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

XChangeMerge: Uma abordagem para visualização de diferenças e mesclagem de documentos XML

Lenita Martins Ambrósio

JUIZ DE FORA
AGOSTO, 2013

XChangeMerge: Uma abordagem para visualização de diferenças e mesclagem de documentos XML

Lenita Martins Ambrósio

Universidade Federal de Juiz de Fora

Instituto de Ciências Exatas

Departamento de Ciência da Computação

Bacharelado em Ciência da Computação

Orientadora: Alessandreia Marta de Oliveira

JUIZ DE FORA

AGOSTO, 2013

XChangeMerge: Uma abordagem para visualização de diferenças e mesclagem de documentos XML

Lenita Martins Ambrósio

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. (UFRJ)

Luciana Conceição Dias Campos
D. Sc. (PUC - RJ)

Victor Stroele de Andrade Menezes
D. Sc. (UFRJ)

JUIZ DE FORA

30 DE AGOSTO, 2013

À minha família.

Aos meus amigos.

Ao Yuri.

Resumo

Documentos XML estão sendo cada vez mais utilizados para a representação de dados na *Web*. Entretanto, tais documentos possuem uma estruturação própria, apesar de serem documentos de texto. Diante disso, pode ser necessária a adaptação de técnicas existentes para viabilizar a detecção de diferenças e a mesclagem eficiente deste tipo de documento. Diferentes algoritmos vêm sendo propostos neste sentido e algumas destas abordagens são apresentadas e comparadas neste trabalho. Além disso, esta monografia apresenta o XChangeMerge, uma abordagem que permite a visualização de diferenças e o controle manual da mesclagem de documentos XML. Baseada no algoritmo 3DM, esta proposta exibe os documentos XML, como uma árvore ordenada, o que permite uma visualização mais clara das alterações. Nesta ferramenta, o usuário pode também acompanhar as alterações aplicadas durante a mesclagem, tornando este processo mais controlado e menos propenso a erros.

Palavras-chave: XML, detecção de diferenças, mesclagem, árvore, visualização.

Agradecimentos

Agradeço a Deus por ter iluminado meu caminho, enchendo-o de boas oportunidades.

Ao meu pai por me apoiar em todas as decisões, sempre se esforçando ao máximo para que eu alcançasse meus objetivos.

À minha mãe, por ter me ensinado desde pequena a valorizar o estudo acima de tudo.

Ao meu namorado, pela dedicação e carinho oferecidos em todos os momentos.

Aos meus avós, pelo incentivo e oração.

À minha tia Rosana, por me fazer acreditar neste sonho.

Aos meus amigos do curso, principalmente à Cristiane e ao Leonardo, por transformarem cada dificuldade em uma divertida história de amizade e companheirismo.

Aos amigos do GETComp, que me ajudaram na construção e correção deste trabalho.

À Alessandreia, minha orientadora, que não mediu esforços, em ajudar com tudo o que foi necessário.

Aos professores Victor Menezes e Luciana Campos pela avaliação deste trabalho.

A todos os demais professores, por compartilharem comigo seus conhecimentos, e aos funcionários do curso, que durante esses anos, contribuíram de algum modo para o nosso enriquecimento pessoal e profissional.

"A felicidade às vezes é uma bênção, mas geralmente é uma conquista."

Paulo Coelho

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Abreviações

3DM	3-way merging, Differencing and Matching
CDL	Change Detection by Level
DTD	Document Type Definition
LCS	Longest Common Subsequence
SCV	Sistema de Controle de Versão
XML	Extensible Markup Language

1 Introdução

Documentos XML estão sendo cada vez mais utilizados para a representação de dados na *Web*. Além disso, este tipo de documento é um padrão para integração de sistemas e comunicação entre aplicações diferentes (BASEX, 2013; DADOS ABERTOS, 2013; XML-DATASETS, 2013). Os documentos XML também podem ser utilizados para o armazenamento de informações (BRAY et al., 2008). É comum, que estes documentos sofram muitas mudanças ao longo do tempo, e também que diferentes pessoas alterem o mesmo documento de forma paralela. Desta forma, alterações executadas por um usuário podem comprometer as alterações executadas por outros, tornando necessário um processo de mesclagem entre as versões geradas.

Gerenciar manualmente as mudanças ocorridas além de ser um processo custoso, é muito propício a erros. Com isso, a fim de automatizar o gerenciamento de mudanças, frequentemente, os documentos XML são armazenados juntamente com outros arquivos do projeto, e tratados como documentos de texto simples. Entretanto, devido ao formato estruturado específico dos documentos XML, mesclagens automáticas feitas por sistemas que não consideram este formato podem gerar documentos mal formados.

