



# XPerseus: Uma Interface Gráfica para Detecção de Diferenças entre Documentos XML

**Rafael Barros Silva**

JUIZ DE FORA
JULHO, 2011

# XPerseus: Uma Interface Gráfica para Detecção de Diferenças entre Documentos XML

**Rafael Barros Silva**

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

Orientadora: Profa. Alessandreia Marta de Oliveira

JUIZ DE FORA
JULHO, 2011

XPERSEUS: UMA INTERFACE GRÁFICA PARA DETECÇÃO DE DIFERENÇAS ENTRE DOCUMENTOS XML

Rafael Barros Silva

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

Aprovada por:

_____

**Profa. Alessandreia Marta de Oliveira Julio** - orientadora
M. Sc. em Engenharia de Sistemas, COPPE/UFRJ, 2003.

_____

**Prof. Jairo Francisco de Souza**
M. Sc. em Engenharia de Sistemas, COPPE/UFRJ, 2007.

_____

**Prof. Michel Heluey Fortuna**
Doutor em Engenharia de Sistemas, COPPE/UFRJ, 2008.

JUIZ DE FORA, MG – BRASIL
JULHO, 2011

# AGRADECIMENTOS

Agradeço a meus pais pelo amor e apoio incondicional em todos os momentos da minha vida que resultaram na pessoa que sou hoje.

À minha avó Maria Helena, *in memoriam*, que sempre incentivou seus netos para que conquistassem seus objetivos.

À minha namorada Lívia que com amor e paciência me ajudou e esperou nos momentos que estive ausente.

À minha orientadora, Alessandreia, pela confiança depositada no trabalho e pela excelente orientação.

Aos colegas do Grupo de Educação Tutorial da Computação (GetComp - UFJF), em especial, Carolina Cunha, Daniel Menandro, Guilherme Martins, Leonardo Chinelate, Pedro Gazzola e Plínio Garcia que ajudaram nas pesquisas e nas revisões dessa monografia.

Ao amigo Brian pelas dicas para a implementação da ferramenta.

À minha prima, Camila, pela ajuda de sempre.

A Deus por sempre me iluminar nas escolhas da vida.

# RESUMO

Documentos XML estão se tornando um padrão para publicação e transporte de dados via Web, sendo utilizados, por exemplo, para definir configurações e leiautes das aplicações. Na maioria das vezes, os arquivos XML são colocados juntos com arquivos de texto comuns e por terem uma estruturação própria, ou seja, uma linguagem de marcação com elementos e atributos formando as *tags* que são definidas pelo desenvolvedor, necessitam de um tratamento diferenciado no que se refere à mesclagem desses documentos. Diante disso, surge a necessidade de se desenvolver sistemas que controlem as versões desses arquivos através de ferramentas específicas de detecção de diferenças entre documentos XML, realizando um processo de mesclagem mais controlado e menos propício a erros. A meta deste trabalho é, inicialmente, a realização de um levantamento bibliográfico, seguido de uma pesquisa das principais ferramentas de comparação e visualização de diferenças de documentos XML disponíveis na literatura. Como objetivo principal, destaca-se a implementação de um protótipo que possa atender as necessidades iniciais dos desenvolvedores quanto à detecção de diferenças nos documentos XML.

**Palavras-chave:** XML. Dados Semi-Estruturados. Sistemas de Controle de Versão. Detecção e Mesclagem de Documentos

## ABSTRACT

XML documents are becoming a standard for publication and data transport on the Web. They are used, for example, to define configurations and layouts of the applications. Most of the times, XML files are put together with common text files. XML files have a particular structure, in other words, they have a markup language with elements and attributes creating tags that are defined by the developer, so these kind of documents need a different treatment in refers to its merge with common text files. Given that fact, there is a need of developing systems that control these files' versions using specific tools that are able to detect the differences between XML documents and to perform a merge process more controlled and less prone to errors. At first, the goal of this paper is to do a literature review, researching and studying the main tools that compare and mix XML documents, and then to implement a graphical tool that is able to meet developers' needs of detecting in the XML documents differences.



**Keywords:** XML. Semistructured Data. Control Version Systems. Detection and Merge of Documents.

# SUMÁRIO

# LISTA DE FIGURAS

**LISTA DE ABREVIATURAS E SIGLAS**

3DM *Three Way Merging*

API *Application Programming Interface*

ASCII *American Standard Code for Information Interchange*

CVS *Concurrent Version System*

DOM *Document Object Model*

DTD *Document Type Definition*

GCS *Gerência de Configuração de Software*

LCS *Longest Common Subsequence*

SCCS *Source Control System*

SCVs *Sistemas de Controle de Versões*

UFRGS *Universidade Federal do Rio Grande do Sul*

XML *Extensible Markup Language*

W3C *World Wide Web Consortium*

# 1. INTRODUÇÃO

A quantidade de dados disponível eletronicamente tem crescido muito. Porém, nem todos são coletados e inseridos em bancos de dados estruturados cuidadosamente projetados. Grandes empresas gerenciam vários repositórios heterogêneos de dados e muitas aplicações requerem acesso a esta informação, armazenada em diferentes modelos e mecanismos de acesso (SACCOL e HEUSER, 2001).

Para suportar o acesso a este tipo de informação, modelos de dados semi-estruturados têm sido propostos. Estes modelos apresentam uma representação estrutural heterogênea, por este motivo não podem ser identificados como informações bem definidas ou o contrário, modelos não-estruturados. Dados *Web* se enquadram nessa definição: em alguns casos os dados possuem uma descrição uniforme (um catálogo de produtos), em outros, algum padrão estrutural pode ser identificado (um documento XML). Afirma-se também que dados semi-estruturados são aqueles no qual o esquema de representação está presente (de forma explícita ou implícita) juntamente com as informações, ou seja, o conteúdo é auto-descritivo. Isto significa que uma análise deve ser feita para que a sua estrutura possa ser identificada e extraída (ELMASRI e NAVATHE, 2005).

Um problema relacionado é que, assim como as informações armazenadas em bancos de estruturados, os dados semi-estruturados evoluem ao longo do tempo, em função, por exemplo, de alterações de cunho técnico. Estas modificações podem levar os dados semi-estruturados a um estado inconsistente, pois as instâncias podem se tornar incompatíveis com as definições mais recentes dos esquemas.

Contudo, a engenharia de software tem a Gerência de Configuração de Software (GCS) como disciplina que aplica procedimentos técnicos e administrativos para identificar e documentar as características físicas e funcionais de um item de configuração, controlar as alterações nessas características, armazenar e relatar o processamento das modificações e o estágio da implementação e verificar a compatibilidade com os requisitos especificados (IEEE, 1990).

Um dos ramos da GCS é o sistema de controle de versões. Esse sistema permite que os itens de configuração sejam identificados, como estabelecido pela função de identificação da configuração, e que eles evoluam de forma distribuída, concorrente, porém disciplinada. Essa característica é necessária para que as diversas solicitações de

modificação efetuadas na função de controle da configuração possam ser tratadas em paralelo, sem gerar inconsistências no sistema de GCS como um todo. Esse sistema possui algoritmos como, por exemplo, o LCS – *Longest Common Subsequence* - (MYERS, 1986) que realiza a comparação de artefatos e detectam pontos em que esses artefatos foram modificados. O algoritmo LCS é usado por alguns dos sistemas de controle de versões atuais como o CVS – *Concurrent Version System* (VENUGOPALAN, 2002).

## 1.1 MOTIVAÇÃO

O controle de versão aplicado a documentos XML vêm se tornando muito importante nos últimos anos. Isso ocorre porque o uso da linguagem XML tem sido cada vez mais difundido para troca de dados na Web, tornando-se um padrão para publicação e transporte de dados na rede.

Diante disso é necessário ter um tratamento diferenciado nos documentos XML em relação aos sistemas de controle de versão, desenvolvendo algoritmos e ferramentas que torne menos árduo o trabalho dos desenvolvedores e transformem a mesclagem em um processo mais controlado e menos propício a erros.

Realizar um breve levantamento bibliográfico sobre esse tema permite reunir um conjunto de documentos, teorias e resultados de projetos recentes para a implementação de uma nova ferramenta de detecção de diferenças de documentos XML que ajude no trabalho do desenvolvedor diminuindo a quantidade de erros nesses tipos de arquivos.

## 1.2 JUSTIFICATIVA

Os documentos XML são submetidos a sistemas de controle de versão, porém, a maioria desses sistemas não considera o formato específico dos arquivos XML, tratando-os como arquivos comuns de texto. Nesses tipos de documentos, o conceito de linha como unidade de comparação não é o ideal, já que a XML define estruturas próprias que seriam recomendadas para esse fim, como, por exemplo, atributos e elementos. Na literatura existem diversas propostas para a comparação e versionamento de documentos XML

Diante disso é necessário um tratamento especial aos documentos XML em relação aos sistemas de controle de versão, avaliando ferramentas que conduzam a um processo de mesclagem mais preciso e controlado, sem prejudicar o controle dos demais arquivos. Há

vários algoritmos específicos que procuram realizar essa mesclagem. Por exemplo, o X-Diff, um algoritmo específico para XML e que possui implementação livre (WANG, DEWITT e CAI, 2003).

## 1.3 OBJETIVO

Diante do que foi mencionado anteriormente, o objetivo deste trabalho é, em um primeiro momento, realizar um breve levantamento bibliográfico relacionado aos sistemas de controle de versão aplicados a documentos XML, levando em conta o formato estruturado dos documentos ao obter e exibir as diferenças entre versões, e conduzindo um processo de detecção de diferenças mais preciso, sem, no entanto, prejudicar o controle dos demais arquivos. Para auxiliar os desenvolvedores que trabalham no cenário mencionado anteriormente, foi idealizada a criação de uma ferramenta, a XPerseus, que permitirá a comparação entre diferentes arquivos XML gerando, como resultado, um arquivo contendo todas as alterações realizadas nos arquivos XML.

## 1.4 ORGANIZAÇÃO DO TRABALHO

Além desta seção introdutória, o documento é composto por mais quatro capítulos. O Capítulo 2 expõe os conceitos básicos de Gerência de Configuração de Software (GCS) e Sistemas de Controle de Versões (SCVs), apresentando uma visão geral sobre quatro importantes SCVs: Subversion, Mercurial, CVS e Git. Também é apresentado o tema de versionamento XML no qual é explicado como os documentos XML são tratados nos SCVs comuns e como deveriam ser abordados. Por fim são apresentados alguns conceitos a respeito da detecção e mesclagem de documentos XML. O Capítulo 3 apresenta um estudo comparativo de cinco ferramentas de detecção e/ou mesclagem de documentos e seus principais algoritmos. A partir desse estudo é construído um quadro com as principais funcionalidades e vantagens de cada ferramenta.

O Capítulo 4 apresenta a ferramenta XPerseus assim como suas funcionalidades, vantagens e desvantagens além de um exemplo de utilização através dos *screenshots*.

Finalmente, o Capítulo 5 traz as considerações finais do trabalho ressaltando as contribuições e propondo novos trabalhos.

# 2. GERÊNCIA DE CONFIGURAÇÃO

## 2.1 GERÊNCIA DE CONFIGURAÇÃO DE SOFTWARE

A Gerência de Configuração surgiu nos anos 50 com o objetivo de controlar as modificações na documentação referente à produção de aviões de guerra e naves espaciais, dada à necessidade da indústria na época da guerra. Mais tarde, passou a abranger artefatos de software, desencadeando o surgimento da Gerência de Configuração de Software (GCS) (MURTA, 2006).

A história da GCS tornou-se visível em meados da década de 60 e 70, quando os microprocessadores se tornaram populares e o software deixou de ser considerado parte integrante do hardware para se tornar um produto independente. Porém, somente a partir dos anos 80 houve uma mudança quanto ao foco da GCS, passando a ser aplicada em organizações não militares. Nesta época, os sistemas se tornaram cada vez maiores e sofisticados, ficando claro que seriam necessárias metodologias próprias, diferentes das usadas no desenvolvimento de hardware, para controlar o desenvolvimento desses sistemas.

A GCS tem como finalidade controlar a evolução dos sistemas de software, utilizando ferramentas e métodos, que visam aumentar a produtividade e reduzir os erros cometidos durante o processo de construção do software.

Segundo Presman (1995), Gerência de Configuração de Software é o conjunto de atividades projetadas para controlar as mudanças pela identificação dos produtos do trabalho que serão alterados, estabelecendo um relacionamento entre eles, definindo o mecanismo para o gerenciamento de diferentes versões destes produtos, controlando as mudanças impostas e relatando as mudanças realizadas.

A utilização de ferramentas e métodos é de fundamental importância no controle sobre os artefatos produzidos e modificados por diferentes recursos, desde o planejamento e levantamento de requisitos, até a construção e entrega do produto. O motivo da sua importância está geralmente associado aos problemas identificados quando a GC não é utilizada no desenvolvimento de software. Tais problemas consistem de: perda de código, impossibilidade de determinar o que aconteceu a um programa ou parte dele após um erro, impossibilidade de determinar quem, porque e quando foram efetuadas modificações, dentre outros problemas.

A seguir são analisados dois destes problemas. Suponha que uma empresa não tenha conhecimento do que seja Gerência de Configuração de Software e que em um determinado projeto, um desenvolvedor esteja modificando os artefatos C1, C2 e C3 em um diretório compartilhado na rede. Paralelamente, um segundo desenvolvedor modifica os artefatos C4, C5 e também o artefato C3, como mostrado na Figura 2.1.

