Para realizar a inferência sobre os dados, podem ser utilizadas várias abordagens. A abordagem escolhida nesse trabalho foi a utilização da linguagem de consulta Datalog, por sua simplicidade e seu grande poder de expressão (Stefano et al., 1989).

Para a compilação das regras Datalog, foi utilizado o interpretador LogicBlox[1].

4.2 LogicBlox

O LogicBlox é uma plataforma para desenvolvimento de aplicações em Datalog. Suporta a criação de aplicações robustas para empresas, como aplicações para planejamento e controle de preços. É comumente utilizado em aplicações onde é necessária a tomada de decisões. Suas aplicações geralmente envolvem grande complexidade, juntamente com a manipulação de grandes quantidades de dados. Além disso, permite um rápido desenvolvimento de aplicações SaaS (Software as a Service) (Campagna et al., 2011).

Quanto à sua licença, o LogicBlox é uma aplicação comercial, mas contém uma distribuição de 2007 que é livre para a realização de pesquisas. Os pesquisadores podem, se desejarem, contribuir com suas idéias para o aprimoramento da plataforma (Bravenboer

[1] http://www.logicblox.com

e Smaragdakis, 2009).

A linguagem suportada pela plataforma é uma variante do Datalog puro denominada $Datalog^{LB}$. Esta linguagem que é baseada em avaliação incremental, com funcionalidades no estilo de triggers, oferecendo suporte a atualizações dinâmicas, especificação declarativa de dependência funcional, escolhas não determinísticas, negação, metaprogramação, bem como outros recursos, tais como agregação utilizada por abordagens de otimização (Campagna et al., 2011).

Outras adições feitas ao Datalog pelo LogicBlox são a implementação da negação em regras e a interpretação de algumas regras como funções (Bravenboer e Smaragdakis, 2009).

A figura 4.1 mostra a arquitetura utilizada para a construção do LogicBlox.

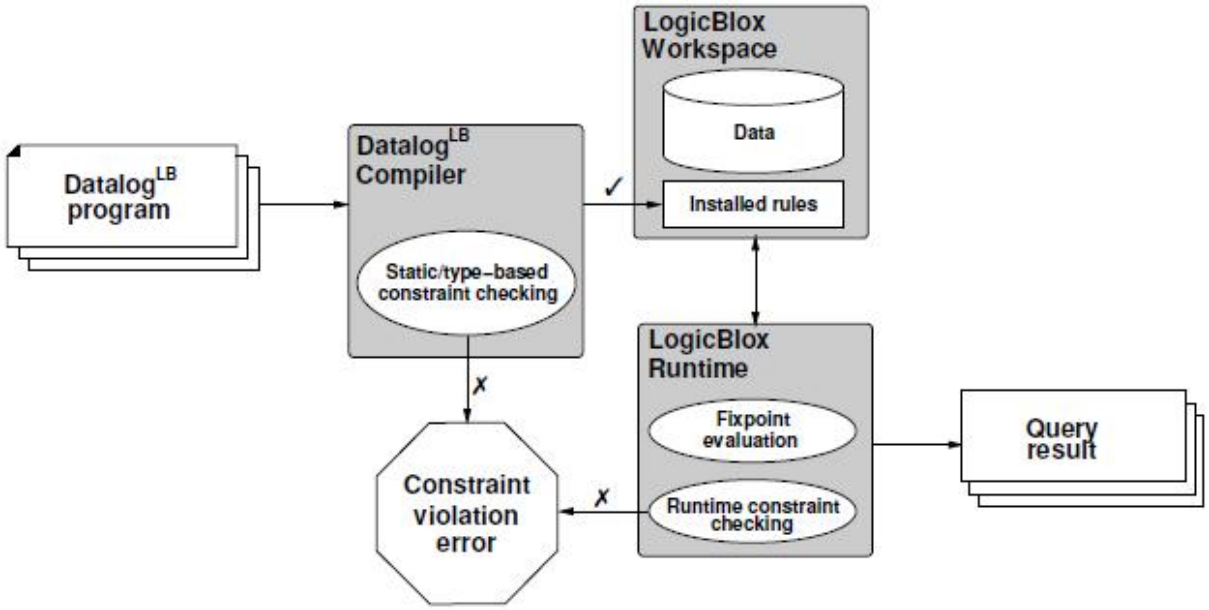

Figura 4.1: Arquitetura do LogicBlox (Marczak et al., 2009)