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 (LINDHOLM, 2001). Este processo de mesclagem é geralmente dividido em duas etapas. A primeira, comumente chamada de diferenciação (*diff*), consiste em calcular um delta que contém uma lista de operações de edição, que, quando aplicado a um dos documentos, produzirá o outro. Na segunda etapa, as alterações feitas em cada uma das versões são conciliadas, por um processo de mesclagem (*merge*), onde a nova versão é produzida.

Para apoiar o gerenciamento de mudança de forma específica para documentos XML, diferentes algoritmos de diferença e mesclagem vêm sendo propostos (COBENA et al., 2002; WANG et al., 2003; RÖNNAU et al., 2009; GUTIÉRREZ-SOTO et al., 2010;

THAO, 2012; OSO, 2013; MACHADO, 2013). Entretanto, além da detecção de diferenças considerando o formato específico dos documentos XML, é desejável que a ferramenta possua um algoritmo de mesclagem que não produza documentos malformados. Outro ponto importante nesse tipo de ferramenta, é que o usuário possa acompanhar as alterações aplicadas durante a mesclagem por meio de uma comparação visual das diferenças, e da resolução manual dos conflitos.

1.1 Justificativa

O uso de documentos XML, vem crescendo de maneira substancial tanto na área de desenvolvimento quanto na troca e no armazenamento de informações. Estes documentos, assim como muitos outros artefatos de software, evoluem ao longo do tempo. Para permitir esta evolução de maneira controlada, estes documentos frequentemente precisam passar por sistemas de gerenciamento de mudança. Entretanto, por ser uma linguagem de marcação com estruturação própria, estes documentos necessitam de um tratamento diferenciado no que se refere à diferenciação e mesclagem.

O uso de ferramentas de gerenciamento de mudanças inadequadas para controlar documentos XML, pode gerar documentos estruturalmente mal formados. Com isso, já existem na literatura diversas propostas de algoritmos e ferramentas que apóiam o processo de mesclagem de documentos XML. Contudo, devido à grande dificuldade desta tarefa, estas propostas ainda possuem muitas limitações, no que diz respeito à complexidade dos algoritmos, à qualidade dos deltas gerados, e ao formato de visualização utilizado. Desta forma, este assunto ainda é alvo constante de pesquisa, justificando o presente trabalho.

1.2 Objetivo

Diante do interesse na detecção de diferenças e mesclagem entre documentos XML, diversas soluções foram desenvolvidas. Esse trabalho tem como objetivo, fazer um levantamento bibliográfico reunindo as principais propostas presentes na literatura, para compará-las em função de alguns critérios definidos.

Além disso, como objetivo principal, destaca-se a integração de uma abordagem que permite a mesclagem de documentos XML ao XChange, denominada XChangeMerge, fornecendo uma visualização eficiente de diferenças através de árvores ordenadas e a resolução manual de conflitos. Isto possibilita ao usuário um maior controle sob as alterações aplicadas durante a mesclagem, tornando este processo mais controlado e menos propenso a erros.

1.3 Organização do trabalho

Esse trabalho está dividido, além desta introdução, da seguinte forma: o capítulo 2 apresenta algumas informações de fundamentação teórica no que se refere à diferenciação e mesclagem de documentos XML. No capítulo 3 são descritos e comparados os trabalhos relacionados. A abordagem XChangeMerge, proposta no presente trabalho, é detalhada e exemplificada no capítulo 4. Finalmente o capítulo 5 apresenta as considerações finais e sugestões de trabalhos futuros.

2 Gerenciamento de mudanças em documentos XML

O gerenciamento de mudanças é de fundamental importância para gerenciar o histórico de um projeto, seja para desfazer, analisar ou recuperar versões estáveis do projeto, bem como para o desenvolvimento colaborativo. Atualmente, existem vários algoritmos e ferramentas para este fim, sendo que muitas têm licença *open source* (código aberto). No entanto, no que diz respeito ao gerenciamento de mudanças em documentos XML, as principais abordagens disponíveis ainda apresentam muitas limitações (MURTA, 2006). A fim de compreender o funcionamento dessas abordagens, este capítulo apresenta alguns conceitos relacionados à detecção de diferenças e mesclagem de documentos XML.

2.1 Controle de versão

O gerenciamento de mudanças geralmente é feito através de um repositório central que armazena todo o histórico de evolução do projeto, e várias cópias locais onde os desenvolvedores trabalham em paralelo. A sincronização entre a cópia local do desenvolvedor e o repositório é feita através dos comandos de *commit* (envia alterações da cópia local para o repositório) e *update* (envia alterações do repositório para a cópia local)(ESTUBLIER et al., 2005).

Para fazer essa sincronização entre o repositório e as cópias locais, os sistemas de controle de versão (SCVs) devem ser capazes de detectar diferenças entre duas versões de um artefato, e mesclar alterações concorrentes do mesmo artefato.