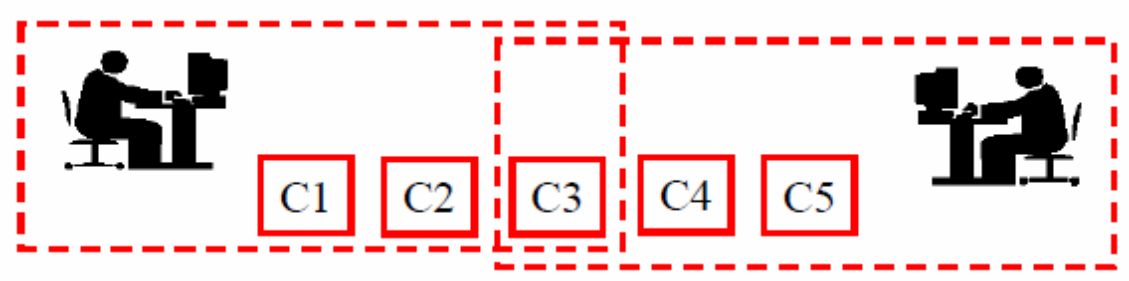


**Figura 2.1 – Espaço de trabalho compartilhado por vários desenvolvedores (MURTA, 2006)**

Nessa situação, o segundo desenvolvedor não notifica o primeiro desenvolvedor sobre o impacto que a modificação do artefato C3 pode causar no código. Consequentemente, o primeiro desenvolvedor, que está usando o mesmo espaço de trabalho, não conseguirá identificar, de forma rápida, o motivo que levou sua implementação a falhar. Este problema acontece pela falta de notificação e pelo compartilhamento de artefatos de software por diversos desenvolvedores.

Suponha que dois desenvolvedores decidiram centralizar os artefatos em um repositório e que cada um realiza suas modificações em sua própria área de trabalho. Após cada modificação, o artefato seria devolvido ao repositório. Analisando essa situação, frequentemente aconteceriam sobreposições ou perdas de modificações realizadas nos artefatos comuns dos projetos de empresas sem a prática da Gerência de Configuração de Software. Um desenvolvedor poderia implementar sua modificação em uma versão desatualizada do artefato e sobrepor a versão mais atual disponibilizada por outro. Este problema ocorre devido a modificações realizadas ao mesmo tempo, quando dois desenvolvedores compartilham o mesmo repositório e não existe uma política de controle ou restrição quanto ao acesso a este repositório (ver Figura 2.2).

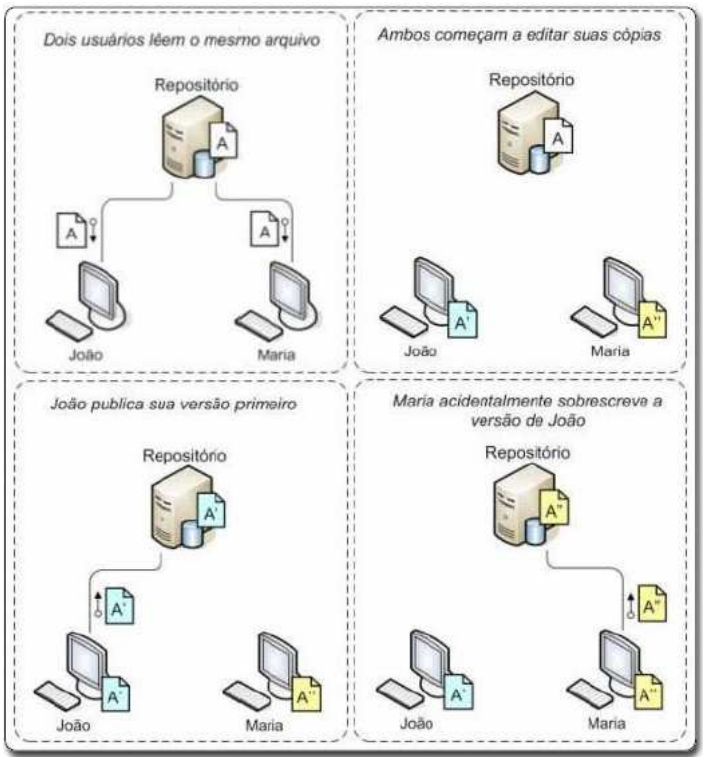


**Figura 2.2 – Repositório centralizado compartilhado por vários desenvolvedores (MURTA, 2006)**

A GCS pode ser tratada sob diferentes perspectivas em função do papel exercido pelo participante do processo de desenvolvimento de software. Na perspectiva gerencial, a GCS é dividida em cinco funções, que são (IEEE, 2005): identificação da configuração, controle da configuração, contabilização da situação da configuração, avaliação e revisão da configuração e gerenciamento de liberação e entrega.

Já na perspectiva de desenvolvimento, a Gerência de Configuração de Software abrange três sistemas principais: controle de versões, controle de modificações e controle de gerenciamento de construção.

O sistema de controle de versões (SCV) é a base de toda a Gerência de Configuração, apoiando as atividades de controle de mudança e controle de gerenciamento de construção. Fornece os seguintes serviços: (1) identificação, armazenamento e gerenciamento dos itens de configuração e de suas versões ao longo de todo o ciclo de vida do software; (2) mantém um histórico de todas as alterações efetuadas nos itens de configuração ao longo do desenvolvimento do projeto; (3) permite a criação de ramificações e rótulos no projeto e (4) recupera uma configuração em um determinado

instante de tempo. Como exemplos de ferramentas de controle de versões têm-se: o Subversion, Git, Mercurial e CVS.

O sistema de controle de modificações fornece um serviço complementar ao oferecido pelo SCV. O foco desse tipo de ferramenta é no armazenamento de todas as informações geradas durante o andamento das solicitações de modificação e no relato dessas informações aos participantes interessados e autorizados. Como exemplos de ferramentas de controle de modificações: o Trac, Bugzilla e Mantis.

O sistema de gerenciamento de construção tem por finalidade automatizar o processo de transformação dos diversos artefatos do software que compõem um projeto em um sistema executável propriamente dito. Por exemplo, o processo de teste e empacotamento de uma aplicação java como um arquivo ".jar". Pode-se citar como exemplos de ferramentas de gerenciamento de construção o Ant, Make e Maven.

Os benefícios gerados pela utilização da Gerência de Configuração de Software são vários. Dentre eles: um aumento na produtividade e eficiência do projeto; possibilidade de informar quando, por que e por quem um artefato foi alterado; facilidade na geração de versões diferentes de um mesmo produto de software; documentação sobre a evolução do sistema; desenvolvimento dependente do processo e não das pessoas; aumento da memória organizacional da empresa.

## 2.2 SISTEMAS DE CONTROLE DE VERSÕES

Em meio ao processo de desenvolvimento de software, diversas pessoas participam da produção de documentos sobre o projeto e da geração de código. Esta característica exige que se tenha certo controle das alterações feitas nestes artefatos, a fim de evitar que uma pessoa ou equipe sobrescreva erroneamente o que foi feito por outra. Diante da necessidade de centralizar e organizar o desenvolvimento de software em equipe surgiram nas décadas de 70 e 80, os SCVs. Um dos principais objetivos destes sistemas é gerenciar arquivos, que são normalmente de documentação ou de código, mantendo, entre outras coisas, um relatório sobre as alterações que foram feitas em cada um destes arquivos, bem como por quem, a que hora e em que dia.

Atualmente, estão disponíveis vários SCVs sendo, muitos deles, de licença *opensource*. Um dos mais conhecidos é o Subversion cujo desenvolvimento iniciou-se em 2000 com a proposta de melhorar as funcionalidades do CVS *(Concurrent Versions*

*System*). Esse último teve seu desenvolvimento iniciado no fim da década de 80 e vem perdendo espaço devido a algumas limitações como, por exemplo, problemas ao mover e renomear arquivos.

A maioria dos SCV é composta por um repositório e uma área de trabalho. O repositório armazena todos os arquivos e, também, todo o histórico de evolução do projeto, registrando todas as alterações realizadas em qualquer item versionado. O desenvolvedor utiliza de uma área ou cópia de trabalho que contém a cópia dos arquivos do projeto e que é monitorada para identificar todas as mudanças ocorridas. Essa área é individual e separada das demais áreas de trabalho. A sincronização entre a área de trabalho e o repositório é feita através dos comandos `commit` e `update`.

O `commit` envia um pacote contendo uma ou mais modificações feitas na área de trabalho (origem) ao repositório (destino). O `update` faz o inverso, ou seja, envia as modificações contidas no repositório (origem) para a área de trabalho (destino).

Cada `commit` gera uma nova revisão no repositório, contendo as modificações feitas, data e autor. Essa revisão é uma versão de todos os dados em um dado instante de tempo. As versões podem ser recuperadas e analisadas quando for necessário. O conjunto dessas revisões forma o histórico do projeto.

Existem dois tipos de sistemas de controle de versões: centralizado (Subversion e CVS, por exemplo) e distribuído (Mercurial e Git). Tanto o controle de versões centralizado quanto o distribuído possuem repositórios e áreas de trabalho. A diferença está em como os dados são manipulados.

No controle de versões centralizado há apenas um repositório central, mas várias áreas de trabalho, sendo uma para cada desenvolvedor. A comunicação entre uma área de trabalho e outra passa obrigatoriamente pelo repositório central. Já no controle de versões distribuído, são vários repositórios autônomos e independentes, um para cada desenvolvedor. Cada repositório possui uma área de trabalho acoplada e a comunicação acontece, primeiramente, entre os dois, utilizando das operações de *commit* e *update*. Um repositório pode se comunicar com outro através das operações de *pull* e *push*. O comando *pull* atualiza o repositório local (destino) com todas as alterações realizadas em outro repositório (origem) e o comando *push* envia as alterações do repositório local (origem) para outro repositório (destino).

Os SCVs são de fundamental importância para controlar o histórico de um projeto, desfazendo, analisando e recuperando versões sem erros ou sem alguma funcionalidade;

bem como para o desenvolvimento de software em equipe, permitindo que diversas pessoas trabalhem em paralelo sobre o mesmo conjunto de documentos e minimizando o desgaste provocado por problemas de conflitos de edições em arquivos.

Além disso, alguns softwares para controle de versões possuem uma característica interessante, que é a de permitir a criação de ramos (*branches*). Esta funcionalidade é utilizada quando se deseja criar versões diferentes de um software, mas que contenham a mesma base. Um exemplo de uso desta funcionalidade é o desenvolvimento de uma versão teste e de uma comercial do software. Ambas possuem a mesma base, mas enquanto a versão comercial disponibiliza todas as funções disponíveis no software, a versão teste inclui apenas algumas.

Apesar de trazer todos estes benefícios para o processo de desenvolvimento de software, o emprego de sistemas para controle de versões ainda apresenta algumas limitações em seu uso. Por exemplo, a maioria possui dificuldade de gerenciar arquivos binários, tais como executáveis, diagramas de classe, vídeos e imagens. Uma outra limitação, é o fato de que os sistemas de controle de versões, em geral, utilizam linhas como unidades de comparação para qualquer arquivo texto. Para arquivos com estruturação própria, como os dados semi-estruturados, esse tipo de comparação passa a gerar documentos mal formados, podendo assim comprometer todo um projeto.

Concluindo, para tentar minimizar as limitações dos sistemas é importante manter a comunicação entre os desenvolvedores da equipe, já que qualquer inconsistência no código pode tornar o projeto inoperante.

## 2.3 EXEMPLOS DE SISTEMAS DE CONTROLE DE VERSÕES

Atualmente, existem no mercado inúmeras ferramentas de controle de versões. São apresentadas descrições de quatro delas: CVS, Subversion, Mercurial e Git.

O CVS (*Concurrent Version System*) é uma ferramenta de código aberto que permite gerenciar diversos arquivos organizados em um diretório, permitindo que estes sejam acessados concorrentemente. Quando dois arquivos são alterados simultaneamente, por usuários diferentes, o CVS faz uma comparação entre eles e procura por eventuais conflitos. Se encontrar algum, o CVS permite que o usuário indique quais modificações devem ser armazenadas. Caso contrário, ele aceita apenas alterações feitas na versão mais atualizada do arquivo.

A cada operação de atualização da cópia do repositório, o CVS incrementa a versão do arquivo. Ele também adiciona dados a um histórico que contém o nome do usuário, a data e hora da modificação e também uma breve descrição do que foi alterado. O CVS permite ainda o desenvolvimento de ramos (*branches*). Desta forma, é possível elaborar, em paralelo, duas ou mais versões diferentes de um mesmo software.

Atualmente, o projeto CVS é mantido por voluntários. Entretanto, devido a algumas limitações, tais como problemas para renomear ou mover arquivos, dificuldade em resolver conflitos e o versionamento limitado a arquivos, o CVS perdeu espaço. Porém, ainda existem diversas distribuições destes sistemas de controle de versão, tais como, TortoiseCVS e Eclipse (com plug-in do CVS). Diversas outras ferramentas foram implementadas com o propósito de substituí-lo, sendo o Subversion uma das mais importantes.

O Subversion é um SCV similar ao CVS. O projeto iniciou em 2000 com a idéia de se construir um CVS melhor, isto é, mantendo o mesmo modelo de trabalho, mas consertando as falhas e limitações que o CVS apresentava. Desde então, vem atraindo uma comunidade cada vez maior de colaboradores e usuários.