Como se pode ver pela figura 4.1, o LogicBlox é composto por um compilador *Datalog Compiler*, que verifica o programa Datalog com relação aos tipos e restrições. Se o programa falhar, ele emite os erros para o usuário. Senão, o programa passa para o *LogicBlox Workspace* - uma instância de banco de dados responsável por armazenar os dados a serem analisados, além de algumas regras e restrições já instaladas. A partir do *workspace*, é permitido o acesso aos dados definidos nesse banco de dados para consulta e modificação. Após a interpretação dos dados, no LogicBlox *Runtime*, são verificadas

novamente as restrições e, caso não haja nenhum erro, é retornado o resultado da consulta solicitada pelo programa Datalog fornecido como entrada (Marczak et al., 2009).

Ao longo da execução do programa pelo *LogicBlox* e, de acordo com as modificações feitas pelo usuário no programa Datalog, as regras são processadas novamente até que nenhum novo fato possa ser derivado. Ao mesmo tempo, são verificadas as restrições do programa, reportando erros e mantendo a consistência do *workspace* (Campagna et al., 2011).

4.3 Abordagem Proposta

Na figura 4.2, é mostrada a abordagem proposta nesse trabalho. A transformação dos documentos de entrada em fatos Datalog é efetuada manualmente, pois o objetivo desse trabalho é a investigação da viabilidade de se utilizar a linguagem para realizar inferência de dados. Esse processo de tradução gera vários fatos a partir de um documento XML, transformando os elementos em predicados e seus conteúdos em constantes. Para relacionar predicados distintos, um identificador (uma constante) é acrescentado como parâmetro, caracterizando que há correspondência entre esses dados no documento XML (por exemplo, relação de pai/filho). Nesta transformação, gera-se um único conjunto de fatos, baseados em informações de duas versões de um documento (v1 e v2), onde a ordem com que os fatos estão dispostos é irrelevante. Assim, é possível comparar as informações dos arquivos utilizando este único conjunto.

Após a obtenção dos fatos (1) e considerando-se as regras de negócio relativas ao domínio da aplicação (2), a máquina de inferência atua sobre os fatos (3). Após a aplicação de um conjunto de regras, pode-se obter a evolução do arquivo XML base para o XML modificado (4). Assim, dada a entrada composta por duas versões de documentos XML e um conjunto de regras de negócio e de inferência relativos a um domínio específico, pode-se inferir a evolução dos documentos XML. O ambiente pode ser generalizado, apenas especificando as regras de acordo com o domínio.

Os documentos XML são convertidos em fatos. Os fatos gerados representam o documento XML, somente com uma identificação da raiz do XML associada à versão (v1 ou v2). Assim, os documentos (v1 e v2) são transformados em um mesmo conjunto de