Durante a detecção de diferenças, a maior parte dos SCVs utilizam linhas como unidades de comparação, e arquivos como unidades de versão. Dessa forma, a comparação entre versões é feita linha a linha. Nestas abordagens, os documentos XML são tratados como um arquivo de texto puro, negligenciando todo o conhecimento acerca do formato dos dados (MURTA, 2006). Com isso, estes sistemas apresentam muitos problemas na

detecção de diferenças entre documentos XML. Por exemplo, na tabela 2.1, quebras de linha entre atributos de um elemento, ou mesmo uma troca de ordem entre estes atributos, são considerados alterações, o que para documentos XML, no entanto, são mudanças irrelevantes (OLIVEIRA et al., 2010).

Tabela 2.1: Mudanças irrelevantes em documentos XML

  <b y=2 x=1> <\b>	 <b y=2 x=1> <\b>
a: versao1.xml	b: versao2.xml

Para tratar documentos XML de forma diferenciada, primeiramente é necessário estabelecer unidades de versão e unidades de comparação mais adequadas ao formato. O uso de linhas como unidade de comparação mostrou-se bastante adequado para arquivos simples de texto, mas no caso de documentos XML, que possuem uma estrutura interna mais elaborada, seria mais adequado usar outras unidades, como elementos e atributos (OLIVEIRA, 2007).

2.2 Detecção de diferenças

A etapa de detecção de diferenças entre duas versões de um documento, é comumente chamada de diferenciação. A entrada de um programa de comparação consiste de duas versões do documento XML, e em alguns casos, da sua DTD (*Document Type Definitions*) ou *XMLSchema*. A saída é um documento que representa as alterações entre os dois documentos de entrada (COBENA et al., 2004).

Para efetuar o gerenciamento de mudanças destes documentos 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 (SILVA, 2011). Como dito anteriormente, as unidades de versão mais adequadas para documentos XML são os elementos e os atributos. Contudo, casar (*match*) os elementos correspondentes nas duas versões, não é uma tarefa trivial. Isso porque, na maior parte dos documentos, os elementos não possuem um atributo ID previamente definido na DTD, o que permitiria seu casamento.

Já existem na literatura vários algoritmos de diferença para documentos XML (COBENA et al., 2004). Com isso, já foram propostas várias formas de identificação dos elementos. Muitos destes algoritmos tentam se aproveitar do ID definido na DTD do documento. Quando isso não é possível, alguns deles possuem técnicas heurísticas para determinar chaves que identificam unicamente cada elemento do documento. Neste caso, os documentos a serem comparados são representados em estruturas de árvores, onde cada elemento recebe um identificador de acordo com a heurística utilizada. Finalmente, a correspondência entre os elementos das versões é feita de acordo com seus valores identificadores (WANG et al., 2003).

Outra abordagem presente na literatura, são os algoritmos que se baseiam em técnicas de similaridade. Onde, os documentos XML são comparados por algoritmos de similaridade que, através de algumas métricas, avaliam o quão semelhante os elementos são, e então o casamento é feito entre os elementos equivalentes (RÖNNAU et al., 2009).

Após o casamento, é preciso identificar quais os elementos sofreram alterações, e que tipo de alteração ocorreu. A maior parte dos algoritmos se aproveitam do formato em árvore dos documentos XML para realizar as comparações. Alguns se baseiam em árvores não ordenadas (WANG et al., 2003), enquanto outros se baseiam em árvores ordenadas COBENA et al. (2002). 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, podem ser utilizados tanto a relação entre um elemento e seu ancestral quanto o posicionamento de um elemento em relação aos seus irmãos (SILVA, 2011).

Considerando que nos documentos XML os elementos são ordenados, a rigor, não seria correto modelar este tipo de documento como uma árvore não ordenada, principalmente se houver o interesse em detectar operações do tipo *"move"*, onde a ordem dos elementos é importante (OLIVEIRA et al., 2010).

Existem três tipos de operações básicas, inserção, remoção e alteração. Mas, algumas abordagens utilizam também as operações de movimentação e cópia, como é o caso de LINDHOLM (2004) que classifica as operações em cinco tipos:

- *insert*: consiste na inserção de um nó, um elemento ou um conteúdo textual, em alguma parte do documento. Uma operação é classificada como inserção quando

um nó da versão 2 não possui nenhum correspondente na versão 1;

- *delete*: consiste na remoção de um nó do documento. Uma operação é classificada como remoção quando um nó da versão 1 não possui nenhum correspondente na versão 2;

- *update*: significa que o nó teve o seu conteúdo alterado. No caso de elementos, pode ser o seu nome ou um de seus atributos. Uma operação é classificada como alteração quando o conteúdo do nó da versão 1 é diferente do conteúdo do nó correspondente na versão 2;

- *move*: significa que um nó foi movido para outra parte do documento. Uma operação é classificada como movimentação quando a posição (de acordo com uma determinada definição) do nó na versão 1 é diferente da posição de seu correspondente na versão 2;

- *copy*: significa que um nó foi copiado para outra parte do documento, ou seja, ele continua onde estava, e também aparece em outra parte do documento. Uma operação é classificada como cópia quando um nó da versão 1 possui dois ou mais correspondentes na versão 2.