O Subversion ultrapassou o uso do CVS para gerenciar o desenvolvimento de projetos *open-source*, tais como o Apache, KDE, GNOME e Samba. Gratuito e de código aberto, ele soluciona diversos problemas que antes eram comuns em outros SCVs. Devido ao mapeamento da estrutura do repositório, por exemplo, é possível renomear e mover arquivos normalmente, mantendo um histórico das ações realizadas. Além disso, ele possui mecanismos que evitam que os arquivos ou diretórios sejam corrompidos durante uma operação de *commit* (atualização no repositório) e suporta o armazenamento de arquivos binários. Por último, ele inclui todas as funcionalidades providas pelo CVS através de uma interface mais simples, facilitando a criação, edição e remoção de arquivos e diretórios.

O Git é um SCV distribuído que teve seu desenvolvimento iniciado em 2005 por Linus Torvalds. O desenvolvimento dessa ferramenta surgiu após vários desenvolvedores do kernel Linux decidirem desistir de utilizar o sistema BitKeeper que possuía a licença gratuita. Isso ocorreu após Larry McVoy, detentor dos direitos autorais, tornar a licença proprietária. Além do desenvolvimento distribuído, o Git possui como características a compatibilidade com vários protocolos de rede, o suporte consistente para desenvolvimentos não-lineares, ou seja, suporta rápidas criações de ramos e realizações de mesclagens (*merge*), e uma eficiente manipulação de grandes projetos devido ao bom

desempenho da ferramenta. Atualmente, diversos projetos importantes utilizam o Git como SCV como, por exemplo, o Eclipse, Fedora e Android.

O Mercurial também é uma ferramenta de controle de versão distribuída. Teve seu desenvolvimento iniciado por volta do ano 2005 pelos mesmos motivos que o Git. Matt Mackall foi o criador e líder de desenvolvimento do Mercurial. Uma das grandes vantagens do Mercurial é a facilidade de aprendizagem que a ferramenta proporciona. O Mercurial é uma ferramenta poderosa, mas simples e mais recomendada a usuários iniciantes do que o Git. Outra grande vantagem do Mercurial é o fato dele ser compatível tanto com o Subversion quanto com o GIT. Isso faz com que o desenvolvedor que utiliza o Mercurial consiga realizar operações como sincronização de repositórios, clone, check out, trabalhar com ramos e outras, tanto no GIT quanto no Subversion.

Um estudo mais aprofundado sobre essas ferramentas pode ser encontrado em (MENANDRO, 2010) e (CEDERQVIST, 1993).

## 2.4 VERSIONAMENTO XML

Nos projetos atuais, documentos na linguagem XML (*eXtensible Markup Language*) são muito utilizados, tanto para definir configurações ou o leiaute da aplicação. Esses projetos normalmente são submetidos a algum SCV, sendo que os arquivos XML são também versionados juntamente com os demais arquivos do projeto.

XML é uma linguagem de marcação para descrever informações. Foi proposto pelo W3C (*World Wide Web Consortium*) em meados de 1990, como um padrão para representação de dados pela *Web*. Essa linguagem facilita a construção de informações mais precisas e possui uma estrutura capaz de ser compreendida por várias linguagens de programação, gerando uma integração maior entre sistemas e linguagens. O XML possui como vantagens o fato de ser um padrão aberto, simples, legível; ter uma interligação com banco de dados distintos; além de ser extensível e flexível, isto é, pode representar qualquer tipo de informação e de qualquer forma. Não possui *tags* pré-definidas, quem as define é o desenvolvedor (BRAY et al., 2008).

Os SCVs, em geral, utilizam linhas como unidades de comparação para qualquer arquivo texto. As unidades de comparação representam os elementos atômicos usados para comparação e identificação de conflitos, ou seja, a comparação ocorre linha por linha. Esse comportamento se mostrou bastante adequado para arquivos simples de texto puro

(MURTA, 2006). Contudo, quando o artefato que está sendo manipulado contém um formato específico, esse tipo de algoritmo passa a gerar resultados não desejados. Isso acontece, por exemplo, nos documentos XML onde sua estrutura é formada de elementos e atributos formando as chamadas *tags.*

Por exemplo, suponha que dois desenvolvedores, trabalhando em paralelo, alterem o elemento "endereço" do fragmento a seguir (Figura 2.3).

```
<endereco
    id = "idEndereco"
    logradouro = "nomeRua"
    bairro = "nomeBairro">
```

**Figura 2.3 – Fragmento de um arquivo XML**

Os dois desenvolvedores inseriram o mesmo atributo em um determinado elemento, mas em linhas diferentes, já que o elemento é formado por várias linhas (Figura 2.4 e Figura 2.5).

```
<endereco
    id = "idEndereco"
    localizacao = "#{coordenadasCidade.mostraCoordenadas}"
    logradouro = "nomeRua"
    bairro = "nomeBairro">
```

**Figura 2.4 – Fragmento de um arquivo XML após alteração de um desenvolvedor**

```
<endereco
    id = "idEndereco"
    logradouro = "nomeRua"
    bairro = "nomeBairro"
    localizacao = "#{coordCidadeEstado.exibirCoordLocalizacao}">
```

**Figura 2.5 – Fragmento de XML após alteração em paralelo**

A maioria dos SCVs realizaria a mesclagem do documento sem detectar nenhum problema, resultando no documento da Figura 2.6. Porém, o documento está mal-formado, pois o elemento possui agora dois atributos com o mesmo nome, algo não permitido na linguagem XML. No caso de arquivos XML, seriam mais adequadas outras unidades de comparação e versão, como elementos e atributos, ao invés de linhas.

```
<endereco
    id = "idEndereco"
    localizacao = "#{coordenadasCidade.mostraCoordenadas}"
    logradouro = "nomeRua"
    bairro = "nomeBairro"
    localizacao = "#{coordCidadeEstado.exibirCoordLocalizacao}">
```

**Figura 2.6 – Resultado do merge executado por um SCV convencional**

Diante disso é necessário um tratamento especial aos documentos XML por parte dos SCVs, avaliando ferramentas que conduzam a um processo de mesclagem mais preciso e controlado, sem prejudicar o controle dos demais arquivos. Há vários algoritmos específicos que procuram realizar essa mesclagem. Por exemplo, o X-Diff, um algoritmo específico para XML e que possui implementação livre (WANG, DEWITT e CAI, 2003). Na literatura também podem ser encontradas propostas para a comparação e o versionamento de documentos XML, por exemplo, em Cobena et al. (2002).

## 2.5 ALGORITMOS DE DETECÇÃO E MESCLAGEM

Nesta seção são apresentados os conceitos de detecção de diferenças, mesclagem de documentos e alguns dos principais algoritmos que realizam essas ações.

### 2.5.1 DETECÇÃO DE ALTERAÇÕES

Nos projetos atuais é comum ter duas ou mais versões de um mesmo documento XML. Diante disso é necessária uma forma de detectar exatamente quais são as diferenças entre esses arquivos, isto é, saber quais foram as mudanças de um arquivo para outro. Para isso, existem algoritmos e ferramentas capacitadas. Um exemplo é o X-Diff, um algoritmo de detecção de diferenças específico para documentos XML que apresenta complexidade da ordem $O(n^2)$ e utiliza um conjunto básico de operações de transformação para construção dos deltas. Esses algoritmos, normalmente, recebem como entrada dois documentos e geram como saída uma seqüência finita de operações de modificação que identificam como transformar o primeiro documento no segundo. Essa seqüência de operações é chamada de delta.

Na representação desses deltas, para arquivos XML, diferentes aspectos ganham ou perdem importância de acordo com o objetivo escolhido. Se o objetivo for gerenciar

versões de arquivos, o delta deve permitir a perfeita reconstrução de um arquivo a partir do outro, de forma eficaz e eficiente. Caso o objetivo seja descrever o histórico de certos elementos ao longo do tempo, é necessário que a representação permita identificar unicamente os elementos do arquivo, bem como perceber quando eles são movidos de um ponto a outro no arquivo, para permitir que se acompanhe todo o tempo de vida do elemento ao longo de diversas alterações. As operações representadas pelos deltas são: inserção, deleção, atualização, cópia e movimentação.

Os algoritmos de detecção de diferenças podem apresentar variações quanto ao modelo de árvore XML utilizado. No modelo não-ordenado, apenas a relação existente entre um elemento e seu ancestral é relevante para fins de identificação de diferenças. Já no modelo ordenado, tanto a relação entre um elemento e seu ancestral quanto o posicionamento de um elemento em relação aos seus irmãos podem ser utilizados na detecção de diferenças. Logo, neste modelo é possível identificar como diferenças as trocas de posição entre dois descendentes de um mesmo ancestral.

Outro exemplo de algoritmo de detecção de diferenças em documentos XML é o XyDiff. Por ser voltado a grandes quantidades de dados, os autores do algoritmo evidenciam como maior objetivo o alto desempenho do algoritmo e aceitam alguma perda de operações na geração do delta entre dois documentos quaisquer.

O XyDiff possui complexidade de tempo de ordem O(n log n) e suporta as operações básicas de diferenciação e a operação de movimentação de elementos. O algoritmo possui características próprias do formato XML para sua operação, como a utilização de identificadores únicos de elementos na diferenciação entre documentos e o modelo ordenado de árvore XML.

O XML TreeDiff, definido em (EPSTEIN; CURBERA, 1998), é outro exemplo de algoritmo de diferenciação. Utiliza APIs DOM (Document Object Model), o que impede de processar documentos de grande porte. Este algoritmo utiliza árvores ordenadas, um conjunto básico de operações para identificar as transformações realizadas em documentos e possui complexidade de ordem pelo menos $O(n^2)$.

Cada algoritmo foca em determinados aspectos da detecção de diferenças e possui propriedades, vantagens, otimizações e requisitos peculiares. Por possuírem propósitos similares, alguns algoritmos têm uma conexão forte com outros, sendo concebidos a partir de melhorias ou adaptações sobre os anteriores.

No contexto desta dissertação, os algoritmos que não levam em consideração as árvores ordenadas serão descartados.

### 2.5.2 MESCLAGEM DE DOCUMENTOS

A mesclagem (ou *merge* em inglês) consiste na fusão automática de versões através da comparação entre elas, quando há um conflito em documentos. Quando dois ou mais desenvolvedores de um SCV obtêm a mesma versão de um determinado arquivo, e executam modificações em paralelo, é necessário consolidar as modificações realizadas. Se o sistema não for capaz de resolver o conflito automaticamente, a mesclagem pode ser feita manualmente. Na maioria das vezes, o SCV mostra o conflito na tela e o desenvolvedor escolhe qual versão deve manter. Quando isso acontece é provável que tenha ocorrido uma falha na comunicação entre os desenvolvedores gerando uma alteração semelhante no mesmo trecho de código do projeto.

### 2.5.3 ALGORITMOS DE DETECÇÃO E MESCLAGEM DE DOCUMENTOS

Para a escolha de um algoritmo eficiente de detecção de diferenças é necessário um estudo comparativo entre os principais algoritmos de detecção em árvores existentes. Essa comparação leva em conta o estudo comparativo realizado em Peters (2005). A seguir são apresentados e explicados os itens utilizados na comparação dos algoritmos:

- **Árvores ordenadas e não-ordenadas:** as árvores são divididas em ordenadas (quando a ordem dos nodos é importante) e as não-ordenadas (quando a ordem dos nodos não é relevante). A maioria das aplicações não exige um tratamento para árvores ordenadas, porém em alguns algoritmos como, por exemplo, o DeltaXML (MONSELL e FONTAINE, 2003) processam os dois tipos de árvores.
- **Memória:** Com o objetivo de reduzir o uso de memória, algoritmos como o X-Diff (WANG, 2003) optam por remover nodos ou subárvores inteiras quando identificam um "casamento" completo, ou seja, porções das árvores que não precisam mais de comparação (útil quando se lida com grandes volumes de dados). Algoritmos como o XyDiff (COBÉNA, 2002) também adotam práticas focando na otimização de uso de memória e tempo de processamento.

- **Complexidade:** Como existe mais de um algoritmo para resolver problemas similares, a análise de complexidade computacional é fundamental. O fato de um algoritmo resolver um dado problema não significa que seja aceitável na prática, a exemplo de quando se lida com grandes volumes de dados.

A Tabela 2.1 reúne um resumo dos itens estudados entre os principais algoritmos. As operações básicas compreendem *insert*, *delete* e *update*.

**TABELA 2.1** – Algoritmos para detecção de diferenças em árvores XML (adaptada de PETERS, 2005)

| Algoritmo | Árvore | Memória | Complexidade | Operações |
|---|---|---|---|---|
| 3dm | Ordenada | - | O(n) | Básicas e move |
| XyDiff | Ordenada | Linear | O(n log n) | Básicas e move |
| DiffXML | Ordenada | Linear | $O(ne+e^2)$ | Básicas e move |
| X-Diff | Não-ordenada | Quadrática | $O(n^2)$ | Básicas |
| DeltaXML | Ambas | Linear | - | Básicas |
| BioDiff | Não – ordenada | Quadrática | $O(n^2)$ | Básicas |

## 2.6 CONCLUSÃO

Qualquer projeto de desenvolvimento de software, seja ele de pequeno, médio ou grande porte, gera uma grande quantidade de informações e artefatos, sendo estas acessadas e modificadas diversas vezes ao longo de todo o ciclo de vida deste projeto. O objetivo do processo de Gerência de Configuração de Software é prover um ambiente controlado e estável, onde as modificações estejam sendo gerenciadas. Garantir a qualidade de seus produtos é um dos objetivos de toda a empresa de desenvolvimento de software. E o processo de gerência de configuração de software colabora significativamente para isso, garantindo integridade, facilidade de manutenção e rastreabilidade dos artefatos durante todo o ciclo de vida de um software.

