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

Inferência em Documentos XML utilizando Prolog

Pedro Otávio Lima Gazzola

JUIZ DE FORA
DEZEMBRO, 2011

Inferência em Documentos XML utilizando Prolog

Pedro Otávio Lima Gazzola

Universidade Federal de Juiz de Fora

Instituto de Ciências Exatas

Departamento de Ciencia da Computação

Bacharelado em Ciência da Computação

Orientador: Alessandreia Marta de Oliveira

JUIZ DE FORA

DEZEMBRO, 2011

Inferência em Documentos XML utilizando Prolog

Pedro Otávio Lima Gazzola

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.

Jairo Francisco de Souza
M. Sc.

Luciana Conceição Dias Campos
D. Sc.

JUIZ DE FORA

07 DE DEZEMBRO, 2011

À minha família.

À Marina.

A todos os meus amigos.

Resumo

A linguagem de marcação XML permite a descrição e representação de dados semiestruturados e seu intercâmbio na Web. Um problema relacionado é que, assim como os dados armazenados em bancos de dados estruturados, os dados semiestruturados evoluem ao longo do tempo, em função, por exemplo, de modificações de cunho técnico. Estas modificações podem levar os dados semiestruturados a um estado inconsistente, pois as instâncias podem se tornar incompatíveis com as definições mais recentes dos esquemas. Uma alternativa para facilitar o gerenciamento destes dados é o uso da Gerência de Configuração neste contexto. Diante disso, a proposta é adaptar técnicas de Gerência de Configuração de Software no que diz respeito ao controle de modificações para esse cenário, fornecendo apoio para a evolução de dados semiestruturados. Para tanto, esse trabalho apresenta uma abordagem baseada em inferência, utilizando a linguagem Prolog, ilustrada como um módulo da ferramenta XPerseus.

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

Agradecimentos

Agradeço a Deus por todas as oportunidades, à minha família por me ensinar a escolher as corretas, e à minha namorada Marina por me apoiar em quaisquer que sejam.

Aos meus pais, Eduardo e Luzia, meus irmãos, Leonardo, Túlio e Carlos, pelo amor, confiança e apoio.

Agradeço aos meus colegas do Grupo de Educação Tutorial da Computação (GETComp UFJF), em especial ao Guilherme Martins, Plínio Garcia, Brian Siervi, Rafael Barros, Carolina Cunha, Lenita Ambrósio, Danúbia Dias e ao professor Rodrigo Luis.

À professora Alessandreia, pela confiança no meu trabalho, pela dedicação e pela orientação, sem as quais não conseguiria construir este trabalho.

A todos os meus amigos, por me darem forças e me ajudarem neste trabalho.

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 o nosso enriquecimento pessoal e profissional.

Sumário

Lista de Figuras

Lista de Abreviações

DCC	Departamento de Ciência da Computação
DTD	Document Type Definition
GCS	Gerência de Configuração de Software
IC	Item de Configuração
OWL	Web Ontology Language
POM	Project Object Model
RDF	Resource Description Framework
SCV	Sistemas de Controle de Versão
SCM	Sistemas de Controle de Mudança
SGC	Sistemas de Gerência de Construção
SWRL	Semantic Web Rule Language
UFJF	Universidade Federal de Juiz de Fora
XML	Extensible Markup Language
W3C	World Wide Web Consortium

1 Introdução

A área de desenvolvimento de software vem evoluindo constantemente nos últimos anos e, dentre as novas aplicações, é possível observar o aumento na utilização de dados semiestruturados, frequentemente representados por documentos XML (*Extensible Markup Language*) (Bray et al., 2008), que são arquivos de comportamento e características peculiares, possuindo uma estrutura hierárquica formada pela organização de marcadores, também chamados de *tags*, que são definidas arbitrariamente pelo desenvolvedor, possibilitando assim uma enorme flexibilidade de representação de informações.

Devido a esta estrutura específica, os documentos semiestruturados estão sujeitos a constantes modificações e alterações em sua forma, evoluindo de acordo com o desenvolvimento do projeto em que estão inseridos. Esta constante evolução dificulta o acompanhamento e gerenciamento da construção do software envolvido.

Para auxiliar a gerência dos sistemas que utilizam estes documentos, o interesse por técnicas e métodos com este propósito mais eficientes é intensificado, com o intuito de reduzir os custos de produção, amenizar os gastos com manutenção e retrabalho, entre outros fatores.

Uma das disciplinas interessadas em pesquisar soluções para estes problemas é a Gerência de Configuração de Software (GCS), que busca formalizar métodos e técnicas para auxiliar na gerência de projetos de software.

Uma das áreas da GCS é o Controle de Mudanças, que procura auxiliar na administração e auditoria de mudanças identificadas em um projeto, buscando e documentando os acontecimentos que culminaram nestas alterações, afim de fornecer uma estrutura para melhor compreensão da evolução de todos os elementos envolvidos em um desenvolvimento de uma aplicação.

Este trabalho procura aplicar esta pesquisa de auxílio à gerência de construção de software no contexto dos dados semiestruturados, ou seja, analisar a aplicação das técnicas e métodos existentes na disciplina GCS e no Controle de Mudanças nestas aplicações que utilizam documentos XML.

A abordagem proposta neste trabalho procura recuperar informações implícitas nestes arquivos, ou seja, realizar uma inferência de informações sobre a evolução de um documento XML, baseada nas mudanças no conteúdo do mesmo. Para tanto, a linguagem Prolog foi escolhida devido a adaptabilidade da mesma para este contexto e sua consolidação no ambiente científico.

1.1 Motivação

Observando o crescimento do número de aplicações provedoras de serviços para a Internet, é possível notar o aumento da utilização de documentos XML, uma forma de representação de dados semiestruturados, presentes no armazenamento e manipulação de diversos tipos de informações.

Consequentemente, cresce a necessidade de ferramentas para auxiliar o desenvolvimento e a manutenção destas aplicações que necessitam de uma abordagem especial, visto que os documentos XML possuem características diferentes dos arquivos estruturados.

1.2 Justificativa

A característica estrutural específica dos documentos XML inviabiliza a utilização de algumas ferramentas voltadas a arquivos comuns de texto, mas possibilita abordagens diferentes para a utilização da GCS nestes documentos semiestruturados.

A organização hierárquica dos dados nestes documentos pode ser aproveitada para recuperar informações implícitas, ou seja, inferir informações não apresentadas no conteúdo dos arquivos, mas alcançáveis a partir de uma análise mais complexa da estrutura destes.

1.3 Objetivos

A meta deste trabalho é auxiliar na gerência da evolução de documentos XML propondo uma abordagem para tanto, contribuindo assim para a pesquisa da disciplina GCS no

contexto de dados semiestruturados no que diz respeito ao Controle de Mudanças.

Esta proposta inclui uma aplicação capaz de ilustrar funcionalmente a abordagem escolhida para a inferência de informações e faz parte da abordagem da ferramenta XPerseus (Silva, 2011).

1.4 Organização do Trabalho

Este trabalho está dividido em 4 capítulos além desta introdução.

O Capítulo 2 se caracteriza pela definição da disciplina GCS e suas áreas de conhecimento. É realizada também uma análise deste mesmo tema no contexto dos dados semiestruturados, incluindo observações sobre ferramentas conhecidas desta área.

O Capítulo 3 apresenta os trabalhos relacionados e diferentes tipos de abordagens com o objetivo de recuperar informações implícitas em estruturas ou realizar inferência de conhecimento em documentos.

O Capítulo 4 aborda os detalhes da aplicação desenvolvida para ilustrar o funcionamento da proposta. Esta aplicação é na verdade um módulo da ferramenta XPerseus. Além disto, um estudo de caso é apresentado para melhor compreendimento do processo de inferência.

Por fim o Capítulo 5 contém as considerações finais sobre a pesquisa e o trabalho realizado, as propostas de trabalho futuro como melhorias à abordagem apresentada.

2 Gerência de Configuração

Devido ao rápido avanço da tecnologia da informação, o desenvolvimento de sistemas de software e a sua manutenção se tornaram processos difíceis e complexos. Estas inovações também geram grandes expectativas quanto ao projeto, muitas vezes representadas por maiores exigências, como prazos curtos e demandas maiores. Devido a estes fatores, é possível notar que um projeto de software está sujeito a possíveis modificações durante a sua elaboração e execução.

A causa destas alterações no contexto do desenvolvimento de um software pode ser de diversas naturezas, como a constante evolução de suas características, mudanças nas especificações e alterações no meio em que será utilizado, novos requisitos funcionais ou novas regras de negócio. Para acompanhar esta necessidade, muito se gasta atualmente com manutenção, correção de erros e outras atividades que levam a um alto índice de retrabalho.

Este capítulo apresenta um breve estudo sobre a GCS, iniciando o tema com histórico desta disciplina na Seção 2.1. A definição é apresentada na Seção 2.2 e suas subseções, exploram algumas das áreas que compõem este campo de conhecimento. A Seção 2.3 explora alguns conceitos da GCS aplicada a dados semiestruturados e por fim tem-se a conclusão na Seção 2.4.

2.1 Histórico

A GCS surgiu no fim dos anos 70, como uma tentativa de amenizar vários problemas e desafios encontrados pelo desenvolvimento de software ao longo dos anos. Na década de 80, o contexto era de grandes sistemas com problemas de composição, versionamento e reconstrução. As ferramentas da época eram construídas especificamente para o local de utilização, ou seja, de uso exclusivo daquele sistema ou empresa. Além disso, eram principalmente focadas no controle de versão dos documentos envolvidos nos projetos, sendo a maioria, conjuntos de *scripts* da plataforma Unix, utilizando ferramentas como

RCS (Tichy, 1982) e Make (Feldman, 1979).

No início da década de 90 os problemas mudaram, tornaram-se obstáculos o suporte a processos, a engenharia concorrente e outros. Neste momento surgiram as primeiras ferramentas de GCS de fato, utilizando bases de dados relacionais, controle de arquivos, suporte a área de trabalho e outras funcionalidades interessantes para o momento (Estublier, 2000).

Atualmente existem muitas dificuldades diferentes destas, como o desenvolvimento remoto, equipes distantes geograficamente e a engenharia Web distribuída. Foi no fim da década de 90 que os sistemas de GCS realmente amadureceram, tornando-se uma parte essencial no desenvolvimento de software, como ferramentas confiáveis e estáveis (Estublier, 2000).

2.2 Gerência de Configuração de Software

A GCS é uma disciplina que utiliza procedimentos técnicos e administrativos para detectar e documentar as características dos elementos de um projeto, ou itens de configuração (IC[1]), além de controlar as alterações nestes componentes, armazenar e relatar o processamento das modificações de cada estágio da implementação e verificar a compatibilidade com os requisitos especificados (IEEE, 2005). Segundo Estublier (2000), a GCS é uma disciplina que permite a evolução dos produtos de software de forma controlada, contribuindo assim para a satisfação de restrições de qualidade e de tempo definidas no planejamento do projeto.

A GCS também pode ser definida como uma forma de gerenciar a construção de um software, identificando, organizando e controlando a evolução do mesmo, ou ainda como o desenvolvimento e a aplicação de procedimentos para gerenciar um sistema em construção (Sommerville, 2011).

O objetivo da GCS é amenizar os custos relacionados à manutenção de software, maximizando a produtividade e minimizando os erros cometidos durante a evolução do mesmo. Para tanto é necessário um processo de desenvolvimento sistemático e discipli-

[1]O termo item de configuração (*configuration item*) representa a agregação de hardware, software ou ambos, tratada pela GCS como um elemento único (IEEE, 1990).

nado, baseado em métodos e ferramentas que possibilitam uma constante auditoria de todos os elementos envolvidos na construção do software (Dantas, 2010).

Dentre os conceitos de GCS, é possível observar quatro fundamentos que dão suporte à meta de auxiliar o processo de construção de software. O primeiro é a identificação, cujo modelo representa os componentes do projeto e o seu tipo, de forma única e de alguma maneira acessível. O aspecto seguinte é o controle. É preciso gerenciar o projeto e suas mudanças do início ao fim, garantindo a entrega de um produto consistente. O próximo fundamento é a documentação, ou seja, armazenar e possibilitar a recuperação de informações vitais e relatórios sobre o estado de todos os elementos e artefatos envolvidos ao longo do projeto. Por último estão presentes a auditoria e a revisão, para validar a completude do produto e manter a consistência dos muitos componentes que são parte do mesmo (Dart, 1991).

Sob a perspectiva de desenvolvimento, a GCS é composta de três subsistemas: Controle de Versões, Controle de Mudanças e Gerenciamento de Construção. O Controle de Versões combina procedimentos e ferramentas para identificar, armazenar e controlar os artefatos de trabalho (itens de configuração que são criados durante o desenvolvimento) e sua evolução ao longo do ciclo de vida do projeto para que ela ocorra de forma organizada. O Controle de Mudanças 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. A Gerência de Configuração não propõe a forma como serão feitas as alterações, apenas controla a maneira como as modificações são realizadas e disponibilizadas. O Gerenciamento de Construção, por sua vez, é responsável pela disponibilização das versões finais para os usuários, automatizando a transformação dos artefatos do projeto em versões executáveis (Murta, 2006).