Identificadas as alterações ocorridas entre as versões do documento, é necessário gerar uma representação dessas alterações, conhecida com *delta*. O *delta* representa as diferenças entre artefatos, e geralmente é composto de operações que levam a construção de um artefato a partir do outro, e pode ter diferentes aspectos dependendo de seu objetivo. Para aplicações que visam gerenciar versões de arquivos, por exemplo, o *delta* deve permitir a perfeita reconstrução de um arquivo a partir de outro, de forma eficiente e eficaz. É importante ressaltar que esse *log* de alterações não deve ser considerado como sendo necessariamente o conjunto de alterações executadas pelo usuário, uma vez que nem sempre é possível detectar qual alteração foi realmente executada (OLIVEIRA, 2007).

2.3 Mesclagem de alterações

A mesclagem de alterações é a parte responsável por fundir modificações executadas em paralelo sobre uma versão base em comum. A mesclagem acontece, se dois ou mais usuários obtém a mesma versão de um determinado arquivo e executam modificações em paralelo. Quando o primeiro usuário terminar suas alterações, para sincronizar o repositório com as novas alterações, ele vai efetuar o *commit*. Quando o segundo usuário terminar e efetuar o *commit*, o sistema detecta que aquele arquivo foi modificado por outro usuário. Então é solicitado que o usuário atualize seu arquivo antes de efetuar o *commit*. Neste momento o sistema precisa consolidar as modificações realizadas pelos usuários em uma mesma versão, é quando a mesclagem acontece (OLIVEIRA et al., 2010).

A mesclagem de documentos, geralmente acontece em duas etapas. A primeira é a detecção de diferenças, descrita anteriormente. Na segunda etapa chamada de conciliação, os casamentos obtidos na fase anterior para as duas versões são utilizados para elaborar a versão final do documento (LINDHOLM, 2001).

A conciliação é uma tarefa que exige muito cuidado, pois, erros cometidos nesta etapa podem resultar na geração de arquivos mal formados. Outra dificuldade desta etapa é que, para cada alteração encontrada, é preciso determinar se ela permanece ou não na nova versão. Esta decisão é bastante complicada, uma vez que podem haver alterações conflitantes de um mesmo elemento do documento, ou seja, as duas versões novas alteraram o mesmo elemento. Neste caso é preciso escolher qual das alterações será mantida na versão final.

Quando a mesclagem é feita somente entre duas versões do documento (*2-way*), as versões são são comparadas uma com a outra. Para a mesclagem do tipo *3-way*, existem três versões do documento, a base e duas novas versões modificadas. Neste caso, cada uma das versões modificadas são comparadas com a versão base, detectando quais alterações foram feitas em cada uma das versões.

Uma das principais abordagens propostas para mesclagem de documentos XML é o algoritmo 3DM (*3-way merging, Differencing and Matching*). No 3DM, um conjunto de regras pré-definidas resolve os conflitos automaticamente, de acordo com o tipo de conflito. Para isso, os possíveis conflitos entre duas versões de um documento XML foram

identificados e classificados considerando as operações de edição envolvidas (LINDHOLM, 2001).

Cada um dos conflitos são descritos e exemplificados a seguir. Os exemplos apresentados são trechos do documento XML gerado por um sistema de "lista de compras compartilhada". Esta lista de compras é gerenciada por meio de um aplicativo de forma que qualquer membro da família possa adicionar coisas para comprar. Esta lista inclui os preços do produto, onde comprar o produto, comentários feitos pelos usuários e também a quantidade do produto a ser comprada.

O conflito *Update/Update* ocorre quando um nó sofre alterações diferentes nas duas versões. A Tabela 2.2 mostra um caso onde o elemento *name* que na versão base era *Emmentaler Cheese* foi alterado na primeira versão para *Colby Cheese* e na segunda para *Cream Cheese*. Neste caso, a solução automática seria usar a versão 1 do documento.

Tabela 2.2: Conflito Update / Update

<shoppinglist> <item> <name>Emmentaler Cheese</name> <quantity>500g</quantity> <price>3.12</price> <shop>LMart</shop> </item> </shoppinglist>	<shoppinglist> <item> <name>Colby Cheese</name> <quantity>500g</quantity> <price>3.12</price> <shop>LMart</shop> </item> </shoppinglist>	<shoppinglist> <item> <name>Cream Cheese</name> <quantity>500g</quantity> <price>3.12</price> <shop>LMart</shop> </item> </shoppinglist>
a: base.xml	b: versao1.xml	c: versao2.xml

Para o caso de nós movidos, a classificação dos tipos de conflito não é a originalmente utilizada por LINDHOLM (2001), já que ele utilizava o mesmo nome para diferentes tipos de conflitos, chamando vários deles simplesmente de *"move"*. Entretanto existem dois tipos de movimentação: *Near Move* significa uma movimentação dentro do mesmo pai, ou seja, uma mera mudança na ordem dos filhos; e *Far Move* significa que o nó foi movido para a lista de filhos de outro pai.