O uso de SCVs tem se tornado comum em muitas empresas devido ao baixo custo de implantação, ao grande número de ferramentas como, por exemplo, o Subversion e o

Mercurial; e a vasta documentação disponível permitindo que os projetos tenham um histórico de suas alterações e o controle de todos os arquivos.

Porém, como já mencionado, a maioria dos SCVs possuem algumas limitações quanto a comparação e mesclagem de documentos com estruturação própria, como é o caso dos arquivos XML, gerando resultados não desejados após o processo de mesclagem. Para isso, existem várias ferramentas e algoritmos que permitem a comparação, detecção e mesclagem desses tipos de arquivos, tornando o processo de mesclagem mais preciso, controlado e menos susceptível a erros.

No próximo capítulo traz um estudo e uma comparação entre algumas ferramentas responsáveis pela detecção e mesclagem de documentos XML.

# 3. FERRAMENTAS DE CONTROLE DE VERSÕES DE DADOS SEMI-ESTRUTURADOS

Neste capitulo são apresentadas as principais ferramentas de detecção e mesclagem de arquivos XML, onde são discutidas as suas principais características e funcionalidades. Após, é apresentado um quadro comparativo dessas ferramentas.

## 3.1 DESCRIÇÃO DAS FERRAMENTAS

Esta seção tem como objetivo apresentar as principais ferramentas de controle de versões para dados semi-estruturados visando o reconhecimento das principais funcionalidades, vantagens e desvantagens do uso de cada ferramenta. E diante disso, ter material suficiente para analisar e planejar a implementação de uma nova ferramenta.

### 3.1.1 DIFFDOG

O DiffDog (ALTOVA, 2011) é uma ferramenta de visualização de diferenças de interface simples e que permite comparar e mesclar qualquer tipo de arquivo baseado em texto, suportando formatos como Unicode, ASCII e outros, além de comparar diretórios, banco de dados e, principalmente, documentos XML. A ferramenta possui uma versão de testes com funcionalidades reduzidas e uma comercial, que possui todas as funcionalidades completas e algumas outras opções. O DiffDog pode ser executado apenas no sistema operacional Windows e pode ser integrado ao Windows Explorer, de modo que dois arquivos selecionados podem ser comparados diretamente no menu do Explorer.

O algoritmo de detecção de diferenças é um algoritmo proprietário, que acompanha a ferramenta. O funcionamento da ferramenta para comparações entre documentos XML é realizado da seguinte forma: dois arquivos selecionados pelo usuário são abertos lado a lado e as diferenças entre os dois são realçadas através de cores e também indicadas com linhas conectoras, um tipo de ligação visual entre as diferentes versões de documentos XML, de modo que se possa visualizar mais rapidamente as diferenças entre as duas. Esta funcionalidade pode se tornar incompreensível quando existirem muitas diferenças entre os dois arquivos. Outra funcionalidade que deve ser destacada é a busca com opções de

filtros, que permite aos usuários procurarem somente as diferenças que os interessam, ignorando previamente algumas opções as quais sejam desnecessárias para a comparação.

O DiffDog não disponibiliza as funções de navegação entre as diferenças e de edição de documento, mas disponibiliza a função de visualização de casamentos.

Por essa ferramenta ser comercial, ela foi avaliada e testada a partir de uma versão *trial.* Porém não foi possível ter acesso ao código fonte da mesma.

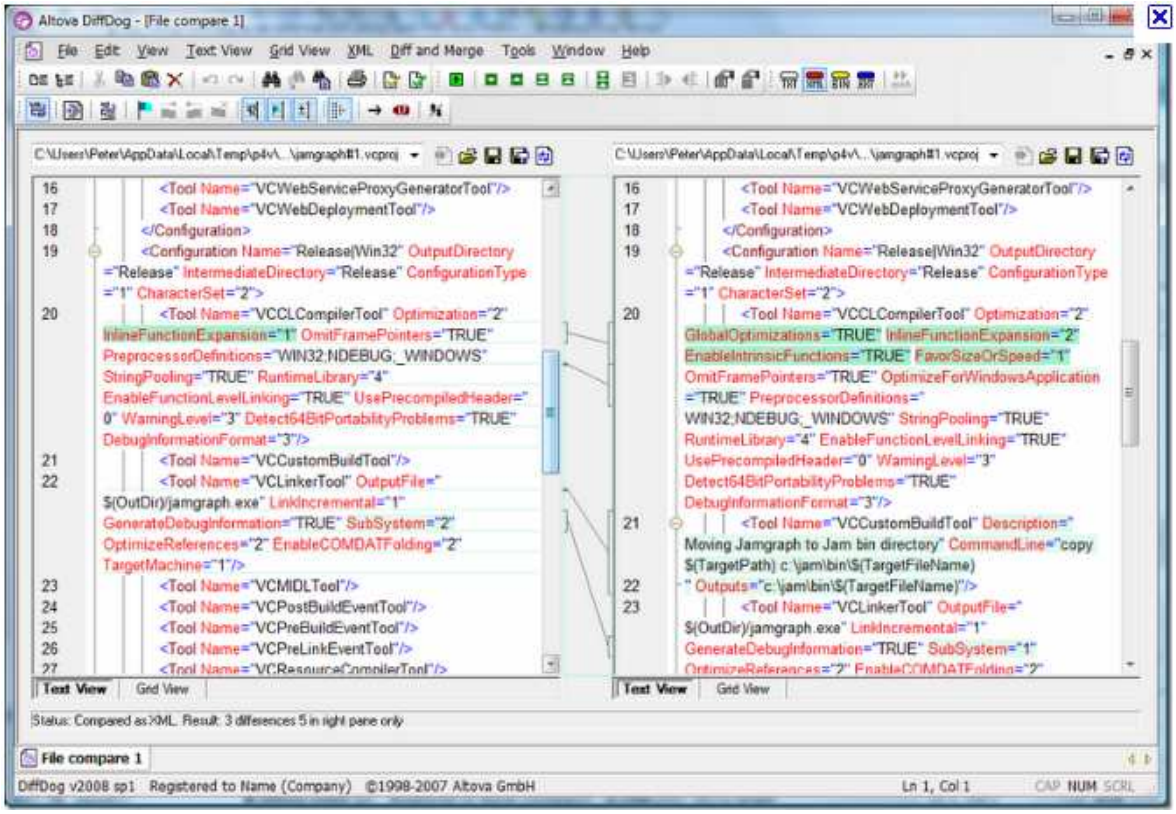


**FIGURA 3.1 – COMPARAÇÃO DE DOCUMENTOS XML**

### 3.1.2 VISUAL X

O VisualX (CECCHIN e HARA, 2009) é uma ferramenta *desktop* desenvolvida para auxiliar os administradores de repositório de dados XML e usuários interessados nas diferenças entre dois documentos XML com o objetivo de identificar com maior facilidade as modificações efetuadas entre versões distintas de um documento. Essa ferramenta foi implementada utilizando a linguagem Java, possui distribuição acadêmica e, além disso, pode ser executado nos sistemas Windows e Linux.

O funcionamento da ferramenta ocorre da seguinte forma: ela recebe como entrada dois arquivos XML escolhidos pelo usuário, que são exibidos lado a lado na tela. Na

aplicação da detecção de diferenças, diferente de outras ferramentas, o VisualX permite ao usuário escolher o algoritmo de diferenças a ser executado, dentre três opções: XyDiff, JXyDiff e XKeyMatch, cada qual com sua peculiaridade.

O XyDiff é um algoritmo para detecção de diferenças em documentos XML voltado para grande quantidades de dados. O JXyDiff é baseado no anterior, e as principais diferenças são a maior portabilidade, por ser escrito totalmente em Java e a apresentação de um delta reduzido. Já o XKeyMatch adiciona um critério semântico à detecção de diferenças.

Após a execução de um destes algoritmos, as diferenças encontradas são mostradas na tela, destacando os elementos XML que foram alterados, quer seja por exclusão, inserção, atualização ou movimentação. Contudo, a ferramenta não apresenta a funcionalidade de visualização dos casamentos. O VisualX permite a navegação entre as diferenças apresentadas, e ao fim, permite ao usuário salvar ou não o delta gerado. Além disso, vale ressaltar que o VisualX oferece recursos como a edição de documentos abertos, o que permite ajustes nos documentos sem a necessidade de abri-los em outros editores; e também esquemas de configurações de cores e dicas de contexto.

Também não foi possível testar e avaliar a VisualX, pois a ferramenta não foi encontrada para *download*.

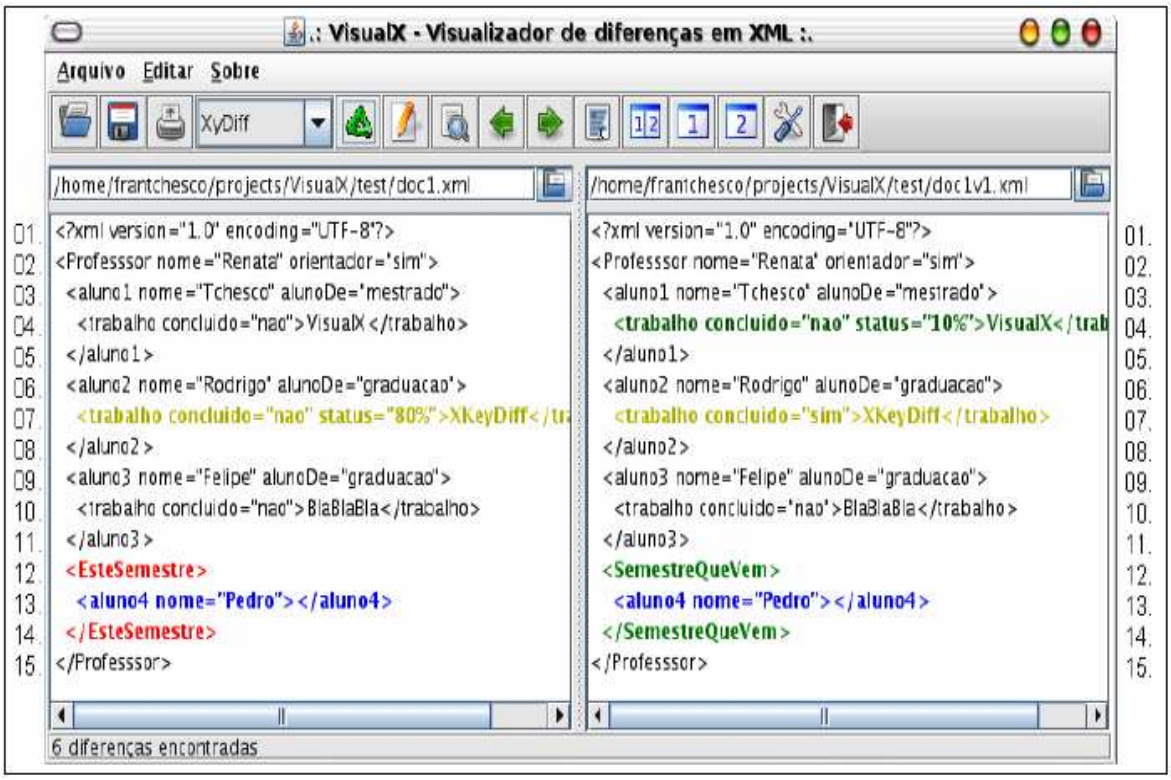


**FIGURA 3.2 – DETECÇÃO DE DIFERENÇAS ENTRE DOCUMENTOS XML**

### 3.1.3 XMLTREEMERGE

O XMLTreeMerge (OLIVEIRA et al., 2011) é uma ferramenta de detecção e comparação de diferenças e mesclagem de documentos XML que foi implementada em um trabalho de conclusão de curso em 2007 na UFRJ, com o objetivo de se estabelecer uma forma mais adequada para controlar as versões de documentos XML, considerando tanto a forma estrutural de exibição das diferenças entre as versões quanto um processo de mesclagem mais preciso e controlado; o que seria uma alternativa às ferramentas já existentes, que possuem algumas ineficiências e inconsistências.

Basicamente, a ferramenta obedece a quatro requisitos: utilização de um algoritmo para detecção de diferenças que seja específico para o formato de documentos XML, a mesclagem não deve produzir documentos mal-formados, possibilitar o acompanhamento do usuário durante o processo de mesclagem e tratar os casos nos quais os arquivos envolvidos não sejam de formato XML.

O algoritmo utilizado na implementação dessa ferramenta foi o 3dm (*3-Way Merging*) que apresenta como vantagens o fato de ser específico para XML, ser um algoritmo capaz de detectar movimentações de nós que tiveram filhos alterados, apresentar cinco tipos de operações (inserção, remoção, atualização, movimentação e cópia) e, também, por possuir uma implementação livre e disponível. Após a execução do algoritmo de diferenças, a ferramenta utiliza as versões do documento e os *logs* gerados pelo algoritmo para apresentar as diferenças para o usuário. As diferenças são apresentadas a partir da visualização de três árvores, uma com a versão base e as outras com duas versões modificadas. Além disso, outra opção fundamental da ferramenta é permitir ao usuário interagir com o documento final, podendo editá-lo como quiser, bem como permitir ao usuário o tratamento dos conflitos.