2.2.1 Controle de Versão

Para garantir um ambiente de desenvolvimento menos propenso a falhas e erros é preciso que a evolução dos artefatos possa acontecer de forma distribuída, concorrente e ordenada, ou seja, deve ser possível que os envolvidos possam trabalhar no mesmo projeto, ou até nos mesmos itens ao mesmo tempo, minimizando a tarefa de combinar resultados.

Construir este ambiente de forma manual não é aconselhável, devido ao alto custo e a pouca confiabilidade resultante deste método. Assim, para garantir um cenário como este, os sistemas de controle de versão (SCV) concentram várias funcionalidades, entre elas a automatização do armazenamento e recuperação de itens, identificação e mesclagem de revisões (Oliveira, 2007).

Atualmente existem várias ferramentas já consagradas deste tipo como o Subversion (Collins-Sussman, 2002), o Git (Chacon, 2010) e o Mercurial (Selenic, 2011).

O Subversion é um projeto de código aberto iniciado no ano 2000, com grande aceitação no mercado. Seu funcionamento se baseia na utilização de um repositório central para armazenar todos os arquivos do projeto, além do histórico das modificações feitas nestes, incluindo informações como o autor da alteração, quando esta foi feita e o que foi alterado. É um SCV centralizado, assim todas as trocas de mensagens são realizadas entre o servidor central e os clientes ou estações de trabalho (Collins-Sussman, 2002).

O Git é um software livre originalmente desenvolvido por Linus Torvalds para gerenciar a evolução do núcleo do sistema operacional Linux, tendo a sua primeira versão lançada em 2005. A ferramenta é especialmente voltada para uso na plataforma citada, mas atualmente pode ser utilizada em outros sistemas operacionais. O funcionamento do Git caracteriza-se por ser um sistema de controle de versões distribuído, ou seja, cada estação de trabalho é um repositório completo, com todos os dados e históricos armazenados, sendo possível realizar operações não somente entre um servidor central e as estações, mas também entre elas (Chacon, 2010).

O Mercurial também é um SCV distribuído e de livre utilização, desenvolvido utilizando a linguagem Python com o intuito de atingir um maior número de plataformas. Principalmente indicada para grandes projetos, devido a eficiência e rapidez que suas operações são realizadas, esta ferramenta é facilmente extensível, através da estrutura modular adotada (Selenic, 2011).

2.2.2 Controle de Mudança

Para um grande projeto de engenharia de software, modificações sem controle levam rapidamente a um ambiente confuso e propenso a erros. Para tais projetos, os sistemas

de controle de mudanças (SCM) combinam procedimentos humanos e funcionalidades automatizadas (Pressman, 2011).

Estes sistemas trabalham de maneira sistemática, documentando todos os dados gerados em qualquer tipo de modificação no projeto durante a evolução do mesmo, e por fim, disponibilizando esta informação para os interessados e autorizados.

Ao envolver dados referentes às solicitações de mudanças, este tipo de serviço é caracterizado por possibilitar o conhecimento do motivo ou razão das alterações e por registrar o que foi alterado, quando e por quem.

Softwares como o Trac (Trac, 2011), Bugzilla (Mozilla, 2011) e Mantis (Mantis, 2006) são exemplos de SCM.

O Trac é um software de código aberto, desenvolvido na linguagem Python, que possui uma interface Web para rastrear as mudanças ocorridas em um projeto, entender o porquê de cada uma e o impacto na aplicação. Fornece algumas funcionalidades interessantes como uma documentação colaborativa, ou seja, que todos envolvidos podem editar e utilizar, fácil integração com ferramentas de controle de versão como o Subversion (Trac, 2011).

O Bugzilla é um software livre, baseado em uma interface Web com o intuito de rastrear defeitos e também possibilitar uma plataforma de requisições. A primeira versão surgiu em 1998, e desde então já foi adotado por vários projetos de software livres ou proprietários. O maior atrativo desta ferramenta é sua velocidade e sua implementação de baixo custo. Este é o resultado de algumas diretivas do software, como evitar ao máximo a geração de grandes arquivos HTML, manter a frequência e a quantidade de dados recuperados das bases ou das aplicações envolvidas a mais baixa possível (Mozilla, 2011).

O Mantis é desenvolvido utilizando a linguagem PHP. Assim seu funcionamento é marjoritariamente baseado em uma interface visualizada através de navegadores, funcionando como clientes. A grande preocupação desta ferramenta é manter a usabilidade do sistema, prezando por simplicidade e clareza das muitas funcionalidades. A primeira versão surgiu em 2006 e após este lançamento, muito se investiu para que fosse possível integrar este SCM com os SCV mais populares como o Git e o Subversion (Mantis, 2006).

2.2.3 Gerência de Construção

O controle de versões e o de modificações ajudam o desenvolvedor de software a manter a ordem na construção do projeto, o que de outro modo seria uma situação confusa. No entanto, mesmo os mecanismos de controle mais bem-sucedidos acompanham a modificação apenas até certa etapa, não há como garantir que a modificação foi adequadamente implementada (Pressman, 2011).

Os sistemas de gerência de construção (SGC) são feitos para automatizar o processo de transformação dos itens do projeto em um produto final, obedecendo às normas, padrões, procedimentos e políticas definidas pela especificação do sistema. Com isso, é possível atingir uma maior produtividade na etapa final de desenvolvimento, além de reduzir o risco de erros e operações equivocadas.

O objetivo dos SGC é automatizar o processo de transformação dos diversos artefatos do software que compõem um projeto em um sistema consumível. Por exemplo, o processo de teste e empacotamento de uma aplicação Java como um arquivo ".jar" (Silva, 2011). São ferramentas comuns de gerenciamento de construção o Ant (Apache, 2000), o Make e o Maven (Apache, 2011).

O Ant é utilizado em projetos de aplicações Java, porque além de ser uma ferramenta sem interface gráfica que tem como objetivo auxiliar no processo de construção, ou *build*, de arquivos, é uma biblioteca para a linguagem Java. O Ant fornece inúmeras funcionalidades nativas para compilar, unir, testar e executar aplicações Java, mas além disso, ele também pode ser utilizado para outros tipos de softwares, por exemplo projetos que utilizem a linguagem C ou C++. De forma geral pode ser usado em qualquer tipo de projeto, devido a sua flexibilidade e a não imposição de nenhum tipo de convenção de código ou organização de diretórios (Apache, 2000).

O Make é um dos softwares pioneiros a implementar um processo de automação de construção, sendo o foco inicial a automação das chamadas ao compilador. Todo o funcionamento desta ferramenta, ainda muito utilizada em sistemas UNIX, gira em torno de um arquivo chamado de *makefile*, criado especialmente para cada projeto, faz o papel de guia para a ferramenta agir de acordo com as especificações do mesmo. Esta ferramenta não é portável devido ao funcionamento de suas funcionalidades, baseadas principalmente

em chamadas ao sistema operacional (Feldman, 1979).

O Maven é uma ferramenta para gerenciamento e automatização de projetos Java, é baseada no Ant mas usa um arquivo como modelo de projeto chamado de *Project Object Model* (POM), que contém as informações básicas da aplicação além da maneira que o artefato final do projeto deve ser construído. Por padrão, possui convenções que buscam as melhores práticas de desenvolvimento, como uma organização de diretórios pré-moldada, suporte a testes unitários e outras. A principal característica é a fácil inserção de módulos, com o intuito de proporcionar um maior grau de caracterização para o desenvolvimento específico em questão (Apache, 2011).

2.3 Gerência de Configuração de Dados Semiestruturados

Junto ao crescimento do mercado de software, é possível notar o aumento da presença de dados semiestruturados em várias aplicações diferentes. Comumente representados por documentos seguindo o formato XML (Bray et al., 2008), que é recomendado para publicação, combinação e intercâmbio de documentos multimídia, desenvolvido pelo consórcio W3C (*World Wide Web Consortium*).

Este formato é uma linguagem de marcação, ou seja, todo o conteúdo do documento é definido por uma ou várias marcas, também conhecidas como *tags*. Estas marcas são definidas pelo desenvolvedor, possibilitando a representação de forma e conteúdo em uma mesma estrutura, fornecendo assim uma enorme flexibilidade de representação, o que contribuiu para o sucesso deste tipo de representação.

Um documento XML é normalmente definido com o auxílio de uma estrutura em árvore, onde os nodos correspondem a elementos, atributos ou valores, e as arestas representam relações entre elementos e subelementos ou entre elementos e valores. Este tipo de estrutura caracteriza o aspecto hierárquico deste formato, fator muito explorado por aplicações que manipulam estes arquivos.

Estas características permitem que documentos XML sejam utilizados em diversos contextos, como por exemplo em aplicações de serviços Web, ou *Web Services*. A utilização

deste formato também pode auxiliar na integração entre clientes e provedores de serviço, como por exemplo em comércio eletrônico, permitindo a troca de mensagens sobre a transação de forma eficiente. Além disto, o encapsulamento de conteúdo XML é adequado para implementação de características de maior complexidade em trocas de dados entre diferentes aplicações, como privacidade e segurança de dados (Moro e Braganholo, 2009).

Com tantas utilizações, esse tipo de arquivo se tornou uma abordagem muito utilizada para representar dados da Internet (Laender et al., 2009). Entretanto, esta abordagem também possui problemas, dependendo do contexto da aplicação por exemplo, documentos XML podem ser muito grandes, tornando a manipulação e as consultas das informações do sistema complexas e custosas.

Para amenizar ou resolver este e outros problemas encontrados nas aplicações que utilizam estes documentos, deve-se construir um processo seguro de desenvolvimento e manutenção destes sistemas. Para tanto, deve-se pesquisar técnicas e métodos que auxiliem na formulação deste processo.

Devido as particularidades dos documentos semiestruturados, como os escritos utilizando a linguagem XML, não é possível aplicar com sucesso todos os fundamentos da GCS. Assim, é preciso realizar a pesquisa e desenvolvimento de uma área de conhecimento que contemple todo este contexto. Esta nova disciplina é chamada de Gerência de Configuração de Dados Semiestruturados, porque herda vários conceitos da GCS definida anteriormente, mas precisa solucionar novos problemas relacionados aos dados citados.

2.3.1 Controle de Versão de Dados Semiestruturados

Algumas ferramentas conhecidas, como Subversion, Mercurial e Git, não possuem um suporte adequado aos documentos XML, que possui uma estrutura específica e diferente da forma dos documentos usuais, mas mesmo assim, são utilizadas para trabalhar com este tipo de arquivo. Por isso, é comum encontrar erros nas operações feitas nestes artefatos, aumentando o número de horas gastas para resolver estes problemas, reduzindo a produtividade da equipe.

Por outro lado, é possível observar que já foram construídos protótipos e ferramentas de controle de versão, criadas especificamente para tratar este problema. Alguns

destes sistemas são XSDelta (Perini, 2006), o XVersion (Giacomel, 2006), o VisualX (Cecchin e Hara, 2009), o XMLTreeMerge (Oliveira et al., 2011) e o XPerseus (Silva, 2011).

Estas ferramentas são capazes de trabalhar seguindo a estrutura hierárquica dos documentos, atenuando a ocorrência de erros e a intensidade destes.

O XSDelta (Perini, 2006) utiliza o algoritmo de detecção XyDiff (Marian et al., 2001). Possui uma interface gráfica que possibilita visualizar textualmente, e com variadas cores, as mudanças entre arquivos XML, que podem ser definidos em DTD ou XML Schema. A ferramenta é capaz de emitir como saída um arquivo XML contendo apenas as diferenças detectadas, de forma a auxiliar a gerência da evolução dos documentos analisados.

A ferramenta XVersion (Giacomel, 2006), ou XML Versioner, utiliza o algoritmo JXyDiff, e além de verificar alterações entre arquivos XML, também possibilita agrupar estes documentos em um novo arquivo. Apesar de possuir uma interface gráfica, não possibilita a navegação entre as diferenças encontradas e não há como mesclar arquivos. Além destas características, é possível observar a possibilidade de se construir um histórico referente aos documentos analisados e a funcionalidade de realizar consultas, como são feitas em um banco de dados.

O VisualX (Cecchin e Hara, 2009) utiliza os algoritmos XyDiff, JXyDiff (Tani et al., 2011) e XKeyMatch (Santos, 2006). Com isso, cria-se a possibilidade da escolha de qual será utilizado. Também exibe através da sua interface gráfica, as diferenças encontradas entre os documentos XML em cores específicas e permite edição dos documentos, mas não disponibiliza a exibição dos casamentos realizados ou a mesclagem dos arquivos.