Desta forma, quando um nó é movido em cada uma das versões para uma posição diferente dentro do mesmo pai, acontece um conflito do tipo *Near Move/Near Move*. Este conflito é exemplificado na Tabela 2.3, onde o elemento *shop* que era o último do *item* foi movido na primeira versão para a primeira posição, enquanto na segunda versão ele é movido para a segunda posição, ambos dentro do mesmo *item*. Para este tipo de conflito,

a solução padrão é mover o elemento conforme a versão 1 do documento.

Tabela 2.3: Conflito Near Move / Near Move

<shoppinglist> <item> <name>Napkin</name> <quantity>1 packet</quantity> <price>3.80</price> <shop>Carrefour</shop> </item> </shoppinglist>	<shoppinglist> <item> <shop>Carrefour</shop> <name>Napkin</name> <quantity>1 packet</quantity> <price>3.80</price> </item> </shoppinglist>	<shoppinglist> <item> <name>Napkin</name> <shop>Carrefour</shop> <quantity>1 packet</quantity> <price>3.80</price> </item> </shoppinglist>
a: base.xml	b: versao1.xml	c: versao2.xml

De maneira similar, existe o conflito *Far Move/Near Move* quando um nó é movido para outra posição no mesmo pai em uma das versões, e para uma posição qualquer em outro pai na outra versão. A Tabela 2.4 apresenta um exemplo, no qual o elemento *currency* que na versão base estava dentro de *price*, na primeira versão apenas trocou de posição com o valor, enquanto na segunda foi movido para fora do elemento *price*. Neste caso, a solução automática consiste em ignorar a movimentação dentro do mesmo pai.

Tabela 2.4: Conflito Far Move / Near Move

<shoppinglist> <item> <name>Priness Cake</name> <quantity>1</quantity> <price> <currency>U$</currency> 9.98 </price> <shop>LMart</shop> </item> </shoppinglist>	<shoppinglist> <item> <name>Priness Cake</name> <quantity>1</quantity> <price> 9.98 <currency>U$</currency> </price> <shop>LMart</shop> </item> </shoppinglist>	<shoppinglist> <item> <name>Priness Cake</name> <quantity>1</quantity> <currency>U$</currency> <price> 9.98 </price> <shop>LMart</shop> </item> </shoppinglist>
a: base.xml	b: versao1.xml	c: versao2.xml

O conflito *Far Move/Far Move* é definido pelo autor como movimentações de um mesmo nó para lugares diferentes nas duas versões, nenhum deles sendo o mesmo pai. Entretanto, não foi possível compor um exemplo onde o 3DM acusasse um conflito deste tipo. Ao testar o algoritmo com bases de dados maiores, foram listados alguns conflitos *Far Move/Far Move*, mas como o próprio autor suspeita estes resultados não estavam corretos.

Apesar disso, a detecção ou não deste tipo de conflito, não parece ser realmente

significativa, pois pela descrição dada pelo autor, este conflito deveria ocorrer quando um elemento fosse movido nas duas versões para pais diferentes. Porém se observarmos nas duas versões novas, o elemento que é antigo pai do que foi movido ficou igual nas duas (com menos um filho) portanto não caracteriza um conflito. Da mesma forma, se os elementos que receberam como filho o elemento movido são elementos diferentes, temos que cada versão alterou um elemento independente, o que também não gera um conflito.

Para os conflitos do tipo *Far Move/Far Move*, a solução padrão do algoritmo é manter ambas as movimentações, ou seja o elemento é copiado, e aparece em ambos os lugares.

O conflitos *Far Move/Delete* ocorre quando um nó é movido para outro pai em uma das versões, e removido na outra. O exemplo da Tabela 2.5 mostra que o elemento *suggestion* foi movido na primeira versão para fora do elemento *price*, e foi removido na segunda versão. Para este conflito, a solução padrão é ignorar a remoção, portanto o elemento é movido.