Vale ressaltar que a ferramenta foi implementada na linguagem Java o que torna o uso da mesma irrestrita aos sistemas operacionais. Essa ferramenta foi disponibilizada para download e com isso testada e avaliada para a realização do quadro comparativo.

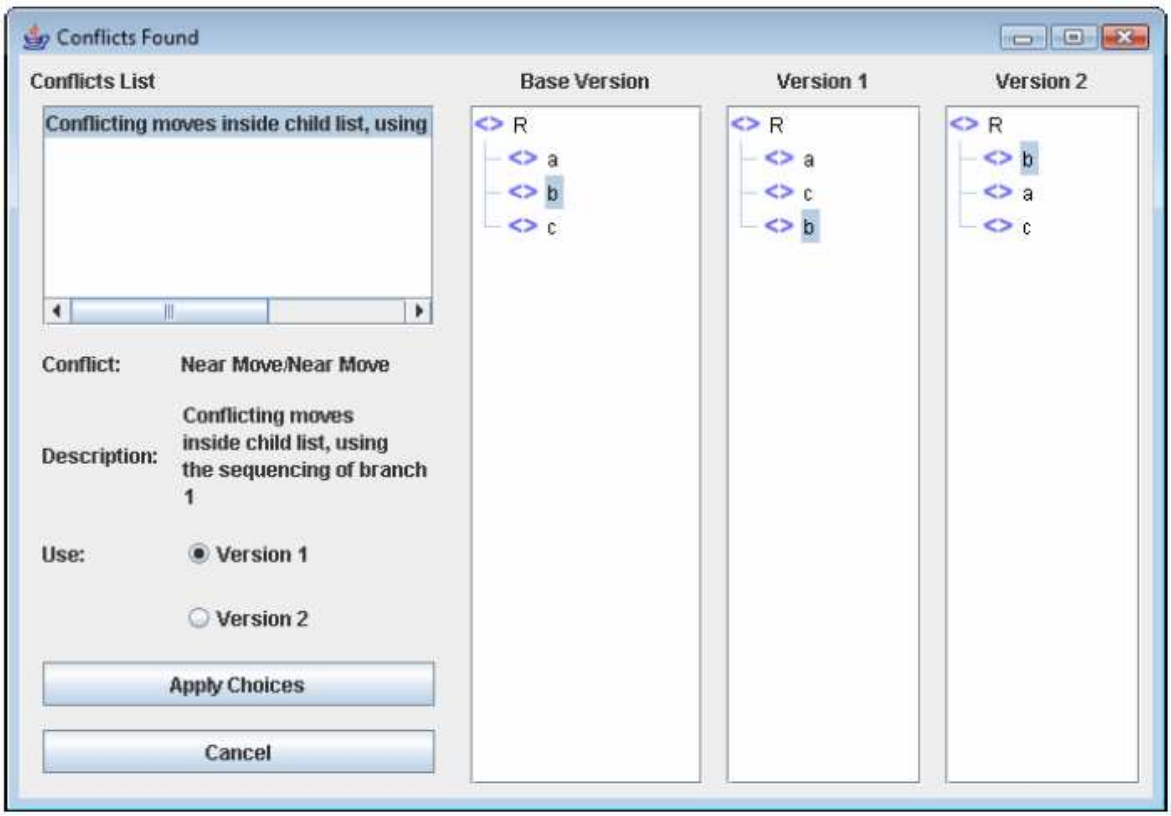


**FIGURA 3.3 – FERRAMENTA EXIBINDO UM CONFLITO**

### 3.1.4 XSDELTA

O XSDelta (BELOTTO, 2006) foi desenvolvida como trabalho de final de curso em 2006 na Universidade Federal do Rio Grande do Sul (UFRGS). Essa ferramenta visual de comparação de esquemas XML, e alguns de seus complementos, foi implementada em Java, para auxiliar administradores de banco de dados semi-estruturados na gerência do processo de evolução de esquemas XML. Além disso, a ferramenta possui distribuição acadêmica e pode ser executada nos sistemas operacionais Windows e Linux. O XSDelta disponibiliza uma interface que mostra aos administradores do banco de dados semi-estruturados todas as operações envolvidas na evolução de um esquema XML. Possui três funcionalidades principais: conversão de esquemas DTD para esquemas XML *Schema*, processamento dos scripts delta para exibição visual das operações de evolução e a geração do arquivo delta.

O fluxo básico do funcionamento da ferramenta é o seguinte: os esquemas XML são submetidos ao processo de detecção de diferenças; já os esquemas DTD são

inicialmente convertidos para o formato XML *Schema* para depois serem submetidos ao mesmo processo. O algoritmo de detecção de diferenças escolhido para implementar o XSDelta foi o XyDiff devido ao suporte das operações de inserção, exclusão, atualização e movimentação, o que diminui o tamanho do delta, além de sua complexidade de execução (O(n log n)), obtida através da otimização de uso de memória e tempo de processamento.

Após a detecção das diferenças, a ferramenta retorna as saídas do algoritmo de detecção, mostrando quais as modificações foram encontradas entre os esquemas, junto à opção de geração do arquivo delta, de formato XML. Contudo, o XSDelta possui algumas limitações, tais como: não permite a edição de documentos e não apresenta a opção de visualização dos casamentos.

Apesar dos esforços, não foi possível testar e avaliar a ferramenta, pois não foi encontrada para download.

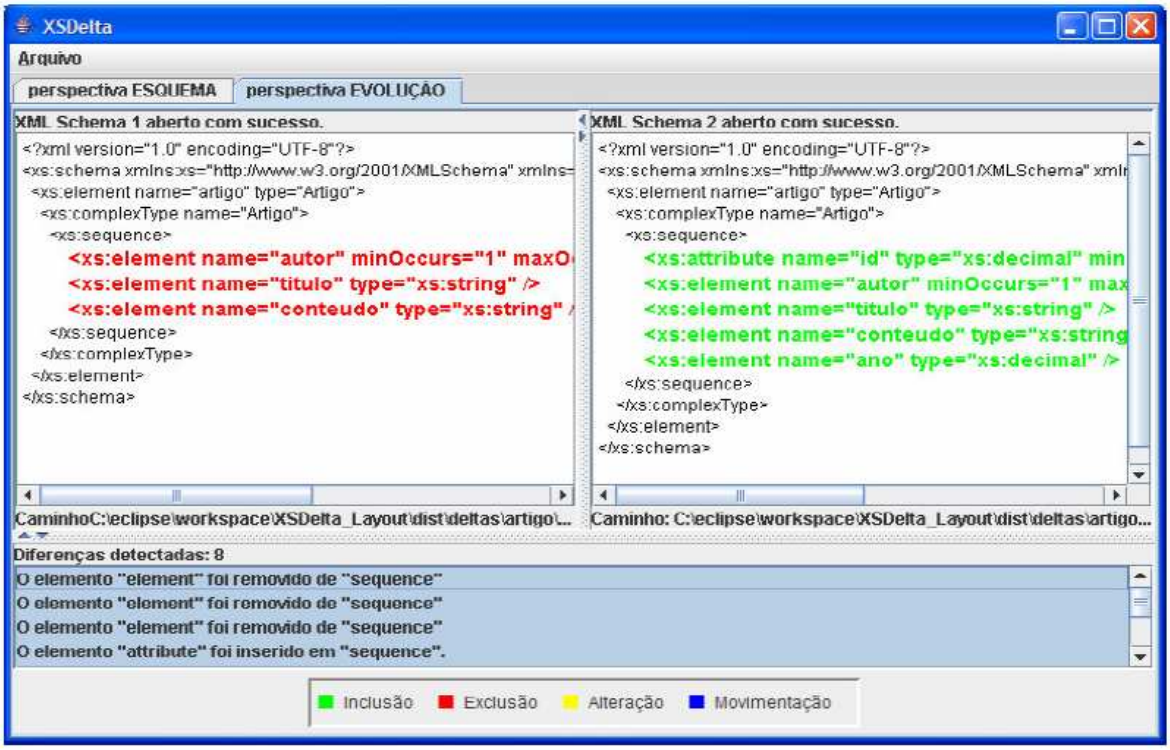


**FIGURA 3.4 – VISUALIZAÇÃO DAS DIFERENÇAS**

### 3.1.5 XVERSION

A última das ferramentas apresentadas é a XVersion (GIACOMEL, 2006). Ela também foi desenvolvida como trabalho de conclusão de curso em 2006 na UFRGS. É

uma ferramenta gráfica de comparação entre arquivos XML, cujos objetivos são: permitir a detecção de diferenças entre documentos XML, agrupar um conjunto de documentos XML e realizar consultas em arquivos XML da mesma forma em que é realizada nos bancos de dados tradicionais. O XVersion foi implementado na linguagem Java pois independe do sistema operacional onde a implementação será executada e também porque se deseja que, no futuro, a ferramenta possa ser utilizada em um ambiente *web*. O algoritmo para detecção de diferenças utilizado na ferramenta foi o XyDiff, pois este apresenta boa documentação e um ótimo desempenho, além de gerar resultados, se não ótimos, muito próximos do ideal.

A ferramenta apresenta três estruturas principais: um comparador de documentos XML, um agrupador de documentos XML e uma função de consulta, todas integradas por uma interface que é utilizada para abrir ou salvar documentos XML. O comparador consiste numa função que recebe dois documentos XML como entrada, gerando um delta como saída, que é o resultado da comparação entre eles, realizada pelo algoritmo XyDiff. Vale ressaltar o uso de uma biblioteca chamada JXyDiff, que foi implementada em Java e que era capaz de executar o algoritmo em questão. Depois de comparados os arquivos, o delta é retornado em tela pelo programa, possibilitando gravação ou edição do documento final.

A ferramenta não suporta a opção de visualização de casamentos, contudo mostra as diferenças entre o arquivo original e a sua nova versão. O agrupador consiste em juntar uma porção de documentos XML em somente um utilizando-se de um algoritmo chamado SCCS. A partir dos documentos XML fornecidos como entrada executa-se o algoritmo de detecção de diferenças, gerando as diferenças entre cada dupla de arquivos. Após isso, é criado um arquivo XML único, utilizando o delta gerado anteriormente. Por fim, tem-se a função de consulta, que tem como objetivo permitir ao usuário abrir um documento criado utilizando o SCCS e filtrar a informação desejada através da execução de consultas sobre este documento. O XVersion foi disponibilizado para download pelo próprio autor e a partir disso testado e avaliado.

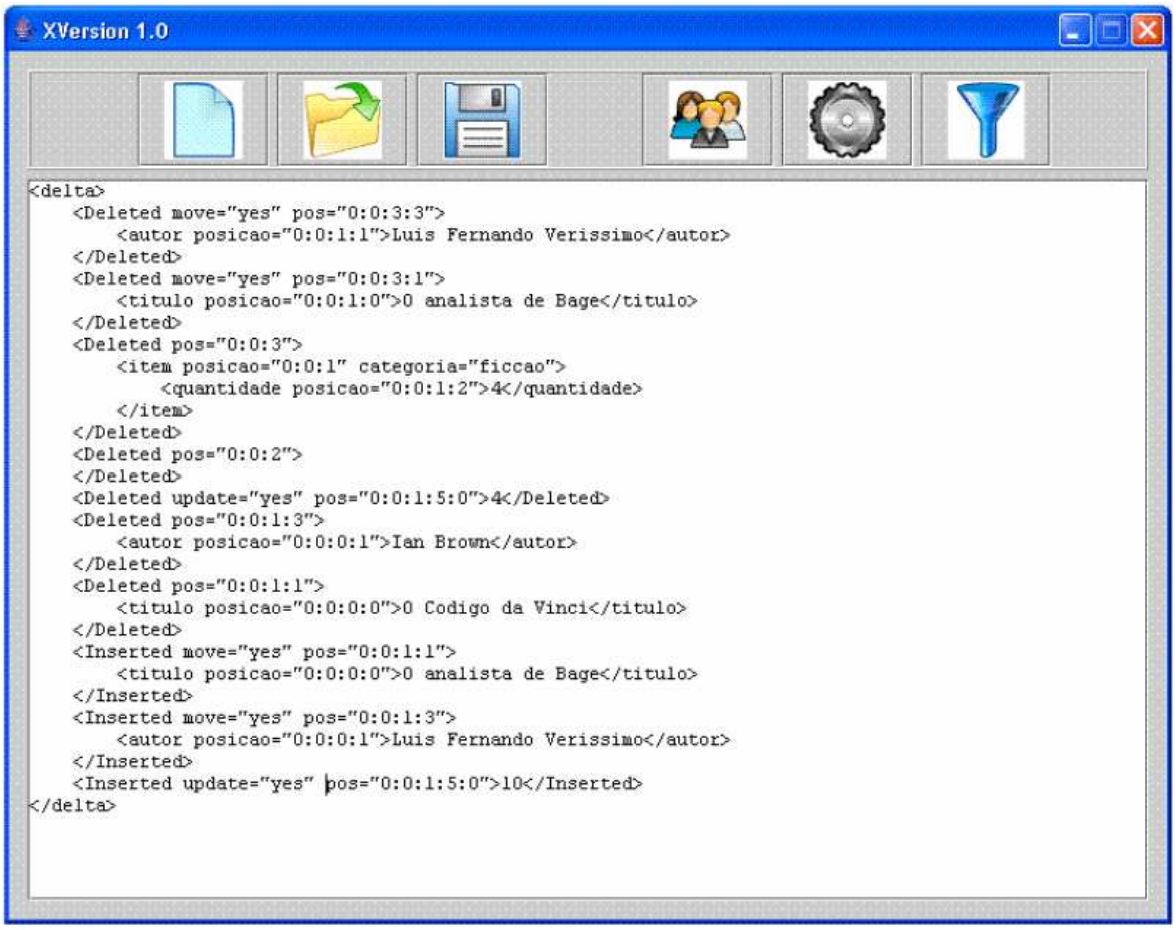


**FIGURA 3.5 – ARQUIVO DELTA GERADO PARA DOIS DOCUMENTOS**

## 3.2 ITENS UTILIZADOS PARA A COMPARAÇÃO DAS FERRAMENTAS

Esta seção descreve os itens utilizados na comparação das ferramentas de detecção e mesclagem de documentos XML. Na próxima seção é apresentado um quadro comparativo entre as ferramentas mostradas na seção 3.1.