A ferramenta XMLTreeMerge (Oliveira et al., 2011) utiliza o algoritmo de diferenciação 3dm (Lindholm, 2004), exibindo em sua interface gráfica as diferenças entre os arquivos. Esta visualização é feita representando os arquivos semiestruturados em forma de árvore. Ao contrário da abordagem textual encontrada mais comumente nas outras ferramentas, nesta forma de visualização, os nós comuns entre os dois documentos são expostos de uma maneira, já aqueles que apresentam diferenças são representados de forma destacada e especial, assim, a ferramenta exibe com maior clareza as modificações dentre documentos XML. Apesar de permitir fazer a mesclagem dos mesmos, XMLTreeMerge

não permite a edição dos documentos envolvidos.

Por último, o sistema XPerseus (Silva, 2011), possui uma interface gráfica para a detecção e visualização de diferenças entre duas versões de um documento XML. Este software utiliza o algoritmo de diferenciação JXyDiff para gerar um arquivo que contém de forma textual todas as alterações identificadas, chamado de *delta*, que pode ser utilizado para a auditoria ou mesclagem destes. A ferramenta foi construída com a linguagem de programação Java, devido a diversos fatores como a possibilidade de utilização em diferentes sistemas operacionais, extensa documentação e fácil suporte a manipulação de documentos XML.

Além destas características, planeja-se um novo módulo para este sistema, possibilitando a combinação das diferenças representadas no arquivo *delta* também através da interface gráfica (Siervi, 2011), e também uma reestruturação no código do programa com melhorias utilizando técnicas de padrões de projeto visando uma maior facilidade de manutenção da ferramenta.

2.3.2 Controle de Mudança de Dados Semiestruturados

A análise da evolução de artefatos utilizada pelos SCM não diferencia documentos semiestruturados de documentos comuns, porque para atingir os objetivos deste tipo de ferramenta, como registro de histórico e auditoria de mudanças, não é preciso uma abordagem específica neste contexto.

As ferramentas mais conhecidas de SCM seguem este raciocínio e não apresentam uma análise diferente para documentos XML, mas é possível aproveitar a estrutura hierárquica destes arquivos para realizar pesquisas relativas à obtenção de resultados mais expressivos para o controle de mudanças, como a recuperação de informações com maior nível de expressividade ou valor semântico.

Outro aspecto interessante a ser analisado é que os próprios SCM geram e contêm dados desta natureza. São usados campos pré-definidos e descrições textuais para estabelecer metadados referentes às solicitações de mudanças e relatórios de erros.

Estas informações são uma valiosa fonte de conteúdo para entender as modificações ocorridas no projeto. Para atingir esta fonte, é possível construir métodos de

pesquisa especializados em acessar conhecimento através dos metadados, algo que não seria possível realizar utilizando os sistemas comuns de busca baseados em texto. Além da busca, com este tipo de informação pode-se automatizar sistemas inteligentes para diagnosticar ou resolver problemas através da análise de experiências registradas no projeto (Tran et al., 2009).

Entretanto, explorar este tipo de informação implícita tem se provado um desafio. Embora seja da preferência dos usuários a possibilidade de descrever os acontecimento de forma livre em um texto, a análise automatizada destas descrições é de baixa eficiência. A alternativa para este tipo de armazenamento são campos pré-definidos para definir os metadados, mas em contrapartida esta abordagem apresenta muitas limitações.

2.3.3 Gerência de Construção de Dados Semiestruturados

O aumento da utilização dos dados semiestruturados também afetou a gerência de construção de várias formas. Este tipo de documento é muito utilizado por SGC para auxiliar a representação da estrutura de diretórios e arquivos de um projeto, para orientar a sequência de operações que a ferramenta deve seguir ou organizar todas as dependências do projeto.

Um exemplo desta importância é a ferramenta citada Maven, que tem todo o seu funcionamento baseado nos arquivos chamados POM, um documento XML que concentra várias informações diferentes do projeto. Assim é possível acompanhar de forma eficiente a evolução de um projeto, que pode ser muito flexível e dinâmica.

A maior parte das funções da gerência de construção dependem desta eficiência, como a análise de riscos, o controle de qualidade, a disponibilização do sistema e da documentação.

2.4 Conclusão

Atualmente o desenvolvimento de software cresce de maneira surpreendente, e para se obter resultados de maneira satisfatória, com uma alta produtividade, menores custos de produção e um produto final satisfatório, é preciso observar os aspectos da disciplina de

GCS.

Tendo em vista este crescimento, e observando a crescente taxa de utilização de dados semiestruturados em diversas aplicações, é possível notar a necessidade por diferentes técnicas que auxiliem as diversas áreas da gerência de projetos para este contexto.

Devido a natureza específica destes dados, estas técnicas devem ser elaboradas de forma que contemplem estas características e novas carências, fatores não levados em consideração nos sistemas de GCS mais antigos.

No próximo capítulo é apresentada uma forma de auxiliar os SCM utilizando métodos de inferência de informações, ou seja, técnicas para recuperar dados implícitos de maneira eficiente.

3 Uso de Inferência na Evolução de Documentos Semiestruturados

Atualmente, é possível observar que devido a complexidade de se gerenciar um grande volume de dados de uma organização, faz-se necessário um sistema que automatize as buscas e o processo de recuperação de informações com maior valor semântico agregado (Fontes, 2011).

Esta necessidade é intensificada devido as características dos sistemas tradicionais de consulta usados na gerência de bases de dados, que utilizam mecanismos de busca baseados em comparação de caracteres, sendo capazes de obter somente resultados superficiais se comparados a profundidade do conhecimento que pode ser armazenado. A abordagem apresentada neste trabalho almeja obter resultados através da análise dos relacionamentos entre os dados, procurando pelo conhecimento presente mas não representado textualmente.

Observando especificamente os sistemas que trabalham sobre dados semiestruturados como documentos XML, é possível observar que a maior parte deles analisam e obtém suas respostas a partir do caminho percorrido na árvore definida pelos componentes XML. As técnicas usadas por estes sistemas observam apenas dados explícitos na árvore, não levando em consideração qualquer tipo de informação implícita que poderia ser recuperado através de uma análise mais complexa (Mury et al., 2011).

Diante disso, este capítulo aborda algumas técnicas de inferência de dados aplicadas à gerência de mudanças, começando com uma introdução ao tema. Em seguida, a Seção 3.1 apresenta os possíveis resultados da aplicação de uma técnica desta natureza, além de alguns aspectos da utilização da mesma. Na Seção 3.2 são apresentadas diferentes propostas para aplicação destes conceitos, com foco sobre o tema deste trabalho, a utilização do Prolog neste contexto. Por fim a Seção 3.3 apresenta alguns trabalhos relacionados presentes na literatura e a Seção 3.4 apresenta a conclusão.

3.1 Uso de Inferência

O conhecimento implícito nos documentos XML pode ser identificado por um conjunto de regras, criadas para fornecer aos sistemas automatizados de inferência o poder de raciocinar sobre estruturas de dados (Fontes, 2011).

Estas regras não podem analisar qualquer conjunto de informações. A base de conhecimento deve ser traduzida para uma forma que os sistemas possam trabalhar, ou seja, um conjunto de declarações e fatos compatíveis com a maneira de funcionamento do sistema.

Por exemplo, para identificar se duas pessoas são irmãs, uma regra é criada definindo que se estas possuem os mesmos pais, elas são irmãos. Então, após análise de uma base fictícia de informações, três declarações são obtidas, a primeira representa os pais de Luiza, que são Maria e Tiago, a segunda representa os pais de Pedro, são eles Antônio e Paula, a última identifica os pais de José, a Maria e o Tiago novamente.

Assim, observando a regra e aplicando-a às declarações, a máquina de inferência pode induzir ou raciocinar que José e Luiza são irmãos.

É possível definir a inferência como a possibilidade de criação de adições lógicas a uma representação de conhecimento, formada através da classificação de uma fonte de informações (Fontes, 2011).

3.2 Abordagens para Realizar a Inferência

Com o objetivo de aplicar estes conceitos de inferência, algumas abordagens foram propostas para trabalhar sobre diferentes tipos de documentos, entre elas se destacam as linguagens Prolog e Datalog e o uso de ontologias e seus raciocinadores.

3.2.1 Ontologias

O termo ontologia é utilizado em muitas áreas diferentes, mas para a organização da informação pode-se conceituá-lo de forma simples como um catálogo de tipos de objetos pertencentes a um mesmo domínio, onde estes podem ser de qualquer natureza, como bens materiais ou abstrações, e o domínio define o campo semântico ao qual se referem

(Almeida e Bax, 2003).

Uma formalização para ontologias as define como descrições de determinados conceitos em um domínio de conhecimento, formadas através de relações, restrições e axiomas, definidos de forma declarativa em uma determinada linguagem (Freitas, 2003).

Um exemplo para domínio seria futebol cujos conceitos ou objetos seriam clube, jogador, Pelé, podendo ser divididos entre gerais como jogador, ou individuais, como Pelé, formando a categorização de conceitos, auxiliando na formulação de ontologias.

Além da categorização, uma ontologia é formada por relações, como Pelé é jogador, ou seja, Pelé é um jogador de futebol. O último componente desta forma de representação é o vocabulário, que forma o grupo de termos utilizados para cada domínio (Fontes, 2011).

Dentre as abordagens que trabalham com ontologias, a maior parte delas utilizam mecanismos chamados de raciocinadores para realizar a inferência a partir da análise de dados representados desta maneira.

Os raciocinadores, ou *semantic reasoners*, são ferramentas computacionais que podem indicar inconsistências, dependências ocultas ou redundâncias em ontologias através da análise de um conjunto de fatos ou axiomas (Santos et al., 2008).

Um dos sistemas bem conhecidos chama-se Pellet (Parsia e Sirin, 2004), que analisa ontologias representadas pela linguagem Web Ontology Language (OWL) (McGuinness e van Harmelen, 2004).

Além da linguagem OWL, existe a Semantic Web Rule Language (SWRL) (Dean et al., 2004), baseada em uma combinação de características de OWL e RuleML. 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.

Embora sejam propostas mais recentes, a OWL e a SWRL são linguagens ainda pouco desenvolvidas em comparação ao Prolog, discutido na Seção 3.2.3.

3.2.2 Datalog

O Datalog é uma linguagem de consulta em bancos de dados baseada em Prolog, existem muitas características em comum entre as duas linguagens, mas as diferenças devem ser comentadas. A forma mais comum de utilização desta linguagem é para auxiliar a pesquisa em grandes bancos de dados relacionais.

Nos últimos anos, o Datalog ressurgiu e vem sendo utilizado para sistemas onde é necessário um nível de abstração mais alto para consultas de estruturas relacionais e gráficas, execução eficiente de consultas recursivas e técnicas de manutenção incremental baseadas em modelos relacionais, raciocínio formal e análise, tais como: integração de dados, redes declarativas, análise de programas, extração de informações, monitoramento de rede, segurança e *cloud computing*. Além disso, a recursividade nas consultas oferecidas pelo Datalog se faz útil dada a complexidade das aplicações existentes hoje em dia, onde é cada vez mais necessária a interligação de dados buscados em diversas fontes e sua análise (Huang et al., 2011).

Datalog é portanto uma linguagem de consulta não procedural baseada na linguagem de programação em lógica Prolog. Como no cálculo relacional, em Datalog, um usuário descreve as informações desejadas sem fornecer um procedimento específico para obter estas informações. A sintaxe de Datalog se assemelha à do Prolog. Entretanto, como o significado dos programas Datalog é definido de uma maneira puramente declarativa, diferente das semânticas mais procedurais do Prolog, o Datalog simplifica a escrita de consultas simples e facilita a otimização de consultas (Korth et al., 2006).

Um programa Datalog consiste em um conjunto de fatos e regras. Os fatos são especificados de modo semelhante à especificação das relações, exceto pelo fato de a inclusão dos nomes dos atributos não ser necessária. As regras se parecem com as visões relacionais (*views*) e especificam relações virtuais que não são armazenadas de fato, mas que podem ser formadas de fatos, aplicando mecanismos de inferência baseados nas especificações das regras. A principal diferença entre as regras e as visões é que as regras podem envolver recursão e assim render relações virtuais, que não podem ser definidas em termos de visões relacionais básicas. Sobre a máquina de inferência, a ideia é deduzir fatos novos do banco de dados interpretando as regras (Elasmri e Navathe, 2005).

Existem algumas desvantagens quanto a utilização do Datalog para a inferência de dados, como a não permissão para utilização de termos complexos como argumentos do predicado e restrições no uso de negação e recursividade (Mury et al., 2011).

Cunha (2011) apresenta uma análise sobre a utilização de Datalog para a inferência de dados, utilizando um estudo de caso com o motor *Logic Blox* (LogicBlox, 2011) para ilustrar o uso desta abordagem no tema citado.