Tabela 2.5: Conflito Far Move / Delete

`<shoppinglist>`
` <item>`
`  <name>Beans</name>`
`  <quantity>500g</quantity>`
`  <price>`
`  <suggestion>red</suggestion>`
`  2.5`
`  </price>`
`  <shop>LMart</shop>`
` </item>`
`</shoppinglist>`
	`<shoppinglist>`
	` <item>`
	`  <name>Beans</name>`
	`  <quantity>500g</quantity>`
	`  <suggestion>red</suggestion>`
	`  <price>2.5</price>`
	`  <shop>LMart</shop>`
	` </item>`
	`</shoppinglist>`
		`<shoppinglist>`
		` <item>`
		`  <name>Beans</name>`
		`  <quantity>500g</quantity>`
		`  <price>2.5</price>`
		`  <shop>LMart</shop>`
		` </item>`
		`</shoppinglist>`

a: base.xml	b: versao1.xml	c: versao2.xml

De maneira similar ao descrito anteriormente, existe o conflito *Near Move/Delete*, que ocorre quando o nó é movido dentro do mesmo pai em uma das versões e removido na outra. Este conflito não estava descrito em LINDHOLM (2001) mas pode-se observar sua existência no *log* de conflitos ao executar o algoritmo. A Tabela 2.6 apresenta um exemplo onde o elemento *price* foi movido em uma das versões, e removido na outra. Neste caso, o algoritmo toma como solução padrão, a versão na qual o elemento foi removido.

Tabela 2.6: Conflito Near Move / Delete

<shoppinglist> <item> <name>Apples</name> <quantity>8</quantity> <price>1.26</price> <shop>LMart</shop> </item> </shoppinglist>	<shoppinglist> <item> <name>Apples</name> <price>1.26</price> <quantity>8</quantity> <shop>LMart</shop> </item> </shoppinglist>	<shoppinglist> <item> <name>Apples</name> <quantity>8</quantity> <shop>LMart</shop> </item> </shoppinglist>
a: base.xml	b: versao1.xml	c: versao2.xml

Por fim, o conflito *Locked/Delete* acontece quando um nó é parte do contexto de uma operação em uma das versões e foi removido da outra. Este conflito é exemplificado na Tabela 2.7. Neste exemplo, observa-se que na primeira versão foi introduzido um novo elemento *kind*, de forma que o anterior passa a ser parte do contexto desta alteração. Já na segunda versão, o elemento *kind* foi removido. Neste caso a solução automática do algoritmo é remover o elemento.

Tabela 2.7: Conflito Locked / Delete

<shoppinglist> <item> <name>Cookies</name> <quantity>1 packet</quantity> <price>1.5</price> <shop>LMart</shop> <kind>chocolate</kind> </item> </shoppinglist>	<shoppinglist> <item> <name>Cookies</name> <quantity>1 packet</quantity> <price>1.5</price> <shop>LMart</shop> <kind>chocolate</kind> <kind>vanilla</kind> </item> </shoppinglist>	<shoppinglist> <item> <name>Cookies</name> <quantity>1 packet</quantity> <price>1.5</price> <shop>LMart</shop> </item> </shoppinglist>
a: base.xml	b: versao1.xml	c: versao2.xml

Além dos conflitos, o 3DM ainda registra avisos de conflito (*"conflict warnings"*), que são conflitos falsos ou que para ele têm uma resolução óbvia e indiscutível.

Quando um nó sofre alterações iguais nas duas versões, o algoritmo registra um aviso do tipo *Equal Updates*. A Tabela 2.8 exemplifica este caso, onde o elemento *quantity* é alterado de 2l para 5l em ambas as versões. É possível perceber que apesar de serem alterações concorrentes, uma vez que as alterações são iguais, este caso não gera um conflito.

Tabela 2.8: Warning - Equal Updates

<shoppinglist> <item> <name>Orange juice</name> <quantity>2l</quantity> <price>1.96</price> <shop>LMart</shop> </item> </shoppinglist>	<shoppinglist> <item> <name>Orange juice</name> <quantity>5l</quantity> <price>1.96</price> <shop>LMart</shop> </item> </shoppinglist>	<shoppinglist> <item> <name>Orange juice</name> <quantity>5l</quantity> <price>1.96</price> <shop>LMart</shop> </item> </shoppinglist>
a: base.xml	b: versao1.xml	c: versao2.xml

De maneira similar, quando um nó é inserido ou copiado para o mesmo lugar em ambas as versões, ocorre um aviso do tipo *Equal Inserts*. Este caso é exemplificado pela Tabela 2.9 onde um elemento *suggestion* foi inserido no mesmo lugar em ambas as versões.

Tabela 2.9: Warning - Equal Inserts

<shoppinglist> <item> <name>Wheat toast</name> <quantity>2 packets</quantity> <price>1.32</price> <shop>LMart</shop> </item> <shoppinglist>	<shoppinglist> <item> <name>Wheat toast</name> <suggestion>whole</suggestion> <quantity>2 packets</quantity> <price>1.32</price> <shop>LMart</shop> </item> </shoppinglist>	<shoppinglist> <item> <name>Wheat toast</name> <suggestion>whole</suggestion> <quantity>2 packets</quantity> <price>1.32</price> <shop>LMart</shop> </item> </shoppinglist>
a: base.xml	b: versao1.xml	c: versao2.xml

Quando dois nós diferentes são inseridos ou copiados na mesma posição em ambas as versões, o algoritmo gera um aviso do tipo *Different Inserts*. A Tabela 2.10 exemplifica este caso, através da inserção de dois itens novos à lista. Neste caso também é fácil perceber que não ocorre conflito, pois apesar dos elementos serem inseridos na mesma posição, eles são elementos diferentes. Portando o algoritmo entende que a mesclagem dessas alterações é óbvia, basta inserir um elemento após o outro.