O primeiro item a ser abordado é a escolha do algoritmo. Cada um deles foca em determinados aspectos e possui vantagens, otimizações e requisitos peculiares. Por possuírem propósitos similares, alguns algoritmos têm uma conexão forte com os outros, sendo concebidos a partir de melhorias ou adaptações sobre os anteriores. Os algoritmos das ferramentas pesquisadas possuem a mesma estrutura, recebendo como entrada dois documentos XML a serem comparados. Após isso, são criadas as representações em formato de árvore para cada documento fornecido e calcula-se a distância mínima entre as duas árvores criadas, através de um algoritmo *Tree Edit Distance*. Esse algoritmo gera um

conjunto mínimo de operações capaz de transformar um documento XML em outro. A saída gerada por esses algoritmos é o conjunto de operações e é chamado de *Delta Script*.

O próximo item é a complexidade do algoritmo. Como existe mais de um algoritmo para resolver problemas similares, a análise de complexidade computacional é fundamental. O fato de um algoritmo resolver um dado problema não significa que seja aceitável na prática, a exemplo de quando se lida com grandes volumes de dados. As funções usadas para calcular a ordem destes algoritmos contêm apenas uma variável *n*, que representa o número de nodos da árvore dada como entrada. Também é interessante saber a velocidade que a ordem de um algoritmo cresce, ou seja, se o tempo de execução cresce linearmente ou quadraticamente.

Um importante item utilizado na comparação é o das operações suportadas. Embora exista um total de cinco operações de edição possíveis em uma árvore (*delete, insert, update, move* e *copy*), nem todos os algoritmos geram um d*elta script* com estas cinco possibilidades. Isto acontece porque para alguns algoritmos continuarem eficientes, eles abrem mão de uma análise mais aprofundada das alterações em uma árvore e procuram por apenas um subconjunto das operações possíveis de serem feitas, ou seja, operações de *move* e *copy*, por gerarem grande impacto no tamanho dos *deltas scripts* são substituídas por *inserts* e *deletes*. Esta simplificação é possível, pois três das cinco operações possíveis podem ser substituídas por uma seqüência de operações *Insert* e *Delete*.

Usar poucas operações no *delta script*, embora torne o algoritmo mais eficiente, pode tornar o d*elta script* muito extenso o que tornará o mesmo mais trabalhoso de ser analisado.

Outro item diz respeito à distribuição da ferramenta. Se a mesma é de origem comercial, *open source* ou acadêmica. As ferramentas comerciais, como a DiffDog, somente liberam o uso mediante pagamento da licença. Mesmo assim não fornecem o código fonte da ferramenta. Já as ferramentas *open source* permitem o seu uso, estudo, adaptação e distribuição para quaisquer fins, conforme garantidos pela definição de Software Livre. Por fim, as ferramentas acadêmicas são desenvolvidas, geralmente, em projetos de graduação, teses de mestrados e doutorados. Nem sempre é possível ter o código fonte dessas ferramentas.

O item linguagem de programação refere-se a qual linguagem foi utilizada para o desenvolvimento da ferramenta. Na maioria das ferramentas são utilizadas linguagens *open source* como Java e C devido a maior documentação e não ter que pagar por uma licença.

Já o item portabilidade refere-se em quais sistemas operacionais a ferramenta pode ser executada. Atualmente, quanto melhor a portabilidade, mais usuários aderem à utilização da ferramenta.

O último item, não menos importante, fica a cargo das funcionalidades das ferramentas. Dentre elas destacam-se a detecção de alterações, a mesclagem de documentos e a edição dos documentos XML na própria ferramenta.

## 3.3 CONCLUSÃO

A Tabela 3.1 apresenta um quadro comparativo das ferramentas abordadas. Essa comparação foi realizada com objetivo de extrair as principais características de cada ferramenta e, também, seus pontos negativos, isto é, mostrar quais as funcionalidades que poderiam ser acrescentadas para construção da nova ferramenta, a XPerseus.

| Itens// Ferramentas | XSDelta | VisualX | DiffDog | XMLTree Merge | XVersion |
|---|---|---|---|---|---|
| Função | Comparação de esquemas XML | Diferença | Diferença | Diferença e mesclagem | Diferença, mesclagem e consultas |
| Algoritmos utilizados | XyDiff | XyDiff, JXyDiff e XKeyMatch | Algoritmo proprietário | 3dm | JXyDiff |
| Complexidade | O(n log n) | O(n log n) | * | O(n) | O(n log n) |
| Operações suportadas | Insert, Delete e Update | Insert, Delete e Update | * | Insert, Delete, Update e Move | Insert, Delete, Update e Move |
| Distribuição | Acadêmica | Acadêmica | Comercial | Acadêmica | Acadêmica |
| Linguagem | Java | Java | * | Java | Java |
| Portabilidade | Linux, Windows | Linux, Windows | * | Linux, Windows | Linux, Windows |
| Detecção de diferenças | Sim | Sim | Sim | Sim | Sim |
| Mesclagem | Não | Não | Sim | Sim | Não |
| Navegação entre diferenças | Sim | Sim | Sim | Sim | Não |

**TABELA 3.1** – Quadro Comparativo

# 4. PROTÓTIPO DA FERRAMENTA XPERSEUS

O objetivo deste capítulo é apresentar a ferramenta visual para detecção de diferenças de documentos XML denominada XPerseus. Inicialmente é dada uma visão geral da ferramenta. Em seguida, são apresentados os algoritmos e as tecnologias utilizadas na implementação dessa ferramenta. Por fim, é apresentado, por meio de *screenshots*, um exemplo prático da utilização da XPerseus.

## 4.1 VISÃO GERAL

A XPerseus é uma aplicação concebida para realizar a detecção das diferenças entre dois documentos XML. Essa ferramenta possui funcionalidades que permitem ao usuário um versionamento mais preciso e menos propenso a erros.

A ferramenta apresenta uma interface clara e simples que permite abrir dois arquivos lado a lado e verificar as alterações que ocorreram entre essas duas versões de documentos XML. A XPerseus disponibiliza a opção de salvar as diferenças detectadas em um arquivo XML, chamado de delta. Esse arquivo pode ser utilizado posteriormente para realizar a mesclagem desses documentos.

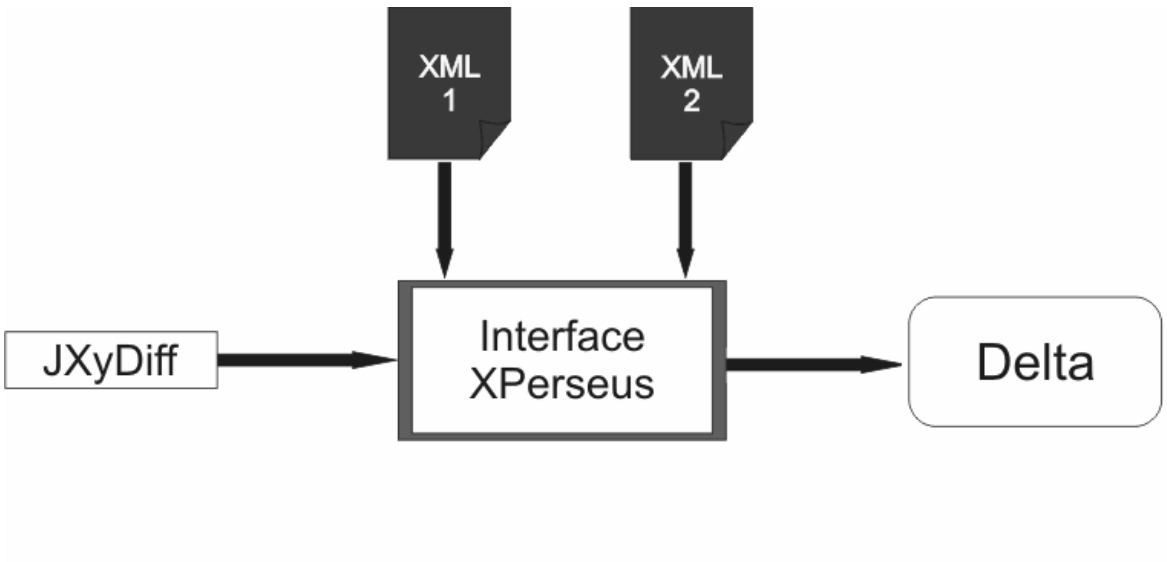


**Figura 4.1 – Arquitetura da XPerseus**

## 4.2 ALGORITMOS E TECNOLOGIAS UTILIZADOS

Para implementar a ferramenta XPerseus, foi utilizada a linguagem de programação Java devido as suas vantagens como, por exemplo, a independência de plataforma utilizada, ou seja, pode ser utilizada tanto em ambiente Windows quanto em Linux; a linguagem de programação é gratuita; existe uma vasta documentação e várias APIs utilizadas na implementação já existiam para essa linguagem como a API Swing utilizada para a criação da interface e a API JDOM utilizada para acessar, manipular e formatar os documentos XML.

A representação dos documentos XML é feita pela interface DOM (*Document Object Model*). Todos os componentes do XML são transformados em nodos de uma árvore como elementos, texto, atributos comentários etc. Após o processamento completo do documento XML, a memória deverá conter a árvore de objetos DOM, a qual disponibiliza para as aplicações qualquer informação relacionada à estrutura e ao conteúdo do documento.

O modelo implementado pela interface DOM é a forma mais flexível para manipulação do conteúdo de um documento XML. Este modelo não é limitado pela ordem em que as estruturas do documento são processadas, logo é possível navegar pelo documento em qualquer ordem. Contudo, como utilização do objeto DOM requer a leitura de toda estrutura XML em uma árvore na memória, pode acontecer um alto consumo de recursos da máquina.

Baseando-se no estudo comparativo sobre alguns algoritmos existentes apresentados na seção 2.5.3, e levando em consideração a possibilidade de reutilizar implementações *open source* disponibilizadas pelos autores, destaca-se um algoritmo que se aproxima das necessidades desse trabalho. O algoritmo utilizado na implementação da XPerseus para a detecção de diferenças foi JXyDiff (XyDiff). Esse algoritmo define o desempenho da XPerseus como um todo, independente do projeto.

O JXyDiff é uma versão, em Java, do algoritmo XyDiff implementado em C++. A versão em Java pode ser encontrada em (TANI, 2006). Sucintamente, o funcionamento do algoritmo XyDiff ocorre da seguinte forma: o XyDiff busca a maior sub-árvore sem modificações possível. A partir desta sub-árvore, são procurados nos nodos pais e filhos da mesma, outros nodos que não sofreram modificações, filtrando, ao final de sua execução, os nodos modificados dos não modificados.

O algoritmo usa também algumas heurísticas para otimizar o desempenho em relação à comparação entre duas árvores. Uma delas é diferenciar a comparação entre nodos que representam texto e nodos que representam atributos, já que, para nodos de atributos, o ordenamento não é considerado. Além disso, nodos representando atributos não possuem filhos. Outra heurística é comparar apenas os nodos que possuem nomes (ou nomes de pais e filhos) iguais, ao invés de comparar todos com todos, como no isomorfismo entre árvores, diminuindo assim o tempo de processamento.

O XyDiff gera um delta script com as operações *Insert*, *Delete*, *Update* e *Move*. Esta última é a mais custosa, pois o algoritmo gasta mais processamento para localizar a permutação de nodos. Além disso, um nodo movido de posição pode estar em um contexto totalmente diferente do anterior (pais, irmãos e filhos diferentes), dificultando sua localização. O autor prova que o algoritmo é correto e que executa em ordem O(n log n). Esta ordem linear, apesar de já ser considerada baixa, só é atingida em raros casos, como quando existem muitas alterações no arquivo. Isto significa que o algoritmo, na maioria dos casos, executa num tempo muito abaixo do que o de sua ordem. O autor também mostra que os *Delta Scripts* gerados são em sua grande maioria bastante próximos da resposta ótima.

O XyDiff é possui complexidade (n log n), o que o torna um algoritmo mais rápido. O “n” da complexidade é o número total de nodos dos dois documentos a serem comparados. Esta velocidade é devido à leitura de dois documentos e da geração de um *hash* para cada nodo. O autor prova em (COBÉNA, 2002) que tanto a execução do mapeamento de baixo para cima (descrito no passo 4 da seção anterior, executado para evitar que operações desnecessárias aconteçam), quanto as construções do delta script acontecem em tempo linear.