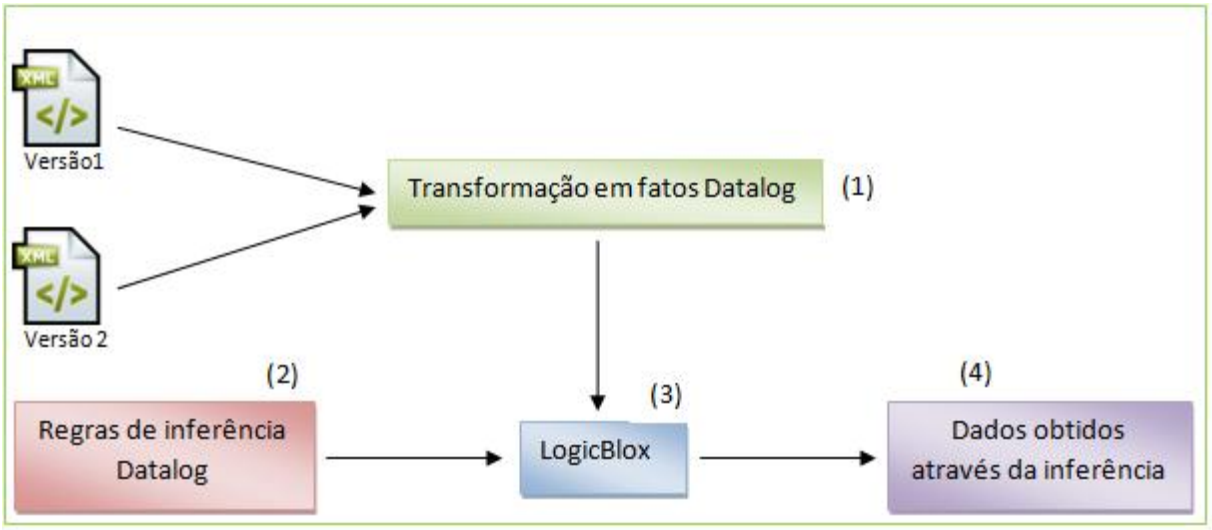

Figura 4.2: Abordagem utilizada.

fatos, possibilitando a inferência.

Para o processamento das regras e fatos, foi utilizado um arquivo ".bat", no Windows. Esse arquivo contem os seguintes comandos:

- bloxbatch -db database -create -overwrite -blocks base: responsável por criar a instância do banco de dados onde são armazenados os fatos.
- bloxbatch -db database -addBlock -file regras.logic: responsável por adicionar as regras ao banco de dados.
- bloxbatch -db database -execute -file fatos.logic: responsável por adicionar os fatos ao banco de dados.
- bloxbatch -db database -print nomeRegra: responsável por exibir o resultado da aplicação das regras. Para cada regra, é necessária uma linha dessa no arquivo .bat.

A seguir, é apresentado o estudo de caso realizado especificamente para esse trabalho.

4.4 Estudo de Caso

Nesta seção, é apresentado o estudo de caso desenvolvido nesse trabalho para exemplificar a utilização de Datalog com inferência de dados e a sua relação com o controle de modificações de dados semiestruturados.

Inicialmente, são selecionados dois arquivos XML contendo informações relativas ao salário, nome, cargo, filial e departamento a que pertencem funcionários cadastrados em uma empresa fictícia, como pode ser visto na listagem 4.1. Os dados contidos nos arquivos são os seguintes:

- Um elemento chamado empresa, que é o pai de todos os elementos;
- Elementos do tipo funcionario, que descrevem as informações sobre os funcionários da empresa;
- Elementos filhos do elemento funcionario, que são: Id, Nome, Salario, Cargo, Filial e Departamento.

Listagem 4.1: Fragmento de Arquivo XML

```
1  <empresa>
2    <nome>Topazio Informatica</nome>
3      <funcionario>
4        <id>1</id>
5            <nome>Pedro</nome>
6            <salario>1000</salario>
7            <cargo>Programador</cargo>
8            <filial>Juiz de Fora</filial>
9            <departamento>Informatica</departamento>
10     </funcionario>
```

São utilizados dois arquivos com os mesmos dados, porém com versões diferentes.

A partir dos arquivos XML, são construídos os fatos que descrevem os dados contidos nesses arquivos. Isso é feito de acordo com a sintaxe exigida pelo LogicBlox para a interpretação das regras. Além disso, o LogicBlox implementa a definição de

tipos estáticos para as componentes atômicas dos fatos. As componentes utilizadas estão representadas na listagem 4.2.

Listagem 4.2: Definição dos Componentes dos Fatos

```
1 empresa(emp) -> string(emp).
2 funcionario(emp, id) -> empresa(emp), int[32](id).
3 nome(id, valor) -> int[32](id), string(valor).
4 salario(id, valor) -> int[32](id), int[32](valor).
5 cargo(id, valor) -> int[32](id), string(valor).
6 filial(id, valor) -> int[32](id), string(valor).
7 departamento(id, valor) -> int[32](id), string(valor).
```

Essa definição de tipos estáticos promove uma validação na entrada dos fatos. Por exemplo: para que seja criado um funcionário, no fato em que se relaciona o funcionario com seu identificador é necessário informar uma *string* que corresponde ao nome do funcionário (linha 2). Se o valor informado for de outro tipo, como um inteiro, será informado ao usuário um erro indicando a incompatiblidade entre os tipos.