Tabela 2.10: Warning - Different Inserts

<shoppinglist> </shoppinglist>	<shoppinglist> <item> <name>Apples</name> <quantity>8</quantity> <price>3.26</price> <shop>LMart</shop> </item> </shoppinglist>	<shoppinglist> <item> <name> Calvin and Hobbes Comic (for Kalle´s Xmas party!) </name> <quantity>1</quantity> <price>4.99</price> <shop>LMart</shop> </item> </shoppinglist>
a: base.xml	b: versao1.xml	c: versao2.xml

O último caso, é o aviso do tipo *Delete/Update*. Este aviso ocorre quando um nó foi alterado em uma das versões, e a subárvore a qual ele pertencia foi removida na outra versão. Este tipo de aviso é exemplificado pela Tabela 2.11, onde o elemento *name* foi alterado para *suggestion* na primeira versão, e o item *"Smoked salmon"* inteiro foi removido na segunda versão. Este caso não é considerado pelo algoritmo como um conflito, uma vez que independente da ordem em que as alterações ocorreram, ele entende que o resultado final deve ser a remoção.

Tabela 2.11: Warning - Delete/Update

<shoppinglist> <item> <name>Smoked salmon</name> <quantity>250g</quantity> <price>2.55</price> <shop>LMart</shop> </item> </shoppinglist>	<shoppinglist> <item> <suggestion> A tia Maija é alergica a salmon </suggestion> <quantity>250g</quantity> <price>2.55</price> <shop>LMart</shop> </item> </shoppinglist>	<shoppinglist> </shoppinglist>
a: base.xml	b: versao1.xml	c: versao2.xml

2.4 Conclusão

O presente capítulo, apresentou os principais conceitos presentes nas abordagens de gerenciamento de mudanças de documentos XML. Compreender estes conceitos, bem como conhecer as principais formas de detecção de diferenças e de mesclagem de documentos

XML são fundamentais para entender o funcionamento do XChangeMerge.

3 Diferença e mesclagem de documentos XML

Existem na literatura, diversas abordagens de algoritmos para diferença e mesclagem de documentos XML. Este capítulo apresenta uma visão geral destas abordagens, através de uma breve descrição dos principais algoritmos existentes. Na seção 3.1 são descritos os trabalhos presentes na literatura que estão relacionados ao tema deste trabalho. A seção 3.2 apresenta a abordagem proposta. A seção 3.3 faz uma análise das abordagens apresentadas. Na seção 3.4 é feita uma análise prática de algumas abordagens. Por fim, a seção 3.5 traz uma conclusão para o capítulo.

3.1 Trabalhos relacionados

Dos trabalhos encontrados na literatura, foram selecionados para compor o presente capítulo o XyDiff (COBENA et al., 2002) e o X-Diff (WANG et al., 2003) porque são os pioneiros nesta linha, que serviram de base para as outras abordagens. Os demais algoritmos foram escolhidos por serem abordagens mais recentes, e alguns também por possuírem a opção de mesclagem dos documentos, estando assim mais relacionados com a abordagem proposta.

3.1.1 XyDiff

O XyDiff (COBENA et al., 2002) é um algoritmo de diferenciação entre documentos XML, concebido para verificar alterações entre versões de documentos XML em um projeto que investiga *data warehouses* dinâmicos. Devido ao seu contexto, armazenamento de grandes volumes de dados, é um algoritmo que prioriza a eficiência em termos de espaço de memória e velocidade, mesmo à custa de alguma perda de qualidade. Desta forma, esta abordagem não gera um delta mínimo, ou seja, um conjunto mínimo de operações através das quais pode-se transformar uma versão na outra (MURTA, 2006). Entretanto,

experiências feitas por seus autores, mostram que o resultado do XyDiff é correto, e na maioria das vezes é bastante próximo da solução ótima.

Este algoritmo trabalha apenas com a diferenciação de dois documentos, ou seja, não faz comparação entre mais de duas versões, e não possui a função de mesclagem. Considera as operações de inclusão, remoção, alteração e movimentação. Sendo esta última a mais custosa, pois um nó movido pode estar em um contexto totalmente diferente do anterior (pais, irmãos e filhos diferentes), dificultando sua localização. Utiliza recursos próprios do formato XML, como os IDs definidos no DTD e o modelo de árvore ordenada, onde a ordem dos filhos é relevante na detecção de alterações.

O XyDiff possui complexidade no tempo de $O(nlog(n))$, onde n é o número de nós (PETERS, 2005). Esta ordem linear, que já é considerada baixa, é atingida somente em raros casos, como quando existem muitas alterações no arquivo. Isto significa que o algoritmo, na maioria dos casos, pode ser executado em um tempo inferior ao descrito. A complexidade de memória também é linear em relação ao tamanho do documento.