3.2.3 Prolog

O Prolog é uma linguagem em que os programas são escritos como regras para provar relacionamentos entre objetos, se identificando como uma linguagem de programação baseada em lógica matemática. O interesse em utilizar lógica como forma de representação do conhecimento foi uma das motivações para o desenvolvimento desta linguagem, inicialmente utilizada para interpretar linguagem natural (Cunha et al., 2007). O motor desta linguagem é definido sobre um subconjunto da lógica de primeira ordem e as Cláusulas de Horn são utilizadas para representar as informações (Coelho, 2002).

A programação em lógica pode ser definida como a representação de um determinado problema a partir de um conjunto finito de sentenças lógicas, chamadas de cláusulas (Palazzo, 1997).

Nesta linguagem, que pode ser brevemente definida pelas características descritas a seguir, não há distinção entre programa e dados.

O sistema de interpretação desta linguagem realiza uma série de soluções baseadas em operações lógicas, analisando a base de dados e o conjunto de regras disponíveis.

Esta base de dados é formada por fatos, que expressam propriedades de objetos ou ligações entre dois ou mais objetos. Os fatos são representados seguindo a notação a seguir, onde está representada a informação que *Boeing* é a fabricante do avião *747*.

```
fabricante(boeing, 747).
```

As regras de um programa Prolog, também conhecidas como cláusulas, são formadas por uma cabeça (ou a conclusão da regra), e um corpo (a condição), que juntos especificam algo que pode ser verdadeiro se as condições apresentadas forem satisfeitas. A cláusula a seguir exemplifica a representação de uma regra. Neste caso, é possível in-

terpretar esse trecho como: “Para todo X e Y, se X é fabricante de Y, então Y é produto de X” (Palazzo, 1997).

```
produto(Y, X) :- fabricante(X, Y).
```

Para se obter respostas sobre um conjunto de fatos utilizando o Prolog, é necessário formular as requisições em consultas, seguindo uma representação como a seguinte, que também pode ser lida como “757 é um dos produtos da empresa Boeing?”. Uma consulta é composta por uma sequência de objetivos e para obter a resposta, o sistema tenta confirmar todos eles, analisando-os como uma conjunção.

```
?-produto(757,boeing).
```

Após análise das tecnologias disponíveis para realizar a inferências de dados, a linguagem Prolog foi escolhida para auxiliar na proposta deste trabalho, devido a maturidade do projeto, que leva a um maior número de informações e orientações disponíveis sobre este sistema. Além disso, a estrutura dos documentos semiestruturados e os recentes trabalhos na tradução dos mesmos para fatos Prolog (Mury et al., 2011), descrito na seção 4.2, e (Coelho, 2002), descrito na seção a seguir, se apresentaram como fatores contribuintes a esta decisão.

3.3 Trabalhos Relacionados

Dentre as diferentes propostas presentes na literatura para realizar um tipo de inferência e recuperação de informações implícitas, destacaram-se os sistemas a seguir, que utilizam mecanismo e técnicas diversas em variados contextos.

Coelho (2002) descreve a implementação de um sistema de processamento de documentos XML em programação em lógica, que permite a validação estática de programas, recorrendo a um sistema de inferência de tipos e ao estabelecimento de uma relação entre os tipos inferidos dos programas e as regras de construção dos documentos XML (DTDs - *Document Type Definition*). A proposta realiza inferência de tipos em um documento XML com a utilização de programação em lógica e visa deduzir o tipo de dado dos elementos contidos na estrutura e utiliza uma DTD para auxiliar o processo de tradução da linguagem XML para a linguagem Prolog. A abordagem proposta traduz todo o do-

cumento XML em um único predicado Prolog. A partir deste predicado, infere-se o tipo de dados dos elementos do documento XML inicial, aumentando o conhecimento sobre tal documento.

O trabalho de Coelho (2002) é uma importante contribuição para a pesquisa deste trabalho, entretanto, a abordagem proposta por ele não cobre a inferência de informações sobre o conteúdo dos elementos contidos no documento XML, principal meta desta proposta.

O ROSA (Porto e Moura, 2006) é um sistema para recuperação de objetos de aprendizado baseado na descrição semântica de seus conteúdos. O sistema tem por objetivo estender o modelo de dados ROSA, a partir de sua representação como uma base de conhecimento, nas quais assertivas são definidas como fatos e restrições e regras são expressas através de fórmulas lógicas. As consultas presentes na extensão do modelo são expressões lógicas com variáveis livres. Sua avaliação sobre a base de conhecimento permite a inferência de novos fatos, segundo a premissa do mundo aberto. A base de conhecimento é implementada em uma máquina de inferência Prolog que trata fatos e regras, correspondendo ao domínio de aplicações suportadas pelo ROSA.

O ROSAI (Costa, 2005) é uma extensão do ROSA que utiliza a programação lógica para expressar mapas conceituais e regras, implementando herança e propriedades de relacionamentos, permitindo assim a inferência de conhecimento não-explícito sobre sua base de dados. Entretanto, esta extensão não foi incorporada formalmente ao modelo de dados ROSA.

A utilização da linguagem Prolog nos sistemas ROSA e ROSAI para realizar a recuperação de informações é muito próxima da proposta neste trabalho, mas a base de conhecimento e os objetos de aprendizado se situam em um contexto diferente dos dados semiestruturados e da GCS.

O ROSA+ (Mattos et al., 2006) corresponde a outra extensão do sistema ROSA que visa deduzir conhecimento semântico através de propriedades de relacionamentos e de regras. Baseado na linguagem de ontologia OWL e na linguagem de regras SWRL, o ROSA+ realiza inferências sobre uma base de dados OWL, recuperando conhecimento não explicitado em sua representação ontológica.

Embora esta proposta ilustre uma utilização contundente da inferência de informações implícitas e possibilite trabalhar com objetos em diferentes níveis de abstração através da arquitetura em camadas, ela se baseia na análise de bases construídas através de ontologias e suas representações, como OWL e nas ferramentas deste contexto, como a citada SWRL. Estas características distanciam o sistema proposto deste trabalho.

O Autômeta (Fontes, 2011) é uma proposta de uma arquitetura para anotação semântica que utiliza os recursos de uma ontologia e um mecanismo de inferência. A proposta é obter relações implícitas sobre os conceitos presentes no documento com o auxílio de um raciocinador, além de prover mecanismos que auxiliam na anotação manual e de um recurso de visualização das triplas anotadas. A ideia é que os raciocinadores, a exemplo do Pellet, sejam capazes de interoperar entre SWRL e OWL, de modo que, após sua execução, as deduções obtidas sejam convertidas em fatos, e uma nova execução seja realizada. Esta iteração deve ocorrer até que uma inconsistência seja detectada ou novos fatos não sejam mais inferidos.

Assim como o sistema ROSA+, esta proposta se diferencia da abordagem deste trabalho nos mecanismos utilizados para realizar a inferência, ao utilizar um raciocinador como o Pellet, baseado em representações OWL e SWRL. Além disto, a proposta não apresenta funcionalidades para o auxílio ao controle de evolução de um documento XML.

Zhao et al. (2005) aborda a mineração de deltas estruturais. O objetivo é extrair conhecimento a partir de sequências de mudanças estruturais em documentos XML, tendo como base as características obtidas a partir das versões de documentos XML. Ao contrário de algumas propostas, a abordagem apresentada leva em consideração o aspecto dinâmico e temporal dos documentos XML e utiliza uma sequência de versões históricas de um documento XML, ao invés de utilizar um conjunto de documentos coletados em um determinado momento. Tal conhecimento pode ser útil em detecção de alterações em documentos XML muito grandes, indexação e máquinas de busca XML.

A principal diferença para este trabalho é que Zhao et al. (2005) leva em consideração apenas as mudanças estruturais, portanto a operação de atualização, que resulta em mudanças de conteúdo não é considerada. Para realizar inferência de informações para auxiliar a gerência da evolução dos documentos XML, é preciso observar as alterações do

conteúdo dos arquivos, por isso esta proposta de mineração não é adequada a pesquisa deste trabalho.

3.4 Conclusão

Dada a definição de inferência de dados é possível observar a variedade de utilizações para uma proposta deste tipo e como os resultados destas podem ser valiosos, devido as características apresentadas anteriormente, como interpretação de relacionamentos e recuperação de informações implícitas.

Após a análise da literatura disponível, este trabalho define a linguagem Prolog como a ferramenta utilizada para inferir informações de documentos XML devido às vantagens apresentadas pela mesma, entre elas a maturidade já alcançada pelo projeto.

4 Controle de Inferência no XPerseus

Como citado anteriormente, a ferramenta XPerseus (Silva, 2011) possui uma interface gráfica para a visualização de diferenças entre documentos XML e inicialmente não possuía funcionalidades dedicadas a inferência de informações sobre as mudanças detectadas. Era dedicado ao controle de versões destes documentos, centrado na identificação dos itens elementos modificados.

O trabalho em desenvolvimento em Siervi (2011), adiciona novos aspectos a ferramenta, como a opção de mesclar diferentes versões de um arquivo XML, fortalecendo a característica de controle de versão do XPerseus.

Observando as alternativas disponíveis para o controle da evolução de documentos XML e as propostas apresentadas anteriormente, este trabalho dedica-se a disponibilizar um estudo e um módulo no contexto do XPerseus que torne possível esta funcionalidade baseada em inferência de informações utilizando a linguagem Prolog.

4.1 Visão Geral

Observando as alternativas disponíveis para o controle da evolução de documentos XML e as propostas apresentadas anteriormente para inferência de dados, a linguagem Prolog foi escolhida para constituir o motor de inferência deste trabalho, dada a maturidade do projeto, fácil aplicação no contexto e a fácil assimilação do método para construir regras.

A expressividade da linguagem herdada da lógica e sua característica não-determinística, ou seja, a possibilidade de se obter múltiplos resultados para a mesma proposta, também valorizam a sua escolha para a tarefa de recuperação de conhecimentos implícitos.

Embora seja necessário um especialista em Prolog para criar as regras de negócio para o contexto em que esta abordagem seja utilizada, a linguagem não impõe grandes obstáculos para o aprendizado. Sua utilização após o domínio das funções básicas se torna intuitiva.

A utilização desta linguagem para realizar a tarefa de inferência proposta também

possui desvantagens, como a possibilidade de ocorrência de laços infinitos de execução e o alto tempo de processamento necessário para a análise de grandes estruturas de dados.

A recuperação das informações não representadas pelos documentos XML é realizada através do método de dedução lógica da linguagem Prolog, também chamada de sequência de refutações. Este método é na verdade a prova do teorema representado pela requisição do usuário, uma consulta por exemplo, com base nos fatos e regras do programa (Palazzo, 1997).

4.2 Abordagem Proposta

A abordagem proposta para este trabalho pode ser visualizada na figura 4.1, onde pode-se dividir o funcionamento da aplicação em três etapas. A primeira etapa é responsável pela construção da base de informações, formada por fatos Prolog. A segunda etapa consiste na análise do contexto de negócio e construção de regras de inferência, que vão estabelecer o que pode ser inferido dos documentos escolhidos. A última etapa consiste em aplicar as regras criadas anteriormente na base de fatos fornecida pelo tradutor de XML em Prolog.

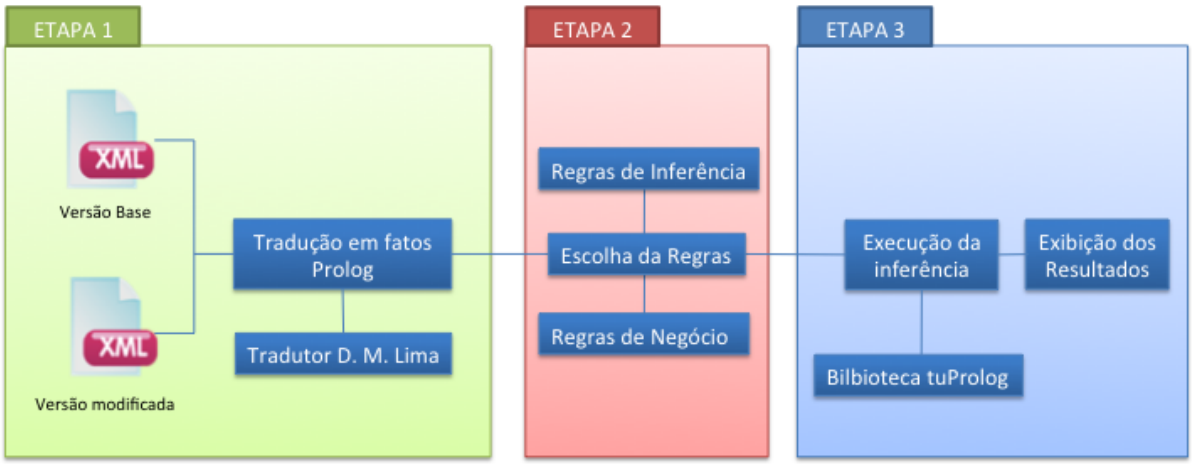

Figura 4.1: Abordagem proposta

O trabalho de Mury et al. (2011) forma a base necessária para a primeira etapa citada anteriormente, a tradução dos documentos XML em uma base de dados. Esta base, representada em um único arquivo, é constituída de fatos Prolog para a aplicação das regras automáticas ou manuais criadas para o contexto selecionado, permitindo ao usuário realizar consultas.