Em conjunto com a definição dos componentes dos fatos, é feita a definição das regras (Listagem 4.3) - onde pode ser verificado, em um contexto de funcionários de uma empresa, se um determinado funcionário recebeu aumento (linhas 1 a 5). Isso é considerado verdade quando o conteúdo do elemento salário da versão 1 do documento XML é menor do que na versão 2. Se o funcionário, além de ter recebido um aumento de salário, mudar de cargo, isto implica em uma promoção (linhas 7 a 11). O funcionário pode, simplesmente, mudar de cargo (linhas 13 a 17); ou ser transferido - o que implica em uma mudança de filial (linhas 19 a 22) -, ou mudar de departamento (linhas 24 a 27). Por último, tem-se a regra que representa um funcionário que foi promovido e transferido (linhas 29 a 31).

Para facilitar a escrita das regras de evolução, foram criadas algumas regras adicionais que garantem, por exemplo, que ao verificar se um funcionário recebeu aumento, a comparação seja feita realmente entre o mesmo funcionário nas duas versões do arquivo. Essa regras auxiliares podem ser vistas no apêndice.

Listagem 4.3: Regras de Inferência

```
1  recebeu_aumento(Nome) <-
2      mesmo_id(F1,F2,Nome),
3      salario(F1,S1),salario(F2,S2),
4      cargo(F1,C), cargo(F2,C),
5      S1<S2.
6
7  foi_promovido(Nome) <-
8      mesmo_id(F1,F2,Nome),
9      salario(F1,S1),salario(F2,S2),
10     cargo(F1,C1),cargo(F2,C2),
11     S1<S2,C1 != C2.
12
13 nova_funcao(Nome) <-
14     mesmo_id(F1,F2,Nome),
15     mesmo_salario(Nome),
16     cargo(F1,C1),cargo(F2,C2),
17     C1 != C2.
18
19 foi_transferido(Nome) <-
20     mesmo_id(F1,F2,Nome),
21     filial(F1,Fil1),filial(F2,Fil2),
22     Fil1 != Fil2.
23
24 novo_departamento(Nome) <-
25     mesmo_id(F1,F2,Nome),
26     departamento(F1,D1),departamento(F2,D2),
27     D1 != D2.
28
29 foi_promovido_e_transferido(Nome) <-
30     foi_transferido(Nome),
31     foi_promovido(Nome).
```

Tendo em mãos os resultados da aplicação dessas regras, é possível acrescentar significado semântico às alterações ocorridas. Em uma empresa, como a deste exemplo, esses resultados podem gerar relatórios sobre o andamento dos funcionários com relação ao plano de carreira, ou sobre os recursos humanos perdidos através de transferências e promoções. A partir daí, podem ser feitas análises ainda mais profundas com o objetivo de mapear a situação da empresa de acordo com as modificações dos dados ao longo do tempo.

Como resultado da aplicação das regras de inferência, é criado um arquivo de texto contendo as informações novas extraídas dos fatos. O arquivo de saída do estudo de caso apresentado na Listagem 4.4.

Listagem 4.4: Arquivo de Saída

```
1  = = = = = = = = = = = = = = = = = = = =
2   Receberam aumento
3  = = = = = = = = = = = = = = = = = = = =
4     Joao
5     Maria
6     Pedro
7  = = = = = = = = = = = = = = = = = = = =
8  Nova Funcao
9  = = = = = = = = = = = = = = = = = = = =
10    Ana
11    Maria
12    Mateus
13    Pedro
14 = = = = = = = = = = = = = = = = = = = =
15 Novo Departamento
16 = = = = = = = = = = = = = = = = = = = =
17    Ana
18 = = = = = = = = = = = = = = = = = = = =
19 Foram Promovidos
20 = = = = = = = = = = = = = = = = = = = =
```

```
21    Maria
22    Pedro
23 = = = = = = = = = = = = = = = = = = = = =
24 Foram transferidos
25 = = = = = = = = = = = = = = = = = = = = =
26    Ana
27    Mateus
28    Pedro
29 = = = = = = = = = = = = = = = = = = = = =
30 Foram promovidos e transferidos
31 = = = = = = = = = = = = = = = = = = = = =
32    Pedro
```

Com a máquina de inferência trabalhando com as, regras é possível verificar que, por exemplo, João recebeu aumento - pois teve seu salário alterado - e que Ana mudou de função e foi transferida, pois somente seu cargo e filial foram alterados. Ou seja, a versão 2 apresenta a evolução da empresa a partir da versão 1. Normalmente, as abordagens baseadas em diferença identificam a mudança sintática nos documentos. Esta análise no entanto está associada ao siginificado da mudança (João recebeu aumento ou Ana mudou de função e foi transferida).