A inserção de nodos na fila de prioridades tem complexidade (log n). Logo, no pior caso, que é quando nenhum nodo do documento original é mapeado no documento alterado e todos os nodos são inseridos na fila de prioridades, a complexidade do algoritmo torna-se (n log n). É interessante perceber que o pior caso dificilmente acontecerá. Então na maioria das vezes o algoritmo executará num tempo bem menor do que o delimitado pelo seu limite superior.

### 4.2.1 ALGORITMO XYDIFF – ETAPAS DA EXECUÇÃO

O algoritmo recebe como entrada duas versões de um documento XML. Uma outra entrada, opcional, é a DTD que define o esquema das versões, se houver. A saída produzida é um documento, também no formato XML (chamado *delta*), que representa o *edit script*, descrevendo as mudanças entre as versões. As etapas do algoritmo são basicamente:

1. **Transformação das entradas:** No intuito de aproveitar a estrutura hierárquica do documento XML, o algoritmo transforma cada entrada em uma árvore.
2. **Utilizar a informação de atributos ID:** os elementos de um XML podem ter definidos na DTD do arquivo um atributo de ID. A segunda fase da execução consiste em encontrar todos os nodos do XML que possuem este atributo. O XyDiff usará este ID de cada nodo para encontrar um nodo correspondente no XML a ser comparado. Os nodos que não possuem atributo de ID serão localizados pelo seu contexto (informações sobre os pais e filhos destes nodos). Observe que é muito mais simples para o algoritmo localizar um nodo tendo seu ID imutável do que usando heurísticas para encontrar este mesmo nodo em um contexto diferente. Assim, quanto mais nodos de ID o documento XML possuir, mais rápido o algoritmo executa.
3. **Computar assinaturas e ordenar as subárvores pelo peso:** nesta fase, é atribuída a cada nodo uma assinatura. Esta assinatura é um *Hash* gerado a partir do conteúdo do nodo e das assinaturas de seus filhos, e será usada para representar a subárvore cujo nodo raiz é o próprio nodo. Além da assinatura, é calculado também o *peso* de cada nodo. O peso é calculado a partir do tamanho do conteúdo do nodo e dos pesos dos nodos filhos. Após o cálculo de todos os pesos, é construída uma fila de prioridade. Esta fila contém todas as subárvores do documento (representadas pelo seu nodo raiz) e é classificada por ordem de peso. Isto significa que as primeiras posições da fila (e, portanto, com maior prioridade) serão ocupadas pelos nodos de maior peso. A primeira posição será ocupada pelo nodo raiz do arquivo. É esta fila de prioridades que decidirá quais subárvores o algoritmo vai verificar antes a equivalência de seus nodos.

4. **Tentar encontrar equivalências iniciando pelos nodos com maior peso:** neste passo é removida a primeira árvore da fila de prioridade (ou seja, o seu nodo raiz) do arquivo alterado e cria-se no documento original uma lista de candidatos, ou seja, nodos que têm a mesma assinatura. O algoritmo usa várias técnicas para escolher qual nodo dentre os candidatos equivale ao nodo original. Caso nenhum candidato seja aceito, o algoritmo irá procurar uma equivalência nos nodos pais dos candidatos.

5. **Otimização:** usar a estrutura criada para propagar a igualdade entre os nodos para evitar o processamento de tarefas complexas e dispensáveis, é feito um mapeamento de baixo para cima e depois de cima para baixo na árvore procurando nodos cujos pais são equivalentes e que tem o mesmo nome. Assim, o algoritmo evita a detecção de inserções e deleções desnecessárias, melhorando a qualidade do delta script gerado.

6. **Calcular o delta script:** esta fase é dividida em 3 passos. No primeiro, são procurados no documento alterado os nodos sem correspondência no documento original. Estes nodos são marcados como inseridos, deletados ou, caso seu conteúdo tenha sido alterado, como atualizados. A partir destas anotações serão calculadas as operações *Insert*, *Delete* e *Update* do *Delta Script*. No segundo passo, buscam-se os nodos idênticos nos dois documentos, porém com pais diferentes. Sabe-se que estes nodos sofreram uma operação *move*. Ainda assim, nodos com pais iguais nos dois documentos podem ter sido movidos junto com seus pais. Para isto o XyDiff busca diferenças nos filhos dos nodos. Este processo é muito custoso para o algoritmo e devido a este fato são usadas heurísticas que otimizam o processo e que o autor também prova em (CÓBENA, 2002) que produzem resultados muito próximos da solução mínima. No terceiro e último passo as operações são reorganizadas e o delta script é gerado. O autor não explica como este processo é realizado.

Após apresentar as características mais importantes do XyDiff e suas principais etapas de execução, é apresentado um exemplo da utilização da ferramenta XPerseus.

## 4.3 EXEMPLO DE USO

Na Figura 4.2 é apresentado uma simplificação do modelo de implementação da XPerseus, mostrando as principais classes que formam o núcleo de funcionamento da ferramenta. A classe chamada XMLDiff foi criada para realizar a detecção das diferenças. A classe InterfacePrincipal controla toda a implementação da ferramenta. Nessa classe está a implementação de toda a interface da XPerseus.

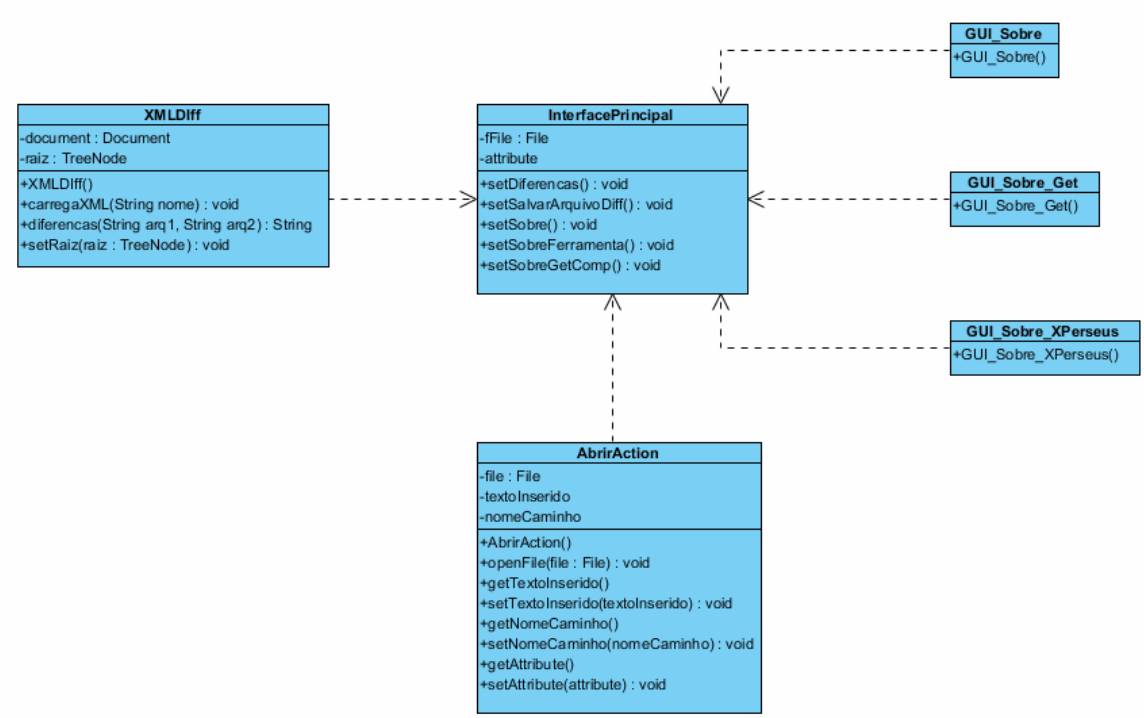


**Figura 4.2 – Modelo de implementação da XPerseus**

Ao executar a ferramenta XPerseus, a tela mostrada na Figura 4.3 é exibida. Nela o usuário pode abrir os dois arquivos XML através do ícone de uma pasta ou através do menu Arquivo→ Abrir, executar o algoritmo de diferenças e salvar o delta gerado.

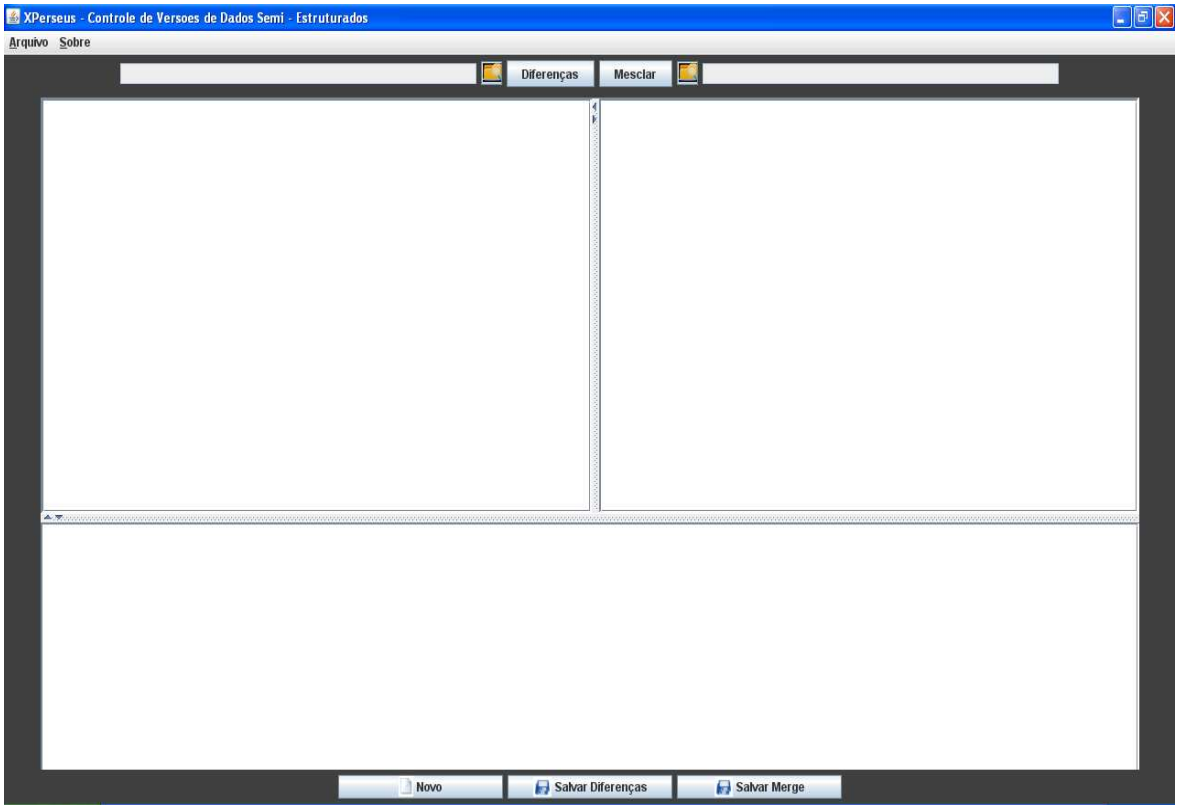


**Figura 4.3 – Interface Principal da XPerseus**

Para esse estudo de caso foi utilizado os arquivos XML como mostra a Figura 4.3.

```
<?xml version="1.0"?>
<ferramenta>
    <usuario id="0">
        <nome>Rafael</nome>
        <idade>26</idade>
        <email>teste@gmail.com</email>
    </usuario>

    <usuario id="1">
        <nome>Luiz</nome>
        <idade categoriaIdade="Cat A2">20</idade>
        <email>luiz@ig.com.br</email>
    </usuario>
</ferramenta>
```

```
<?xml version="1.0"?>
<ferramenta>
    <usuario id="0">
        <nome>Rafael</nome>
        <idade>26</idade>
        <email>teste@gmail.com</email>
    </usuario>

    <usuario id="1">
        <nome>mudei aqui!</nome>
        <idade>20</idade>
        <email>luiz@ig.com.br</email>
    </usuario>
</ferramenta>
```

**Figura 4.4 – Arquivos XML utilizados como exemplo**

Para abrir os arquivos, basta clicar no ícone de uma pasta para cada um dos arquivos como mostrado na Figura 4.5. Ao clicar no ícone é aberta uma janela solicitando que o usuário escolha qual arquivo necessita comparar. Após abrir cada arquivo, seu caminho é mostrado no campo acima do arquivo já aberto.