Na abordagem apresentada por Mury et al. (2011), a tradução produz vários fatos a partir de um único documento XML, transformando os elementos em predicados e seus conteúdos em constantes. Para relacionar predicados diferentes, um parâmetro é adicionado, na verdade se trata de um identificador (uma constante Prolog), caracterizando que há correspondência entre esses dados no documento XML (por exemplo, relação de voo/avião ou qual avião é utilizado em um determinado voo).

Para ilustrar esse processo tradutor, será utilizado o documento XML descrito na listagem 4.1.

Listagem 4.1: Exemplo de um documento XML

```
1  <empresaArea>
2          <frota>
3                  <operacao>Venda</operacao>
4                  Em
5                  <data>30/10/2011</data>
6                  <aviao capacidade=``100">
7                          Embraer
8                  </aviao>
9          </frota>
10 </empresaArea>
```

O resultado do processo é descrito resumidamente a seguir, podendo ser observado na listagem 4.2.

Listagem 4.2: Fatos traduzidos

```
1  empresaarea(id1).
2  frota(id1, id2).
3  xml/elementoMisto(id1, 'Em ').
4  operacao(id2, 'venda').
5  data(id2, '30/10/2011').
6  aviao(id2, id3, 'embraer').
7  capacidade(id3, '100').
```

O funcionamento do tradutor é formado por cinco tipos de tradução, começando com a tradução da raiz do documento, o único elemento que não possui pai, como visto na linha 1 da listagem 4.1, e por isso, deve ser tratada de com uma abordagem diferente.

Normalmente a raiz é um elemento composto e com isso, o resultado da tradução destas raízes é um fato com o nome do elemento e argumento único igual a um identificador criado pelo sistema para estabelecer uma relação com seus elementos filhos, representado pela linha 1 da listagem 4.2 (Mury et al., 2011).

O segundo tipo é a tradução de elementos simples sem atributos, que podem ser observados nas linhas 3 e 5 da listagem 4.1, estes elementos possuem o seu conteúdo textual como único filho na árvore XML, assim o resultado é um fato com nome igual ao nome do elemento e argumentos iguais ao identificador do elemento pai e o conteúdo do elemento corrente, como pode ser observado nas linhas 4 e 5 da listagem 4.2.

O terceiro tipo é a tradução de elementos simples com atributo, exemplificados na linhas 6 a 8 da listagem 4.1, onde os atributos são considerados filhos do elemento que os contém, para isso um novo identificador é gerado para que possam referenciar seu pai. O resultado é composto por um fato que representa o elemento, mais um fato para cada atributo existente: o primeiro possui o nome do elemento e três argumentos: o identificador do seu elemento pai, um novo identificador gerado para representá-lo e o seu conteúdo textual, como pode ser visualizado nas linhas 6 e 7 da listagem 4.2.

O quarto tipo é a tradução de elementos complexos, como a marcação frota na linha 2 da listagem 4.1, estes elementos complexos possuem outros elementos como filhos. Portanto, um novo identificador deve ser gerado para relacionar os filhos ao pai. Consequentemente, o resultado da tradução é a geração de um fato com o nome do elemento composto e dois argumentos: o identificador do seu elemento pai e um novo identificador gerado para referenciá-lo, o fato resultado está descrito na linha 2 da listagem 4.2. Seus filhos são analisados normalmente e divididos entre estas regras.

O último tipo é a tradução de elementos mistos, que são manipulados de maneira parecida ao tipo anterior, a não ser pela criação de fatos específicos para os filhos texto do elemento, já que não possuem identificação. Este tipo de marcação pode ser melhor compreendida observando as linhas 2 a 9 da listagem 4.1, em especial a linha 4. Assim, seus filhos do tipo texto são representados por fatos “xml/ElementoMisto” com argumentos iguais ao identificador do elemento pai, o elemento complexo, e o seu conteúdo textual, como visto na linha 3 da listagem 4.2. Os outros filhos do elemento são analisados

novamente dentro das regras que se adequem às suas características.

O processo de tradução analisa cada documento XML separadamente e o resultado deste método é unido em um único arquivo e enviado a próxima etapa do módulo.

Este sistema tradutor foi disponibilizado pelo autor em forma de uma biblioteca para a linguagem Java, para fazer parte da ferramenta construída neste trabalho.

A segunda etapa consiste na criação das regras para a análise dos dados, que também são escritas em Prolog. Estas estruturas, exemplificadas na listagem 4.3, são utilizadas junto aos fatos fornecidos pela etapa anterior, pelo método de execução da linguagem Prolog, descrito na próxima etapa do módulo.

Listagem 4.3: Fatos traduzidos

```
1 rotas_novas_nao_percorridas(NomeRota) :-
2         nome_rota(X,NomeRota),
3         nova_rota(NomeRota),
4         not rotas_percorridas(NomeRota).
```

O papel das regras é identificar relacionamentos entre os fatos gerados pela tradução das duas versões do documento XML analisado. Estes relacionamentos podem ser semelhanças, por exemplo um mesmo elemento presente nas duas versões, diferenças, como um novo elemento inserido ou um antigo removido, e também podem identificar informações implícitas nas relações, por exemplo o significado de uma mudança de um elemento ou da remoção do mesmo.

As regras podem ser caracterizadas como de negócio ou de inferência. Embora não possuam nenhuma diferença sintática ou estrutural, o propósito delas é diferente. As regras de negócio são utilizadas para obter informações e exibir resultados que auxiliam na gerência da evolução dos documentos XML de um determinado contexto. Já as regras de inferência possuem igual importância, mas menor visibilidade, porque não tem o objetivo específico de retornar informações relevantes ao contexto especificado, mas constituem a base das regras que o fazem, as regras de negócio.

É importante notar que os valores dos identificadores inseridos pelo tradutor seguem uma ordem incremental porque dependem do número de elementos identificados no documento XML, além disso, a cada nova ordem para tradução dos documentos, eles

devem se diferenciar das traduções anteriores também, para evitar problemas na execução das regras.

Observando este comportamento, foi estabelecido um padrão para a inserção das regras referentes às duas versões do documento XML. Este padrão define que as referências ao identificador do elemento raiz da primeira versão do documento XML devem ser escritas como "base", e o elemento correspondente na segunda versão deve ser escrito como "modificado", facilitando assim a criação das regras e a leitura das mesmas, caso contrário seria necessário identificar os elementos com identificadores do tipo "id2" ou "id89".

A terceira etapa consiste na aplicação das regras fornecidas na base de dados proveniente dos documentos XML. O que a máquina de inferência Prolog faz é tentar encontrar provas da verdade de cada relação especificada. Normalmente, a relação conterá variáveis e parte do processo envolverá encontrar valores para variáveis que tornem verdadeira a relação (Rich, 1988).

O processo de execução da linguagem Prolog é inteiramente simulado pela biblioteca tuProlog (Denti et al., 2005), disponível na linguagem Java e de código aberto, esta ferramenta possibilita a inserção dos fatos e regras das etapas anteriores em uma estrutura onde é possível obter os resultados e assim retorna-los à aplicação.

Para realizar este processo completo na ferramenta XPerseus, que tem sua interface principal representada na figura 4.2, é necessário ativar o módulo de inferência através do menu superior, "Módulo de inferência", e selecionar o botão "Ativar".

É possível notar a já dividida área gráfica da aplicação em três setores e a mudança de alguns botões após a ativação como na figura 4.3. Os dois setores superiores na interface são utilizados para carregar as duas versões do documento XML a ser analisado.

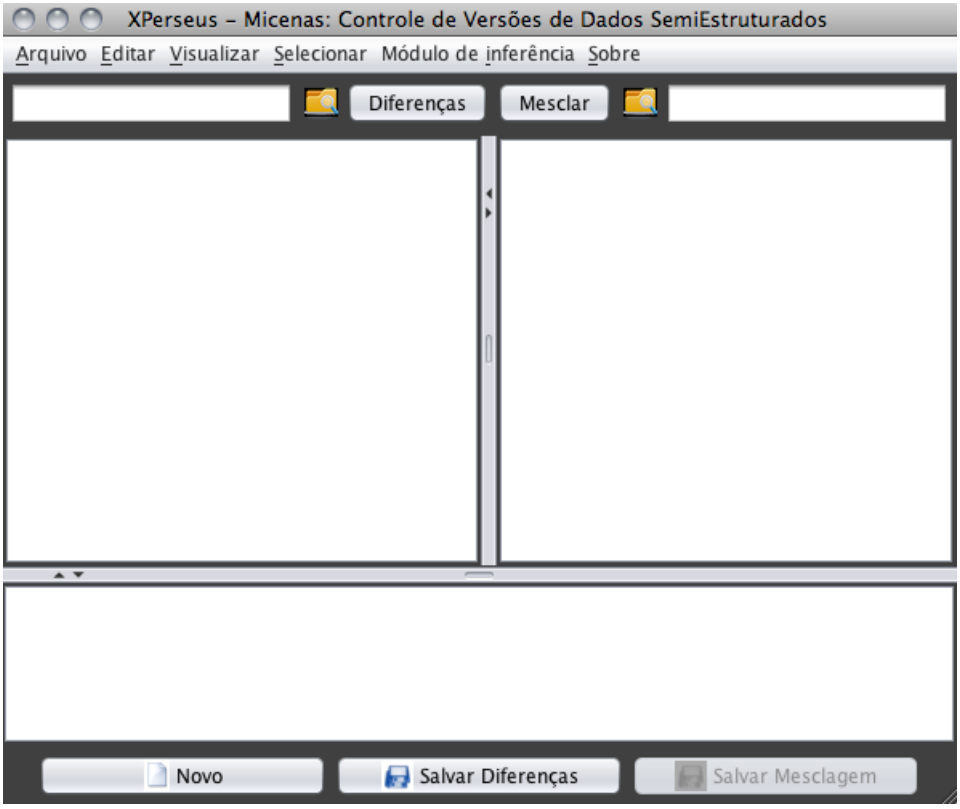

Figura 4.2: Interface principal

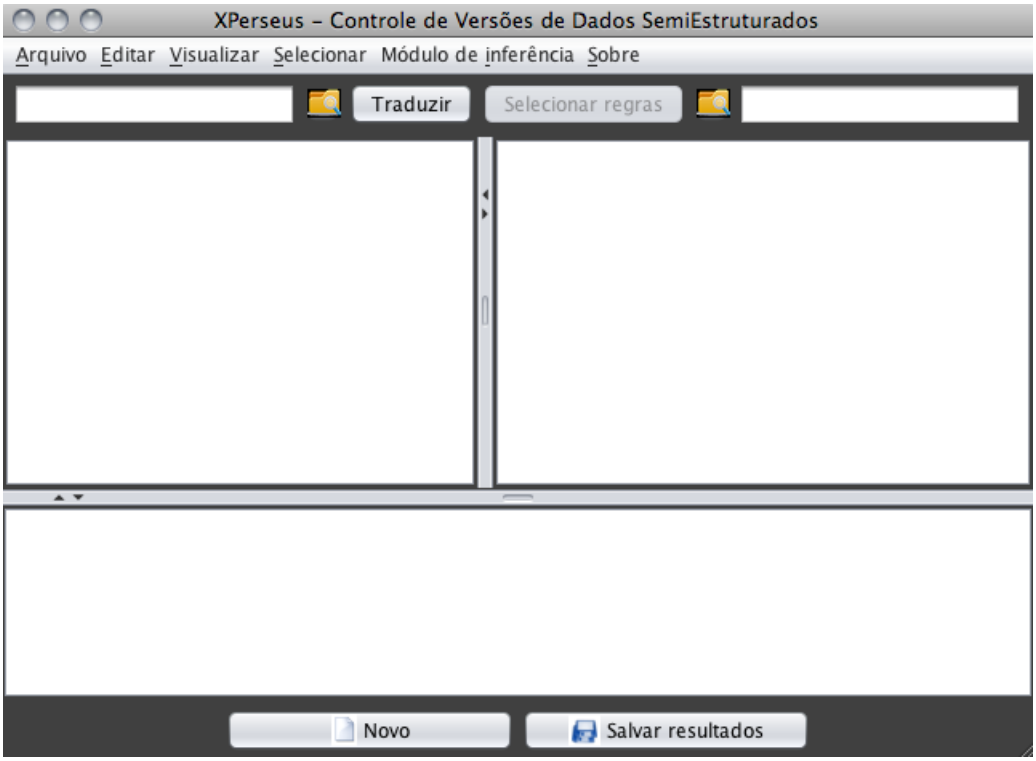

Figura 4.3: Interface principal do XPerseus com módulo de inferência ativado

Após a seleção dos arquivos, inicia-se o processo de tradução ao utilizar o botão "Traduzir", que mostra o resultado em fatos no setor inferior da aplicação, como representado pela figura 4.4. Após a etapa de seleção e interpretação das regras de inferência, são exibidos os resultados da inferência nesta mesma área.