4.5 Conclusão

Neste capítulo, foi apresentada uma abordagem para a inferência de dados utilizando o Datalog, usada juntamente com o controle de modificações com o intuito de trazer significado às mudanças ocorridas em documentos semiestruturados, sua futura incorporação à ferramenta XPerseus.

5 Considerações Finais

O propósito de um sistema de controle de modificações é registrar a razão de alteração de um artefato, independente de sua natureza - o que torna, a princípio, a proposta de se criar mais um sistema de controle de modificações para dados semiestruturados desnecessária. Optou-se, então, neste trabalho pela investigação de formas para compreender a evolução de documentos XMLs em alto nível. Para tanto, foi proposta uma abordagem para a evolução de versões de um documento XML, que foram transformadas em fatos Datalog o que, a partir de um conjunto de regras, possibilitou inferir a intenção do usuário ao criar a segunda versão. As razões das mudanças podem ser entendidas como a evolução dos documentos.

Datalog se motrou uma linguagem muito útil para a escrita das regras de inferência - embora simples, contém características que oferecem um grande poder de processamento, como o suporte a consultas recursivas e a negação de predicados. Seu ressurgimento pode contribuir em diversas áreas como na área de Redes de Computadores - onde o acesso de nós em uma rede podem ser feitos de forma recursiva - ou em aplicações comerciais, onde é movimentado um grande volume de dados, e há uma clara necessidade de eficiência na execução de consultas. Na verdade, o Datalog pode ser útil em todas as áreas que fazem uso de dados semiestruturados e onde a inferência de dados se mostra útil para auxiliar no Controle de Modificações.

Como trabalho futuro, uma proposta é a implementação de um protótipo em Java para a criação de um ambiente que permita ao usuário acompanhar o processo inteiro de modo mais intuitivo, participando de todas as etapas - incluindo a conversão dos dados presentes nos arquivos XML em fatos Datalog. Outra possibilidade corresponde a uma investigação ampla relacionada ao acompanhamento da evolução da OWL e dos raciocinadores associados, com o intuito de possibilitar a inferência de dados.

Após a construção desse ambiente, é proposta a incorporação do mesmo na ferramenta XPerseus, que é uma ferramenta criada com o objetivo de oferecer suporte ao controle de versões de arquivos XML (Silva, 2011). Estão sendo desenvolvidas, também,

as funcionalidades de mesclagem de dados, além de uma outra forma de inferência de dados, onde as regras e fatos são escritos em Prolog (Gazzola, 2011).

Referências Bibliográficas

Foundation, A. S. **Apache Ant**. http://ant.apache.org/, 2000.

Foundation, A. S. **Apache Maven**. http://maven.apache.org/, 2011.

Bravenboer, M.; Smaragdakis, Y. Exception analysis and points-to analysis: Better together. **ISSTA**, 2009.

Bravenboer, M.; Smaragdakis, Y. Strictly declarative specification of sophisticated points-to analyses. **OOPSLA**, 2009.

Campagna, D.; Sarna-Starosta, B. ; Schrijvers, T. Approximating constraint propagation in Datalog. **CICLOPS**, 2011.

Chacon, S. **ProGit**. Apress, 2010.

Cobena, G.; Abiteboul, S. ; Marian, A. **Detecting changes in xml documents**. In: ICDE. IEEE Computer Society, 2002.

Collins-Sussman, B. The subversion project: buiding a better CVS. **Linux J.**, v.2002, p. 3–, February 2002.

Cunha, A.; Albrecht, F. ; de Araujo Fernandes, R. Q. Inferência sobre ontologias: o reencontro com prolog. 2007.

Cunha, C. O.; Garcia, P. A. **A gerência de configuração no contexto da melhoria do processo de software brasileiro**. Departamento de Ciência da Computação – Universidade Federal de Juiz de Fora, 2010.

Elmasri, R. E.; Navathe, S. **Sistema de Banco de Dados, 4ª Edição**. Editora Addison Wesley, 2005.

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

Gazzola, P. O. L. **Inferência em documentos xml utilizando PROLOG**. Technical report, UFJF, 2011.