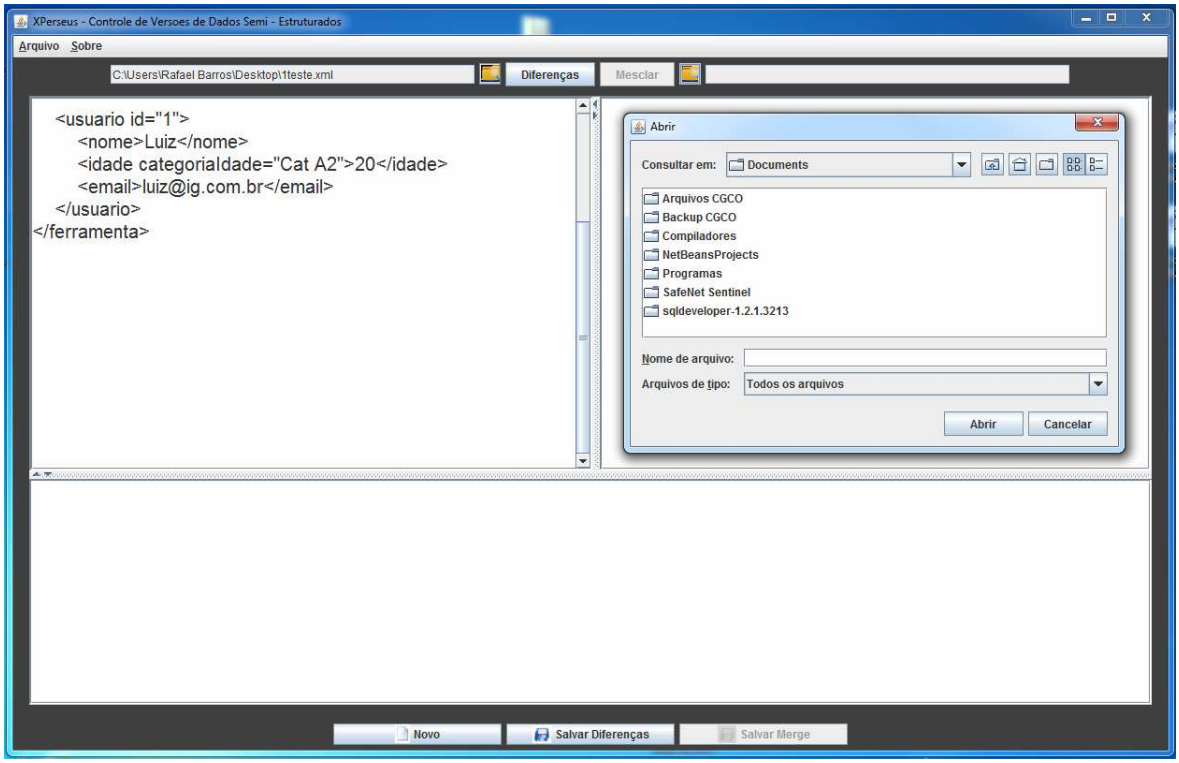


**Figura 4.5 – Abertura dos arquivos XML**

A funcionalidade de detecção de diferenças consiste em uma função que recebe dois arquivos XML como entrada e retorna como saída o *Delta* resultante da comparação entre esses arquivos. Foi utilizado a biblioteca JXyDiff que executa o algoritmo de detecção XyDiff.

Após a realização da comparação, é retornado, em formato de String, o delta script com as diferenças entre os documentos XML. Esse arquivo gerado pelo XyDiff mostra as operações de deleção e inserção realizadas no arquivo com a versão mais antiga. Também mostra a posição original dos nodos, para que, eventualmente, possa ser localizado dentro do arquivo XML.

As posições no arquivo delta funcionam da seguinte forma. Observando a Figura 4.6, tem-se que a linha “<Deleted update = “yes” pos = “0:1:3:1:0”> Luiz </Deleted>” mostra a posição 0:1:3:1:0. Isto é, ocorreu uma alteração no segundo nível, no nodo ferramenta (0:1) representado por um número diferente de zero, nesse exemplo o 1. Esse número representa o nodo alterado, nesse caso o nodo ferramenta. Prosseguindo, dentro do nodo ferramenta, o terceiro nodo (usuário) foi alterado, representado pelo número 3 (0:1:3). Esse nodo é o “<usuário id = “1”>”. E dentro desse nodo, o nodo <nome> foi alterado (0:1:3:1:0). E assim é aplicado para as outras alterações, inserções e deleções.

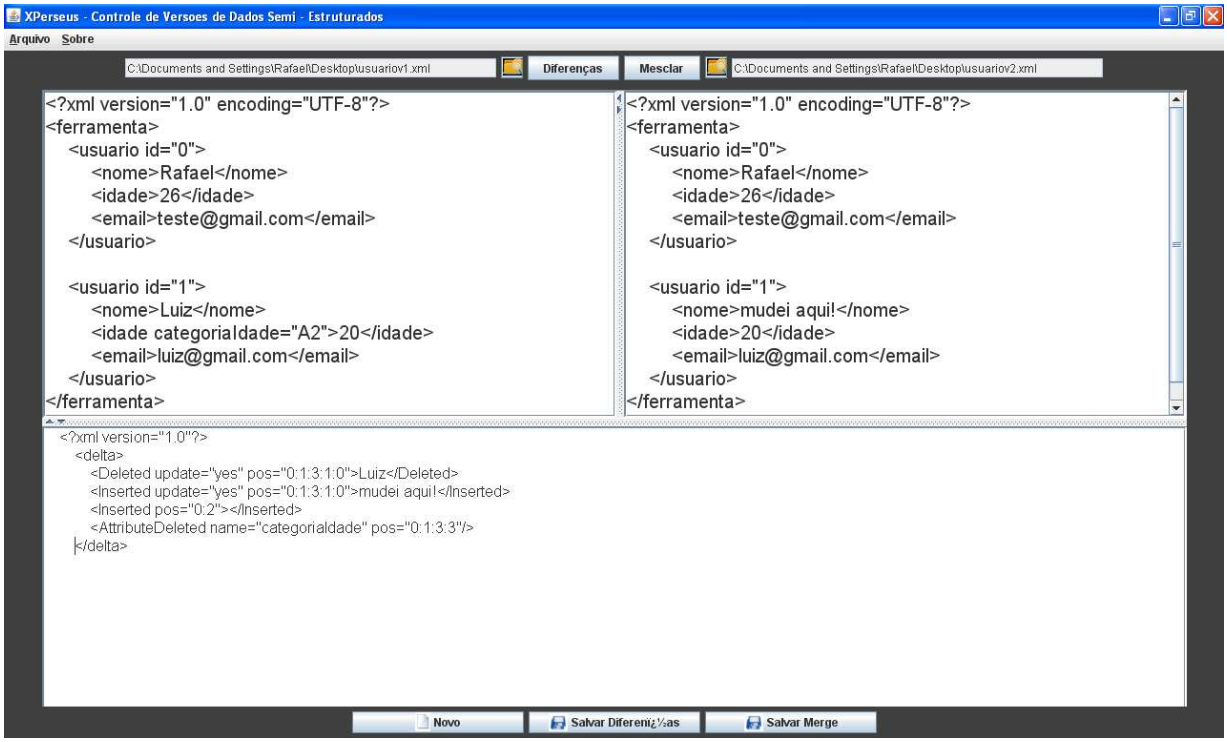


**Figura 4.6 – Delta Script gerado**

## 4.4 CONCLUSÃO

Este capítulo apresentou a XPerseus, uma ferramenta desenvolvida com os objetivos de analisar documentos XML através da detecção das suas diferenças O algoritmo utilizado foi o XyDiff (JXyDiff), pois apresentou a melhor relação de custo e documentação existente.

## 5. CONSIDERAÇÕES FINAIS

Esta monografia apresentou a XPerseus, uma ferramenta visual capaz de realizar a detecção de diferenças entre documentos XML com o objetivo de tornar o processo de versionamento de dados semi-estruturados mais preciso e menos propenso a erros.

A XPerseus disponibiliza aos usuários a geração de um arquivo XML, chamado delta script ou somente delta, que contém todas as diferenças entre dois arquivos XML.

As principais contribuições dessa ferramenta foram, além da detecção das diferenças entre dois documentos XML, uma interface gráfica clara e simples, pois a maioria das ferramentas que comparam documentos desse tipo são aplicações de linha de comando, complicadas de serem utilizadas por usuários acostumados a utilizarem ferramentas com interface gráficas. Uma outra contribuição é toda a base para a adição de uma nova funcionalidade, a de mesclagem de documentos. A XPerseus já possui grande parte da implementação dessa funcionalidade, não sendo disponibilizada nesse trabalho devido ao curto tempo para o desenvolvimento da mesma.

Espera-se que a XPerseus seja o primeiro passo para a construção de uma ferramenta cada vez mais eficiente e com várias funcionalidades voltadas diretamente para o meio acadêmico.

Para projetos futuros pode-se também abordar a questão da criação algoritmos próprios de detecção e mesclagem de documentos e a inserção de outros algoritmos já existentes para possibilitar a escolha pelo usuário no momento de executar as funcionalidades da ferramenta. Outro objetivo poderia ser uma nova abordagem na ferramenta realizando uma análise semântica das alterações identificadas para possibilitar um controle de mudanças destes documentos. Por último, avanços na edição dos documentos são propostos, pois diversas ferramentas no mercado não dispõem dessa funcionalidade.

Considera-se que os objetivos propostos neste trabalho foram atingidos e que as propostas de melhorias surgiram em decorrência do desenvolvimento do trabalho.

## REFERÊNCIAS BIBLIOGRÁFICAS


BRAY, T.; MALLER, E.; YERGEAU, F.; SPERBERG-MCQUEEN C.M.; PAOLI, J. **Extensible Markup Language (XML) 1.0 (Fifth Edition)**. [S.1.]: W3C, 2008. W3C Recommendation, Disponível em: <http://www.w3.org/TR/REC-xml/>. Acesso em Fevereiro, 2011.

CECCHIN, FRANTCHESCO, HARA, CARMEM. **Uma interface gráfica para algoritmos de detecção de diferenças entre documentos xml.** Escola Regional de Banco de Dados (ERBD), 2009.

CEDERQVIST, P. **Version Management with CVS**. Technical Report, 1993.

CHACON, S. **Pro Git**. 288p. Pub. Apress, August, 2009.

COBENA, G; ABITEBOUL, S; FRANCE, R; MARIAN, A. Detecting Changes in XML Documents. In: ICDE, 2002. Proceedings…[S.1.]: IEEE Computer Society, 2002. p.41-52.

COBENA, G.; ABDESSALEM, T.; HINNACH, Y. A comparative study for XML change detection. In: Bases de Données Avancées (BDA), 2002. INRIA, 2002.

DIFFDOG. Disponível em: <http://www.altova.com/diffdog/diff-merge-tool.html>, Último acesso: 08/07/2011.

ELMASRI, R. E., NAVATHE, S., Sistema de Banco de Dados, 4ª Edição, Editora Addison Wesley, 2005

EPSTEIN, D.; CURBERA, F. **XML TreeDiff – A set of Java beans that enable efficient differentiation and updating of DOM trees.** Disponível em: http://alphaworks.ibm.com/tech/xmltreediff. Acesso em: Junho/2011

GIACOMEL, F. S., **XVersion - Uma Ferramenta Gráfica para Gerenciamento e Consulta de Versões de Documentos XML.** Tese de Graduação, UFRGS (2006).

IEEE, 2005, **Std 828 - IEEE Standard for Software Configuration Management Plans, Institute of Electrical and Electronics Engineers.**

LEON, A. **A guide to software configuration management**. Norwood, MA, USA: Artech House, Inc., 2000.

MASON, M. **Pragmatic Version Control: Using Subversion (The Pragmatic Starter Kit Series)**. Pragmatic Bookshelf, 2006.

MENANDRO, D. T. **Análise Comparativa entre Sistemas de Controle de Versões**. Tese de Graduação, UFJF (2010)

MURTA, L. G. P. **Gerência de configuração no desenvolvimento baseado em componentes**. 213p. Tese de Doutorado, COPPE, UFRJ, Rio de Janeiro, Brasil, 2006.

MYERS, E. W., **An O (ND) Difference Algorithm and its Variations, Algorithmica**, v. 1, n. 2 (Março), p. 251-266, 1986.

OLIVEIRA, A. P., OLIVEIRA, A. M., BRAGANHOLO, V., MURTA, L. (2010). **Gerenciando alterações em documentos XML.** Artigo publicado na revista RITA. Volume 17, número 3 (2010).

O'SULLIVAN, B. **Mercurial: The Definitive Guide**, 288p, Pub. O'Reilly Media, June, 2009.

PERINI, A. B., SILVEIRA, V. N. K., GALANTE, R. (2006). **Xsdelta: Uma Ferramenta Visual para Comparação de Esquemas XML.** SBBD (Brazilian Symposium on Databases).

PETERS, L. J. **Change Detection in XML Tress: a Survey**. In: 3rd Twente Student Conference on IT, University of Twente, Enschede, Netherlands. June, 2005**.**

PRESSMAN, Roger S. **Engenharia de software.** 5. Ed. São Paulo, SP: McGraw-Hill, 2002. 843 p. ISBN 8586804258.

SACCOL, D. B.; HEUSER, C. A., **Materialização de Visões XML**, In: XXVII Conferência Latino americana de Informática, Mérida. Livro de Resumos, 2001.

TANI, R.; MOLLI, P.; JOUILLE, F. **JXyDiff LibreSource**. Disponível em http://potiron.loria.fr/projects/jxydiff/; Acesso em: setembro 2006.

VENUGOPALAN, V., **CVS Best Practices** - Revision 0.6, Free Software Foundation, 2002.

WANG, Y.; DEWITT, D. J.; CAI, J. Y. **X-Diff: an effective change detection algorithm form XML documents**. In: Data Engineering, 2003. Proceedings 19TH Internacional Conference on, 2003. Proceedings... [S1.: s.n],2003. P.519-530.

W3C, **Extensible Markup Language (XML) 1.0 (Fifth Edition)**, W3C, 2008.