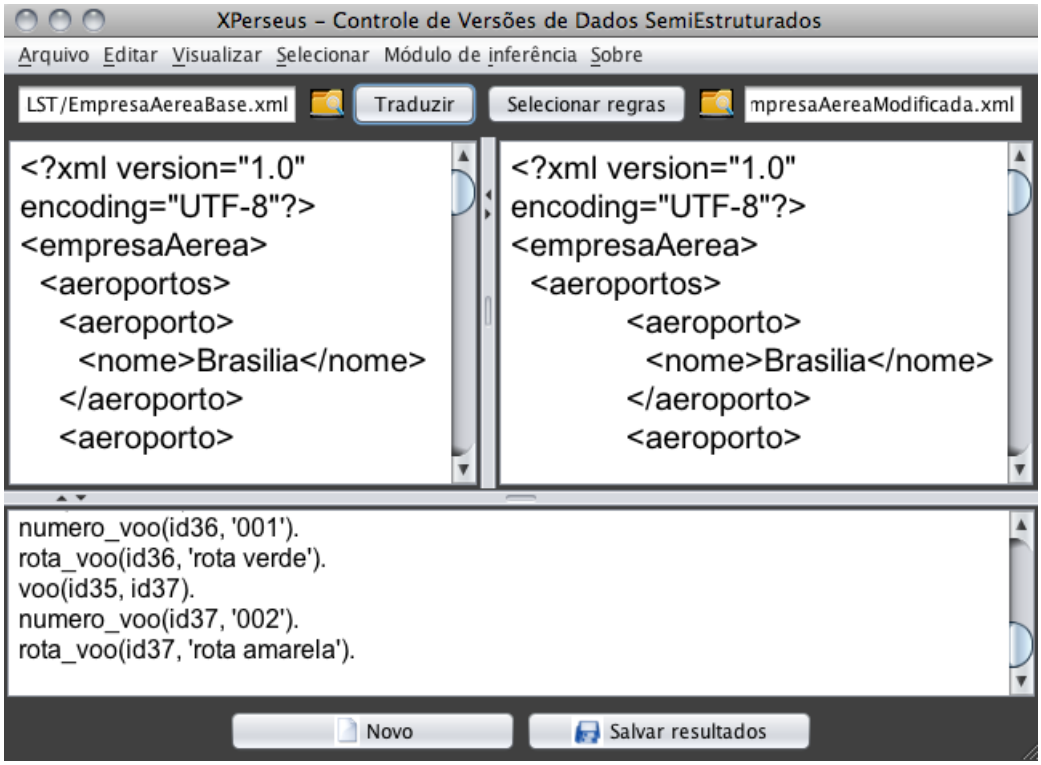

Figura 4.4: XPerseus e a exibição dos fatos traduzidos

Para dar prosseguimento a inferência de informações, deve-se clicar no botão "Selecionar regras", então uma nova janela se abre, como ilustrado na figura 4.5, dispondo de uma área de texto para a inserção do arquivo de regras específicas ao contexto escolhido.

Figura 4.5: XPerseus e a seleção das regras de inferência

Após a seleção do arquivo com as regras, é possível filtrar a visualização do resultado da interpretação de algumas delas, após esta escolha, deve-se concluir o processo com o botão "Concluir".

O resultado do processo de inferência é exibido na tela principal da aplicação, como ilustrado na figura 4.6.

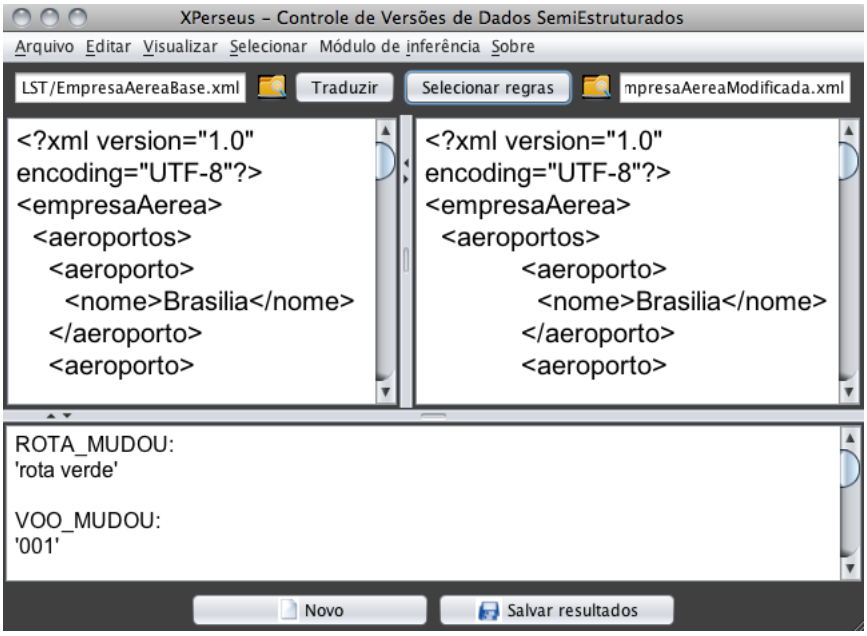

Figura 4.6: XPerseus e a exibição dos resultados da inferência

Resumidamente, a proposta consiste em: 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.

4.3 Estudo de Caso

Para melhor compreendimento do funcionamento do módulo de inferência criado para o XPerseus, um estudo de caso foi criado para ilustrar a utilização do mesmo.

O contexto escolhido é a atividade simplificada de uma empresa aérea, onde a primeira versão do documento XML analisado representa algumas informações básicas armazenadas, como pode ser visto na listagem A.1. É possível notar que estão presentes neste arquivo os aeroportos disponíveis para receber um voo da empresa, assim como as rotas permitidas entre eles e os voos que ela efetivamente realiza.

Então uma evolução desta atividade foi simulada na listagem A.2, onde uma nova rota foi inserida, chamada de "Rota Azul", novos aeroportos adicionados, são eles "Pampulha" e "Serrinha", e uma rota já existente foi alterada, a "Rota Verde".

A tradução completa destes dois documentos XML para Prolog utilizando o módulo de inferência do XPerseus tem como resultado uma lista de fatos como exibido na listagem A.3.

Observando o contexto e a estrutura dos arquivos analisados, foram criadas regras para recuperar informações que possam auxiliar no compreendimento das mudanças dos documentos. Todas as regras criadas podem ser observadas na listagem A.4.

Analisando esta lista de regras por partes, é possível verificar que o trecho, ilustrado na listagem 4.4, exibe as regras de inferência criadas, ou seja, regras com o único propósito de auxiliar na construção de regras de negócio mais complexas.

Dentre estas regras é possível identificar regras chamadas "id_aeroporto_base" e "id_aeroporto_mod", e o mesmo tipo de construção para os elementos rota e voo. O propósito destas regras é encontrar o identificador atribuído pelo tradutor a estas marcações, tanto na versão base do documento XML e na versão modificada.

Além destas, as regras "mesmo_aeroporto", "mesmo_voo" e "mesma_rota" tem o objetivo de recuperar os identificadores atribuídos a um mesmo objeto nas versões base e modificada do documento XML analisado, assim, é possível comparar as diferenças

destes dois elementos através do ambiente Prolog. As regras depois destas se referem a manipulação de listas no contexto escolhido.

Listagem 4.4: Regras de inferência

```
1  id_aeroportos_base(ID) :- aeroportos(base,ID).
2  id_aeroportos_mod(ID) :- aeroportos(modificado,ID).
3  (...)
4
5  mesma_rota(Rb,Rm,Nome) :-
6          id_rotas_base(X),
7          id_rotas_mod(Y),
8          rota(X,Rb),
9          rota(Y,Rm),
10         nome_rota(Rb,Nome),
11         nome_rota(Rm,Nome).
12
13 (...)
14
15 concatenar([],L,L):-!.
16 concatenar([X|M],L,[X|Y]):-concatenar(M,L,Y).
17
18 elim12([],L,L):-!.
19 elim12([X|M],L,S):-elim12(M,T,S).
20
21 (...)
```

Entre as regras de inferência e as de negócio, podem existir algumas estruturas que além de auxiliar na formação de outras regras mais complexas, também possibilitam a recuperação direta de informações relevantes para o controle de mudanças do projeto que envolve os documentos analisados. Estas regras estão exibidas a seguir na listagem 4.5.

Listagem 4.5: Regras de inferência e de negócio

```
1  rota_mudou(Nome):-
2          mesma_rota(A,B,Nome),
3          aeroportos_visitados(A, C),
4          aeroportos_visitados(B, D),
5          visitado(D,NovoPonto),
```

```
6           not visitado(C,NovoPonto).
7
8   voo_mudou(X):-
9           mesmo_voo(A,B,Numero),
10          rota_voo(A,RotaA),
11          rota_mudou(RotaA),
12          numero_voo(A, X).
13
14  novo_aeroporto(X) :-
15          id_aeroportos_mod(ID),
16          aeroporto(ID, Y),
17          not mesmo_aeroporto(W, Y, Nome),
18          nome(Y, X).
19
20  nova_rota(NomeRota) :-
21          id_rotas_mod(ID),
22          rota(ID,Y),
23          not mesma_rota(W, Y, Nome),
24          nome_rota(Y,NomeRota).
25
26  novo_voo(X) :-
27          id_voo_mod(ID),
28          voo(ID,Y),
29          not mesmo_voo(W,Y,Numero),
30          numero(Y,X).
31
32  aeroportos_atendidos(Visitado) :-
33          voos(modificado,X),
34          voo(X,IdVoo),
35          rota_voo(IdVoo,NomeRota),
36          rotas(modificado,T),
37          rota(T,IdRota),
38          nome_rota(IdRota,NomeRota),
39          aeroportos_visitados(IdRota, IdVisitados),
40          visitado(IdVisitados,Visitado).
```

É possível notar a presença de regras que identificam a presença de novos ele-

mentos na segunda versão do arquivo analisado. São elas “novo_aeroporto”, “nova_rota” e “novo_voo”. Além destas, estão definidas as regras que são capazes de identificar uma mudança em um itinerário de uma rota ou de um voo, em “voo_mudou” e “rota_mudou”. Por fim, a regra “aeroportos_atendidos” auxilia na construção das listas utilizadas por outras regras.

O resultado da aplicação destas regras segue na listagem 4.6

Listagem 4.6: Resultado das regras de inferência e de negócio

```
1  NOVA_ROTA:
2  'rota azul'
3
4  NOVO_AEROPORTO:
5  pampulha
6  serrinha
7
8  NOVO_VOO:
9
10 ROTA_MUDOU:
11 'rota verde'
12
13 VOO_MUDOU:
14 '001'
15
16 AEROPORTOS_ALCANCAVEIS_ROTA:
17 brasilia
18 guarulhos
19 fiumicino
20 galeao
21 congonhas
22 brasilia
23 confins
24 cumbica
```

A regra “rotas_percorridas”, ilustrada a seguir na listagem 4.7, tem como objetivo identificar todas as rotas que são utilizadas em um ou mais voos da empresa aérea representada pelo exemplo.

Listagem 4.7: Regra rotas_percorridas

```
1 rotas_percorridas(NomeRota):-
2         voos(modificado,X),
3         voo(X,IdVoo),
4         rota_voo(IdVoo,NomeRota),
5         rotas(modificado,IdRotas),
6         rota(IdRotas,IdRota),
7         nome_rota(IdRota,NomeRota).
```

O resultado da execução desta regra neste estudo de caso segue na listagem 4.8.

Listagem 4.8: Resultado de rotas_percorridas

```
1 ROTAS_PERCORRIDAS:
2 'rota verde'
3 'rota amarela'
```

A regra "rotas_novas_nao_percorridas", a seguir na listagem 4.9, permite a visualização das novas rotas adicionadas à versão modificada do documento XML mas que não são utilizadas por nenhum voo. As rotas que satisfazem este estado e obtidas pela aplicação seguem na listagem 4.10.

Listagem 4.9: Regra rotas_novas_nao_percorridas

```
1 rotas_novas_nao_percorridas(NomeRota) :-
2         nome_rota(X,NomeRota),
3         nova_rota(NomeRota),
4         not rotas_percorridas(NomeRota).
```

Listagem 4.10: Resultado rotas_novas_nao_percorridas

```
1 ROTAS_NOVAS_NAO_PERCORRIDAS:
2 'rota azul'
```

A regra "lista_aeroportos_atendidos", listagem 4.11, trata da exibição dos aeroportos que são atendidos pela empresa aérea, ou seja, onde pelo menos um voo utiliza uma rota que atinge este aeroporto, o resultado desta cláusula é apresentada na listagem 4.12.