Hajiyev, E.; Verbaere, M. ; Moor, O. **Codequest: Scalable source code queries with Datalog.** In: Thomas, D., editor, ECOOP. Springer, 2006.

Huang, S. S.; Green, T. J. ; Loo, B. T. **Datalog and emerging applications: An interactive tutorial**. In: SIGMOD, 2011.

The Institute of Electrical and Eletronics Engineers. **IEEE Standard Glossary of Software Engineering Terminology**, 1990.

The Institute of Electrical and Eletronics Engineers. **IEEE Std 828-2005 (Revision of IEEE Std 828-1998)**, 2005.

Korth, H. F.; Silberschatz, A. **Sistema de Banco de Dados, 2ª Edição Revisada**. Editora Addison Wesley, 2006.

Lima, S. J.; Ng, Y. **Change discovery of hierarchically structured, order-sensitive data in html/xml documents.** In: SAINT. IEEE Computer Society, 2004.

Marczak, W. R.; Huangy, S. S.; Bravenboery, M.; Sherrz, M.; Loo, B. T. ; Arefy, M. Secureblox: Customizable secure distributed data processing. 2009.

Communications, N. **Bugzilla**, 1998.

Murta, L. G. P. **Gerência de Configuração no Desenvolvimento Baseado em Componentes**. PESC/COPPE/UFRJ, Rio de Janeiro, Outubro 2006. Tese de Doutorado - UFRJ.

Nardon, F. B. **Estudo e construção de um sistema gerenciador de banco de dados dedutivo**. 1996. Dissertação de Mestrado - UFRGS.

Saccol, D. B.; Heuser, C. A. Integration of xml data. **Proceedings of the VLDB 2002 Workshop EEXTT and CAiSE 2002 Workshop DTWeb on Efficiency and Effectiveness of XML Tools and Techniques and Data Integration over the Web-Revised Papers**, 2003.

Selenic. **Mercurial**. http://mercurial.selenic.com/, 2011.

Silveira, V. N. K. **X-spread - um mecanismo automático para propagação da evolução de esquemas para documentos xml**. Setembro 2007. Dissertação de Mestrado - UFGRS.

Silva, R. B. **Xperseus: Uma interface gráfica para detecção de diferenças entre documentos xml**. Technical report, UFJF, Julho 2011.

Stefano, C.; Gottlob, G. ; Tanca, L. What you always wanted to know about Datalog (and never dared to ask). **IEEE Trans. Knowl. Data Eng.**, 1989.

Thayer, R. H.; Christensen, M. J. **Project Manager's Guide to Software Engineering's Best Practices**. Los Alamitos, CA, USA: IEEE Computer Society Press, 2002.

W3C.org. **Extensible markup language (xml)**, 1996.

White, B. A. **Software configuration management strategies and rational clearcase**. Addison-Wesley, 2000.

Zhao, Q.; Bhowmick, S. S.; Mohania, M. K. ; Kambayashi, Y. **Discovering frequently changing structures from historical structural deltas of unordered xml.** In: Grossman, D.; Gravano, L.; Zhai, C.; Herzog, O. ; Evans, D. A., editors, CIKM. ACM, 2004.

A Apêndice

Documentos XML nos quais foram baseados os fatos utilizados

Listagem A.1: Primeira Versão dos Dados

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <empresa>
3    <nome>Topazio Informatica</nome>
4      <funcionario>
5        <id>1</id>
6            <nome>Pedro</nome>
7            <salario>1000</salario>
8            <cargo>Programador</cargo>
9            <filial>Juiz de Fora</filial>
10           <departamento>Informatica</departamento>
11     </funcionario>
12     <funcionario>
13           <id>2</id>
14       <nome>Joao</nome>
15           <salario>2000</salario>
16           <cargo>Analista de Sistemas</cargo>
17           <filial>Juiz de Fora</filial>
18           <departamento>Informatica</departamento>
19     </funcionario>
20     <funcionario>
21           <id>3</id>
22           <nome>Mateus</nome>
23           <salario>1900</salario>
24           <cargo>Analista</cargo>
25           <filial>Rio de Janeiro</filial>
26           <departamento>TI</departamento>
```