Listagem 4.11: Regra lista_aeroportos_atendidos

```
1 lista_aeroportos_atendidos(Lista):-
2         findall(Visitado,aeroportos_atendidos(Visitado),ListaVisitados),
3         eliminaRep(ListaVisitados,Lista).
```

Listagem 4.12: Resultado lista_aeroportos_atendidos

```
1 LISTA_AEROPORTOS_ATENDIDOS:
2 [brasilia,guarulhos,fiumicino,galeao,congonhas,confins,cumbica]
```

A regra "lista_aeroportos_alcancaveis", representada na listagem 4.13, exibe o conjunto de aeroportos que pertencem a pelo menos um rota da empresa aérea. A informação obtida com essa regra segue na listagem 4.14.

Listagem 4.13: Regra lista_aeroportos_alcancaveis

```
1 lista_aeroportos_alcancaveis(ListaAeroportos) :-
2         findall(Visitado,visitado(_,Visitado),Lista),
3         eliminaRep(Lista,ListaAeroportos).
```

Listagem 4.14: Resultado lista_aeroportos_alcancaveis

```
1 LISTA_AEROPORTOS_ALCANCAVEIS:
2 [brasilia,guarulhos,fiumicino,galeao,confins,cumbica,congonhas,serrinha]
```

Seguindo este processo de tradução dos documentos XML, criação das regras e inserção das regras no módulo, é possível obter os resultados como os listados em A.5.

4.4 Conclusão

Neste capítulo foi realizada uma análise da proposta e da aplicação, onde é possível observar que pode-se inferir diferentes tipos de informação de documentos XML utilizando a linguagem Prolog. Estas informações podem ser utilizadas para auxiliar na gerência de mudanças de um projeto, ampliando a base de conhecimento disponível além das alterações identificadas.

Esse conhecimento adquirido pode refletir o real motivo, ou parte dele, da evolução do documento analisado, fator importante na GCS.

Além das vantagens da utilização desta proposta já mencionadas, a definição do contexto e a construção das regras podem ser realizadas em uma única oportunidade, e após o termino destas, a aplicação pode ser utilizada inúmeras vezes no cenário escolhido.

5 Considerações Finais

A gerência de controle de mudanças caracteriza-se pelo controle da evolução dos IC de um projeto, analisando principalmente as razões das mudanças identificadas. Para auxiliar nesta tarefa, foram propostos sistemas específicos deste contexto, mas ainda é possível observar a necessidade por formas mais complexas de análise destas mudanças. Por esta razão, optou-se neste trabalho uma abordagem que busque uma interpretação dos documentos XML em alto nível, como a inferência de informações.

Existem muitas abordagens diferentes para realizar uma recuperação de informações sobre uma base de dados. Este trabalho apresentou uma proposta com o objetivo de auxiliar o controle da evolução de arquivos XML, utilizando a linguagem Prolog para recuperar informações implícitas nestes documentos, favorecendo assim a gerência de projetos deste contexto.

Para ilustrar o funcionamento desta abordagem, foi criado um módulo para a ferramenta XPerseus, anteriormente dedicada exclusivamente ao controle de versões de documentos XML. Este módulo, criado com a linguagem Java, utiliza um tradutor de documentos XML para Prolog e uma plataforma capaz de interpretar código Prolog dentro do ambiente da máquina virtual Java. Através de duas versões de um arquivo XML e de regras criadas por um especialista para o contexto analisado, o sistema permite inferir informações que não estão explicitamente representadas.

Como forma de evolução e trabalhos futuros, seria vantajoso realizar um estudo sobre como analisar não somente documentos XML, mas também *XML Schemas*, ou seja, modelos que definem de forma básica toda ou parte da estrutura destes documentos, sendo assim possível a criação de um conjunto mais complexo de regras para recuperar mais informações sobre a evolução dos documentos analisados, visto que a funcionalidade do tradutor utilizado não diferencia tipos de dados como números, datas ou valores lógicos.

O próprio tradutor citado não foi explorado inteiramente. A opção existente neste para analisar *XML Schemas* pode ser utilizada para auxiliar futuramente em novas etapas do XPerseus, além disso, através do trabalho de Mury et al. (2011) também é possível a

inferência de algumas regras Prolog referentes aos tipos definidos pelo modelo.

Existem mais características da linguagem Prolog que podem ser exploradas em trabalhos futuros, por exemplo após a criação da base de conhecimento e definição das regras referentes ao contexto do projeto onde é possível obter resultados ainda mais específicos através da formulação de consultas Prolog caso seja criado um funcionalidade para tal, como um terminal específico do módulo de inferência para a ferramenta XPerseus.

A criação de funcionalidades mais complexas como a automação da etapa de construção de regras, ou parte dela, também pode trazer resultados ainda melhores. Isto pode ser feito através de uma maior integração com uma ferramenta de controle de versões, de forma que seja possível analisar as razões das mudanças identificadas individualmente e então tentar identificar um padrão com base nestas modificações.

Estas funcionalidades visam uma maior interação com o usuário e o controle de mudanças, possibilitando o acompanhamento do processo de maneira mais natural, dispensando ou amenizando o conhecimento técnico específico para a manipulação da linguagem Prolog.

Referências Bibliográficas

Almeida, M. B.; Bax, M. P. Uma visão geral sobre ontologias: pesquisa sobre definições, tipos, aplicações, métodos de avaliação e de construção. **Ci. Inf. vol.32 no.3 Brasília Sept./Dec. 2003**, 2003.

Apache Software Foundation. **Apache Ant**. `http://ant.apache.org/`, 2000.

Apache Software Foundation. **Apache Maven**. `http://maven.apache.org/`, 2011.

Bray, T.; Paoli, J.; Sperberg-McQueen, C. M.; Maler, E. ; Yergeau, F. Extensible Markup Language (XML) 1.0 (Fifth Edition). Nov 2008.

Cecchin, F.; Hara, C. Uma interface gráfica para algoritmos de detecção de diferenças entre documentos XML. **Escola Regional de Banco de Dados (ERBD)**, 2009.

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

Coelho, J. M. N. Processamento de XML em Programação em Lógica. **Universidade do Porto**, 2002.

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

Costa, F. H. T. ROSAI: Uma Proposta de Representação do Modelo ROSA em Linguagem Lógica. **Tese de Mestrado, IME**, abril 2005.

Cunha, A.; Albrecht, F. ; Fernandes, R. Q. A. Inferência sobre Ontologias: o reencontro com PROLOG. **Rio de Janeiro, IME**, 2007.

Cunha, C. O. Uso de Datalog para Realizar Inferências em Documentos Semiestruturados. Monografia em desenvolvimento. **UFJF**, 2011.

Dantas, C. Gerência de Configuração de Software - Desenvolvendo software de forma eficiente e disciplinada. **Engenharia de Software Magazine**, v.2, 2010.

Dart, S. **Concepts in configuration management systems**. In: Proceedings of the 3rd international workshop on Software configuration management, SCM '91, New York, NY, USA, 1991. ACM.

Dean, M.; Schreiber, G.; Bechhofer, S.; van Harmelen, F.; Hendler, J.; Horrocks, I.; McGuinness, D. L.; Patel-Schneider, P. F. ; Andrea, L. **SWRL: A Semantic Web Rule Language Combining OWL and RuleML**. `http://www.w3.org/Submission/SWRL/`, 2004.

Denti, E.; Omicini, A. ; Ricci, A. Multi-paradigm Java-Prolog integration in tuProlog. **Science of Computer Programming**, v.57, n.2, p. 217 – 250, 2005.

Elmasri, R. E.; Navathe, S. **Sistemas de Banco de Dados**. 4. ed., 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.

Feldman, S. Make - A Program for Maintaining Computer Programs. **v9, n. 4 (April), pp.255-265**, 1979.

Fontes, C. A. **Explorando Inferência em um Sistema de Anotação Semântica**. 2011. Dissertação de Mestrado - IME.

Freitas, F. L. G. Ontologias e a Web Semântica. **Pós-Graduação em Informátca - UniSantos**, 2003.

Giacomel, F. S. XVersion - uma ferramenta gráfica para gerenciamento e consulta de versões de documentos XML. **UFRGS**, 2006.

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

Institute of Electrical Electronics Engineers. **Std 610.12 - IEEE Standard Glossary of Software Engineering Terminology**. IEEE, 1990.

Institute of Electrical and Electronics Engineers. **IEEE Standard for Software Configuration Management Plans**. IEEE, 2005.

Korth, H. F.; Silberschatz, A. ; Sudarshan, S. **Sistemas de Banco de Dados**. 5. ed., Elsevier, 2006.

Laender, A. H. F.; Moro, M. M.; Nascimento, C. ; Martins, P. An X-Ray on Web-Available XML Schemas. **SIGMOD Record**, v.38, n.1, March 2009.

Lindholm, T. A three-way merge for XML documents. **In Symposium on Document Engineering**, p. 1–10, 2004.

LogicBlox. **Logicblox**. `http://www.logicblox.com/`, 2011.

Mantis. **Mantis Bug Tracker**. `http://www.mantisbt.org/`, 2006.

Marian, A.; Abiteboul, S.; Cobena, G. ; Mignet, L. Change-centric management of versions in an XML warehouse. **In International Conference on Very Large Data Bases, VLDB**, p. 581–590, 2001.

Mattos, D.; Moura, A. M. C. ; Cavalcanti, M. C. ROSA+: Um Repositório de Objetos de Aprendizagem com Suporte a Inferência e Regras. **XXI Simpósio Brasileiro de Banco de Dados**, 2006.

McGuinness, D. L.; van Harmelen, F. **OWL: Web Ontology Language Overview**. Disponível em: `http://www.w3.org/TR/owl-features/`, Último acesso em Novembro 2011, 2004.

Moro, M.; Braganholo, V. **Desmistificando XML: da pesquisa à prática industrial**. In: Atualização em Informática, p. Cap. 5 p. 231–278. SBC, 2009.

Communications, N. **Bugzilla**. `http://www.bugzilla.org/`, 1998.

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

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

Oliveira, A. P. Gerenciando Alterações em Documentos XML. **Projeto final de Curso - UFRJ**, 2007.

Oliveira, A. P.; Oliveira, A. M.; Braganholo, V. ; Murta, L. Gerenciando Alterações em Documentos XML. **COPPE-UFRJ**, 2011.

Palazzo, L. A. M. **Introdução à Programação Prolog**. 1. ed., EDUCAT - Editora da Universidade Católica de Pelotas, 1997.

Parsia, B.; Sirin, E. Pellet: An OWL DL Reasoner. 2004.

Perini, A. B. XSDelta: Uma ferramenta visual para comparação de esquemas XML. 2006.

Porto, F.; Moura, A. M. C. ROSA: A Data Model and Query Language for e-Learning Objects. **PUC-Rio**, 2006.

Pressman, R. S. **Engenharia de Software - Uma Abordagem Profissional**. 7. ed., ARTMED - McGraw-Hill, 2011.

Rich, E. **Inteligência Artificial**. 2. ed., McGraw-Hill, 1988.

Santos, R. C. XKeyMatch : um algoritmo semântico para detecção de diferenças entre documentos XML. 2006.

Santos, D. S. M.; Soares, I. P. ; Matos, R. S. Semantic Web Reasoning. Junho 2008.

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

Siervi, B. M. Mesclagem intuitiva de documentos XML com XPerseus - Micenas. Monografia em desenvolvimento. **UFJF**, 2011.

Silva, R. B. XPerseus: Uma Interface Gráfica para Detecção de Diferenças entre Documentos XML. Julho 2011.

Sommerville, I. **Engenharia de Software**. 9. ed., Pearson, 2011.

Tani, R.; Molli, P. ; Jouille, F. **JXyDiff LibreSource**. `https://github.com/tanob/jxydiff`, 2011.

Tichy, W. F. **Design, implementation, and evaluation of a Revision Control System**. In: Proceedings of the 6th international conference on Software engineering, ICSE '82, p. 58–67, Los Alamitos, CA, USA, 1982. IEEE Computer Society Press.

Tran, H.; Lange, C.; Chulkov, G.; Schowalder, J. ; Kohlhase, M. Applying Semantic Techniques to Search and Analyze Bug Tracking Data. **Journal of Network and Systems Management**, 2009.

Software, E. **Trac**. `http://trac.edgewall.org//`, 2011.

Zhao, Q.; Chen, L.; Bhowmick, S. ; Madria, S. XML structural delta mining: Issues and challenges. **Elsevier Science**, 2005.

A Apêndice

A seguir seguem as versões completas dos documentos utilizados no estudo de caso descrito no Capítulo 4. Primeiro os documentos XML utilizados para realizar a análise.

Listagem A.1: Exemplo de um documento XML - versão base

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <empresaAerea>
3    <aeroportos>
4      <aeroporto>
5        <nome>Brasilia</nome>
6      </aeroporto>
7      <aeroporto>
8        <nome>Guarulhos</nome>
9      </aeroporto>
10     <aeroporto>
11       <nome>Congonhas</nome>
12     </aeroporto>
13     <aeroporto>
14       <nome>Confins</nome>
15     </aeroporto>
16     <aeroporto>
17       <nome>Fiumicino</nome>
18     </aeroporto>
19     <aeroporto>
20       <nome>Cumbica</nome>
21     </aeroporto>
22     <aeroporto>
23       <nome>Galeao</nome>
24     </aeroporto>
25   </aeroportos>
26
27   <rotas>
28     <rota>
```

```
29          <nome_rota>Rota Verde</nome_rota>
30                  <aeroportos_visitados>
31                          <visitado>Brasilia</visitado>
32                          <visitado>Guarulhos</visitado>
33                          <visitado>Fiumicino</visitado>
34                          <visitado>Galeao</visitado>
35                  </aeroportos_visitados>
36      </rota>
37      <rota>
38          <nome_rota>Rota Amarela</nome_rota>
39                  <aeroportos_visitados>
40                          <visitado>Brasilia</visitado>
41                          <visitado>Confins</visitado>
42                          <visitado>Cumbica</visitado>
43                  </aeroportos_visitados>
44      </rota>
45    </rotas>
46
47    <voos>
48      <voo>
49          <numero_voo>001</numero_voo>
50          <rota_voo>Rota Verde</rota_voo>
51      </voo>
52      <voo>
53          <numero_voo>002</numero_voo>
54          <rota_voo>Rota Amarela</rota_voo>
55      </voo>
56    </voos>
57  </empresaAerea>
```

Listagem A.2: Exemplo de um documento XML - versão modificada

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <empresaAerea>
3    <aeroportos>
4          <aeroporto>
5            <nome>Brasilia</nome>
```

```
6         </aeroporto>
7         <aeroporto>
8           <nome>Guarulhos</nome>
9         </aeroporto>
10        <aeroporto>
11          <nome>Congonhas</nome>
12        </aeroporto>
13        <aeroporto>
14          <nome>Confins</nome>
15        </aeroporto>
16        <aeroporto>
17          <nome>Fiumicino</nome>
18        </aeroporto>
19        <aeroporto>
20          <nome>Cumbica</nome>
21        </aeroporto>
22        <aeroporto>
23          <nome>Galeao</nome>
24        </aeroporto>
25    <aeroporto>
26      <nome>Pampulha</nome>
27    </aeroporto>
28        <aeroporto>
29          <nome>Serrinha</nome>
30        </aeroporto>
31  </aeroportos>
32
33  <rotas>
34        <rota>
35      <nome_rota>Rota  Verde</nome_rota>
36                <aeroportos_visitados>
37                        <visitado>Brasilia</visitado>
38                        <visitado>Guarulhos</visitado>
39                        <visitado>Fiumicino</visitado>
40                        <visitado>Galeao</visitado>
41                        <visitado>Congonhas</visitado>
```

```
42                  </aeroportos_visitados>
43          </rota>
44          <rota>
45        <nome_rota>Rota Amarela</nome_rota>
46                  <aeroportos_visitados>
47                          <visitado>Brasilia</visitado>
48                          <visitado>Confins</visitado>
49                          <visitado>Cumbica</visitado>
50                  </aeroportos_visitados>
51          </rota>
52          <rota>
53          <nome_rota>Rota Azul</nome_rota>
54                  <aeroportos_visitados>
55                          <visitado>Galeao</visitado>
56                          <visitado>Serrinha</visitado>
57                          <visitado>Brasilia</visitado>
58                  </aeroportos_visitados>
59      </rota>
60    </rotas>
61
62    <voos>
63      <voo>
64          <numero_voo>001</numero_voo>
65          <rota_voo>Rota Verde</rota_voo>
66      </voo>
67      <voo>
68          <numero_voo>002</numero_voo>
69          <rota_voo>Rota Amarela</rota_voo>
70      </voo>
71    </voos>
72  </empresaAerea>
```

A seguir todos os fatos obtidos após a tradução dos arquivos XML inseridos na aplicação.

Listagem A.3: Fatos traduzidos

```
1  empresaaerea(base).
```

```
2  aeroportos(base, id2).
3  aeroporto(id2, id3).
4  nome(id3, 'brasilia').
5  aeroporto(id2, id4).
6  nome(id4, 'guarulhos').
7  aeroporto(id2, id5).
8  nome(id5, 'congonhas').
9  aeroporto(id2, id6).
10 nome(id6, 'confins').
11 aeroporto(id2, id7).
12 nome(id7, 'fiumicino').
13 aeroporto(id2, id8).
14 nome(id8, 'cumbica').
15 aeroporto(id2, id9).
16 nome(id9, 'galeao').
17 rotas(base, id10).
18 rota(id10, id11).
19 nome_rota(id11, 'rota verde').
20 aeroportos_visitados(id11, id12).
21 visitado(id12, 'brasilia').
22 visitado(id12, 'guarulhos').
23 visitado(id12, 'fiumicino').
24 visitado(id12, 'galeao').
25 rota(id10, id13).
26 nome_rota(id13, 'rota amarela').
27 aeroportos_visitados(id13, id14).
28 visitado(id14, 'brasilia').
29 visitado(id14, 'confins').
30 visitado(id14, 'cumbica').
31 voos(base, id15).
32 voo(id15, id16).
33 numero_voo(id16, '001').
34 rota_voo(id16, 'rota verde').
35 voo(id15, id17).
36 numero_voo(id17, '002').
37 rota_voo(id17, 'rota amarela').
```

```
38
39 empresaaerea(modificado).
40 aeroportos(modificado, id18).
41 aeroporto(id18, id19).
42 nome(id19, 'brasilia').
43 aeroporto(id18, id20).
44 nome(id20, 'guarulhos').
45 aeroporto(id18, id21).
46 nome(id21, 'congonhas').
47 aeroporto(id18, id22).
48 nome(id22, 'confins').
49 aeroporto(id18, id23).
50 nome(id23, 'fiumicino').
51 aeroporto(id18, id24).
52 nome(id24, 'cumbica').
53 aeroporto(id18, id25).
54 nome(id25, 'galeao').
55 aeroporto(id18, id26).
56 nome(id26, 'pampulha').
57 aeroporto(id18, id27).
58 nome(id27, 'serrinha').
59 rotas(modificado, id28).
60 rota(id28, id29).
61 nome_rota(id29, 'rota verde').
62 aeroportos_visitados(id29, id30).
63 visitado(id30, 'brasilia').
64 visitado(id30, 'guarulhos').
65 visitado(id30, 'fiumicino').
66 visitado(id30, 'galeao').
67 visitado(id30, 'congonhas').
68 rota(id28, id31).
69 nome_rota(id31, 'rota amarela').
70 aeroportos_visitados(id31, id32).
71 visitado(id32, 'brasilia').
72 visitado(id32, 'confins').
73 visitado(id32, 'cumbica').
```

```
74 rota(id28, id33).
75 nome_rota(id33, 'rota azul').
76 aeroportos_visitados(id33, id34).
77 visitado(id34, 'galeao').
78 visitado(id34, 'serrinha').
79 visitado(id34, 'brasilia').
80 voos(modificado, id35).
81 voo(id35, id36).
82 numero_voo(id36, '001').
83 rota_voo(id36, 'rota verde').
84 voo(id35, id37).
85 numero_voo(id37, '002').
86 rota_voo(id37, 'rota amarela').
```

A listagem A.4 representa de forma completa todas as regras utilizadas.

Listagem A.4: Regras criadas

```
1  id_aeroportos_base(ID) :- aeroportos(base,ID).
2  id_aeroportos_mod(ID) :- aeroportos(modificado,ID).
3
4  id_rotas_base(ID) :- rotas(base,ID).
5  id_rotas_mod(ID) :- rotas(modificado,ID).
6
7  id_voos_base(ID) :- voos(base,ID).
8  id_voos_mod(ID) :- voos(modificado,ID).
9
10 mesma_rota(Rb,Rm,Nome) :-
11         id_rotas_base(X),
12         id_rotas_mod(Y),
13         rota(X,Rb),
14         rota(Y,Rm),
15         nome_rota(Rb,Nome),
16         nome_rota(Rm,Nome).
17
18 mesmo_aeroporto(Ab,Am,Nome) :-
19         id_aeroportos_base(X),
20         id_aeroportos_mod(Y),
```

```
21          aeroporto(X,Ab),
22          aeroporto(Y,Am),
23          nome(Ab,Nome),
24          nome(Am,Nome).
25
26  mesmo_voo(Vb,Vm,Numero) :-
27          id_voos_base(X),
28          id_voos_mod(Y),
29          voo(X, Vb),
30          voo(Y, Vm),
31          numero_voo(Vb, Numero),
32          numero_voo(Vm, Numero).
33
34  concatenar([],L,L):-!.
35  concatenar([X|M],L,[X|Y]):-concatenar(M,L,Y).
36
37  elim12([],L,L):-!.
38  elim12([X|M],L,S):-elim12(M,T,S).
39
40  eliminaX([],_X,[]):-!.
41  eliminaX([X|M],X,Z):- eliminaX(M,X,Z),!.
42  eliminaX([R|M],X,[R|Z]):- eliminaX(M,X,Z),!.
43
44  eliminaRep([],[]):-!.
45  eliminaRep([X|M],S):-
46          not(lista(X)),
47          eliminaX(M,X,T),
48          eliminaRep(T,Y),
49          concatenar([X],Y,S).
50  eliminaRep([X|M],S):-
51          elim12(X,M,T),
52          eliminaRep(X,Y),
53          eliminaRep(T,J),
54          concatenar([Y],J,S).
55
56  rota_mudou(Nome):-
```

```
57          mesma_rota(A,B,Nome),
58          aeroportos_visitados(A, C),
59          aeroportos_visitados(B, D),
60          visitado(D,NovoPonto),
61          not visitado(C,NovoPonto).
62
63  voo_mudou(X):-
64          mesmo_voo(A,B,Numero),
65          rota_voo(A,RotaA),
66          rota_mudou(RotaA),
67          numero_voo(A, X).
68
69  novo_aeroporto(X) :-
70          id_aeroportos_mod(ID),
71          aeroporto(ID, Y),
72          not mesmo_aeroporto(W, Y, Nome),
73          nome(Y, X).
74
75  nova_rota(NomeRota) :-
76          id_rotas_mod(ID),
77          rota(ID,Y),
78          not mesma_rota(W, Y, Nome),
79          nome_rota(Y,NomeRota).
80
81  novo_voo(X) :-
82          id_voo_mod(ID),
83          voo(ID,Y),
84          not mesmo_voo(W,Y,Numero),
85          numero(Y,X).
86
87  rotas_percorridas(NomeRota):-
88          voos(modificado,X),
89          voo(X,IdVoo),
90          rota_voo(IdVoo,NomeRota),
91          rotas(modificado,IdRotas),
92          rota(IdRotas,IdRota),
```

```
93          nome_rota(IdRota,NomeRota).
94
95  rotas_novas_nao_percorridas(NomeRota) :-
96          nome_rota(X,NomeRota),
97          nova_rota(NomeRota),
98          not rotas_percorridas(NomeRota).
99
100 aeroportos_atendidos(Visitado) :-
101         voos(modificado,X),
102         voo(X,IdVoo),
103         rota_voo(IdVoo,NomeRota),
104         rotas(modificado,T),
105         rota(T,IdRota),
106         nome_rota(IdRota,NomeRota),
107         aeroportos_visitados(IdRota, IdVisitados),
108         visitado(IdVisitados,Visitado).
109
110 lista_aeroportos_atendidos(Lista):-
111         findall(Visitado,aeroportos_atendidos(Visitado),ListaVisitados),
112         eliminaRep(ListaVisitados,Lista).
113
114 lista_aeroportos_alcancaveis(ListaAeroportos) :-
115         findall(Visitado,visitado(_,Visitado),Lista),
116         eliminaRep(Lista,ListaAeroportos).
```

Todos os resultados obtidos após a aplicação de todas as regras de negócio na base de fatos obtida.

Listagem A.5: Resultados da inferência

```
1 ROTA_MUDOU:
2 'rota verde'
3
4 VOO_MUDOU:
5 '001'
6
7 NOVO_AEROPORTO:
8 pampulha
```

```
9  serrinha
10
11 NOVA_ROTA:
12 'rota azul'
13
14 NOVO_VOO:
15
16 ROTAS_PERCORRIDAS:
17 'rota verde'
18 'rota amarela'
19
20 ROTA_NOVA_NAO_PERCORRIDA:
21 'rota azul'
22
23 AEROPORTOS_ATENDIDOS:
24 brasilia
25 guarulhos
26 fiumicino
27 galeao
28 congonhas
29 brasilia
30 confins
31 cumbica
32
33 LISTA_AEROPORTOS_ALCANCAVEIS:
34 [brasilia, guarulhos, fiumicino, galeao, congonhas, confins, cumbica]
35
36 AEROPORTOS_ATENDIDOS:
37 [brasilia, guarulhos, fiumicino, galeao, confins, cumbica, congonhas, serrinha]
```