```
27      </funcionario>
28      <funcionario>
29              <id>4</id>
30              <nome>Ana</nome>
31              <salario>1800</salario>
32              <cargo>Gerente de Marketing</cargo>
33              <filial>Juiz de Fora</filial>
34              <departamento>Publicidade</departamento>
35      </funcionario>
36      <funcionario>
37              <id>5</id>
38              <nome>Maria</nome>
39              <salario>1700</salario>
40              <cargo>Gerente de Recursos Humanos</cargo>
41              <filial>Barbacena</filial>
42              <departamento>Publicidade</departamento>
43      </funcionario>
44  </empresa>
```

Listagem A.2: Segunda Versão dos Dados

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <empresa >
3    <funcionario>
4      <id>1</id>
5          <nome>Pedro</nome>
6          <salario>2000</salario>
7          <cargo>Analista de Sistemas</cargo>
8          <filial>Barbacena</filial>
9          <departamento>Informatica</departamento>
10   </funcionario>
11   <funcionario>
12     <id>2</id>
13         <nome>Joao</nome>
14         <salario>3000</salario>
15         <cargo>Analista de Sistemas</cargo>
16         <filial>Juiz de Fora</filial>
17         <departamento>Informatica</departamento>
18   </funcionario>
19   <funcionario>
20         <id>3</id>
21         <nome>Mateus</nome>
22         <salario>1900</salario>
23         <cargo>Gerente de Projetos</cargo>
24         <filial>Juiz de Fora</filial>
25         <departamento>TI</departamento>
26   </funcionario>
27   <funcionario>
28         <id>4</id>
29         <nome>Ana</nome>
30         <salario>1700</salario>
31         <cargo> Consultora de Marketing </cargo>
```

```
32          <filial>Niteroi</filial>
33          <departamento>Publicidade</departamento>
34    </funcionario>
35    <funcionario>
36      <id>5</id>
37          <nome>Maria</nome>
38          <salario>1700</salario>
39          <cargo>Diretora de Recursos Humanos</cargo>
40          <filial>Juiz de Fora</filial>
41          <departamento>Publicidade</departamento>
42    </funcionario>
43 </empresa>
```

Fatos Datalog obtidos após a conversão:

Listagem A.3: Fatos convertidos

```
1  +empresa("v1").
2
3  +funcionario("v1",1).
4  +nome(1,"Pedro").
5  +salario(1, 1000).
6  +cargo(1, "Programador").
7  +filial(1,"Juiz de Fora").
8  +departamento(1, "Informatica").
9
10 +funcionario("v1",2).
11 +nome(2,"Joao").
12 +salario(2, 2000).
13 +cargo(2, "Analista de Sistemas").
14 +filial(2,"Juiz de Fora").
15 +departamento(2, "Informatica").
16
17 +funcionario("v1",3).
18 +nome(3,"Mateus").
19 +salario(3, 1900).
20 +cargo(3, "Analista").
21 +filial(3,"Rio de Janeiro").
22 +departamento(3, "TI").
23
24 +funcionario("v1",4).
25 +nome(4,"Ana").
26 +salario(4, 1800).
27 +cargo(4, "Gerente de Marketing").
28 +filial(4,"Juiz de Fora").
29 +departamento(4, "Publicidade").
30
```

```
31 +funcionario("v1",5).
32 +nome(5,"Maria").
33 +salario(5, 1700).
34 +cargo(5, "Gerente de Recursos Humanos").
35 +filial(5,"Barbacena").
36 +departamento(5, "Recursos Humanos").
37
38 +empresa("v2").
39
40 +funcionario("v2",1).
41 +nome(1,"Pedro").
42 +salario(1, 2000).
43 +cargo(1, "Analista de Sistemas").
44 +filial(1,"Barbacena").
45 +departamento(1, "Informatica").
46
47 +funcionario("v2",2).
48 +nome(2,"Joao").
49 +salario(2, 3000).
50 +cargo(2, "Analista de Sistemas").
51 +filial(2,"Juiz de Fora").
52 +departamento(2, "Informatica").
53
54 +funcionario("v2",3).
55 +nome(3,"Mateus").
56 +salario(3, 1900).
57 +cargo(3, "Gerente de Projetos").
58 +filial(3,"Juiz de Fora").
59 +departamento(3, "TI").
60
61 +funcionario("v2",4).
62 +nome(4,"Ana").
```

```
63 +salario(4, 1800).
64 +cargo(4, "Consultora de Marketing").
65 +filial(4,"Niteroi").
66 +departamento(4, "Marketing").
67
68 +funcionario("v2",5).
69 +nome(5,"Maria").
70 +salario(5, 2100).
71 +cargo(5, "Diretora de Recursos Humanos").
72 +filial(5,"Barbacena").
73 +departamento(5, "Recursos Humanos").
```

Regras utilizadas

Listagem A.4: Regras

```
1  empresa(emp) -> string(emp).
2  funcionario(emp, id) -> empresa(emp), int[32](id).
3  nome(id, valor) -> int[32](id), string(valor).
4  salario(id, valor) -> int[32](id), int[32](valor).
5  cargo(id, valor) -> int[32](id), string(valor).
6  filial(id, valor) -> int[32](id), string(valor).
7  departamento(id, valor) -> int[32](id), string(valor).
8
9
10 mesmo_id(F1,F2,Nome) <-
11     funcionario(V1,F1),funcionario(V2,F2),
12     V1 != V2, F1 = F2,nome(F1,Nome) .
13
14 mesmo_salario(Nome) <-
15     mesmo_id(F1,F2,Nome), salario(F1,S),salario(F2,S),
16     nome(F1,Nome).
17
18 mesmo_cargo(Nome) <-
19     mesmo_id(F1,F2,Nome),
20     cargo(F1,C),cargo(F2,C), nome(F1,Nome).
21
22 mesmo_departamento(Nome) <-
23     mesmo_id(F1,F2,Nome),departamento(F1,D),
24     departamento(F2,D), nome(F1,Nome).
25
26 mesma_filial(Nome) <-
27     mesmo_id(F1,F2,Nome),filial(F1,F),
28     filial(F2,F), nome(F1,Nome).
29
30 //----------------------
```

```
31 recebeu_aumento(Nome) <-
32     mesmo_id(F1,F2,Nome),
33     salario(F1,S1),salario(F2,S2),
34     cargo(F1,C), cargo(F2,C),
35     S1<S2.
36
37 foi_promovido(Nome) <-
38     mesmo_id(F1,F2,Nome),
39     salario(F1,S1),salario(F2,S2),
40     cargo(F1,C1),cargo(F2,C2),
41     S1<S2,C1 != C2.
42
43 nova_funcao(Nome) <-
44     mesmo_id(F1,F2,Nome),
45     mesmo_salario(Nome),
46     cargo(F1,C1),cargo(F2,C2),
47     C1 != C2.
48
49 foi_transferido(Nome) <-
50     mesmo_id(F1,F2,Nome),
51     filial(F1,Fil1),filial(F2,Fil2),
52     Fil1 != Fil2.
53
54 foi_promovido_e_transferido(Nome) <-
55     foi_transferido(Nome),
56     foi_promovido(Nome).
57
58 novo_departamento(Nome) <-
59     mesmo_id(F1,F2,Nome),
60     departamento(F1,D1),departamento(F2,D2),
61     D1 != D2.
```

Arquivo que contém o resultado obtido após a aplicação das regras sobre os fatos

Listagem A.5: Resultado da Aplicação das Regras de Inferência

```
1  = = = = = = = = = = = = = = = = = = = =
2   Receberam aumento
3  = = = = = = = = = = = = = = = = = = = =
4     Joao
5     Maria
6     Pedro
7  = = = = = = = = = = = = = = = = = = = =
8  Nova Funcao
9  = = = = = = = = = = = = = = = = = = = =
10    Ana
11    Maria
12    Mateus
13    Pedro
14 = = = = = = = = = = = = = = = = = = = =
15 Novo Departamento
16 = = = = = = = = = = = = = = = = = = = =
17    Ana
18 = = = = = = = = = = = = = = = = = = = =
19 Foram Promovidos
20 = = = = = = = = = = = = = = = = = = = =
21    Maria
22    Pedro
23 = = = = = = = = = = = = = = = = = = = =
24 Foram transferidos
25 = = = = = = = = = = = = = = = = = = = =
26    Ana
27    Mateus
28    Pedro
29 = = = = = = = = = = = = = = = = = = = =
30 Foram promovidos e transferidos
```

```
= = = = = = = = = = = = = = = = = = = = =
   Pedro
```