As entradas para o algoritmo são duas versões de um documento XML, e se existir, a DTD que define o esquema das versões. Sua saída é um documento delta, também no formato XML, que descreve as mudanças entre as versões. Esse algoritmo possui uma etapa preliminar de transformação das entradas. Para que a estrutura hierárquica do documento XML seja aproveitada, cada entrada é transformada em uma árvore. E então a diferenciação propriamente dita é feita em cinco fases descritas a seguir.

A primeira etapa, consiste na utilização da informação de atributos ID. Encontra os nós que possuam um atributo ID definido pela DTD dos documentos. A existência de um atributo ID para um dado nó provê uma condição única para casar tal nó: o valor do ID deve ser o mesmo. Se tal par de nós é encontrado em ambos os documentos, eles são casados. Os nós que não possuem atributo de ID serão localizados pelo seu contexto (informações sobre os pais e filhos destes nós). É mais simples para o algoritmo localizar um nó tendo seu ID, que usando heurísticas para encontrar este mesmo nó em um contexto diferente COBENA et al. (2004).

Em seguida, o algoritmo computa assinaturas e ordena as subárvores pelo peso. Nesta fase, cada nó recebe uma assinatura gerada através de uma função de *hash*, que

considera o conteúdo do nó e as assinaturas de seus filhos para o cálculo de um ID único para o nó. Esta assinatura será utilizada para representar com exclusividade o conteúdo da subárvore inteira, enraizada neste nó. Além da assinatura, é calculado também o peso de cada nó, de acordo com o tamanho do conteúdo, em nós de texto, e com a soma dos pesos dos filhos, para nós de elemento.

Através dos pesos é construída uma fila de prioridade, que contém todas as subárvores do documento. As primeiras posições da fila, com maior prioridade, são ocupadas pelos nós de maior peso. A primeira posição contém o nó raiz do arquivo. Esta fila fornece ao algoritmo qual é a próxima subárvore para a qual ele deve encontrar uma correspondência.

A terceira etapa consiste em tentar encontrar correspondências a partir da fila de prioridades. Neste passo é removida a primeira árvore da fila de prioridade do documento alterado. Uma lista de candidatos é feita com os nós da versão antiga que têm a mesma assinatura. A partir desta lista, o algoritmo obtém o melhor candidato e então faz o casamento entre os nós. Se não há nenhuma correspondência e o nó é um elemento, seus filhos são adicionados à fila.

Se existem muitos candidatos, o melhor candidato é aquele cujo pai já foi correspondido, e equivale ao pai do elemento a ser casado. Se nenhum candidato for aceito, o algoritmo verifica um nível acima. O número de níveis a serem verificados depende do peso do nó. Quando um candidato é aceito, o casamento é propagado para seus antepassados, enquanto eles tiverem a mesma assinatura.

A etapa seguinte é de otimização, e consiste em usar a estrutura criada para propagar os casamentos entre os nós. É feita uma busca *bottom-up* (de baixo para cima) e outra *top-down* (de cima para baixo) na árvore tentando casar os nós da versão original com os da nova versão, de tal forma que eles tenham a mesma assinatura, e seus pais sejam equivalentes. Nesta etapa, o algoritmo evita a detecção de inserções e deleções desnecessárias, melhorando a qualidade do delta gerado.

Por fim, a última etapa é o cálculo do delta. Primeiramente, são procurados no novo documento os nós que não possuem um correspondente na nova versão. Estes são então marcados como inseridos. De maneira análoga, os nós presentes na versão antiga

sem correspondentes na nova versão são marcados como removidos. Os nós que foram casados, mas que possuem conteúdos diferentes, são tidos como alterados. A segunda parte da geração do delta é encontrar os nós que são casados, mas que têm pais diferentes em cada versão, e nós casados que possuem posições diferentes em relação a seus irmãos. Estes sofreram a operação de movimentação. Finalmente, estas operações são reorganizadas e o delta, no formato XML, é produzido.

3.1.2 X-Diff

O X-Diff (WANG et al., 2003) é um algoritmo eficiente para calcular a diferença ideal entre duas versões de um documento XML. Esta abordagem considera características de domínio de documentos XML e introduz vários conceitos de chave, como a assinatura do nó, e *XHash*. O algoritmo é uma combinação destas técnicas com o padrão de correção árvore a árvore, que consiste numa abordagem de identificação de alterações entre documentos no formato de árvore.

Sua principal característica é o uso de árvores não ordenadas, que somente avalia o relacionamento pai-filho, não levando em conta a ordenação entre os irmãos (COBENA et al., 2004). Os autores defendem que, por ter um resultado mais preciso das alterações, este modelo se adequa melhor à maioria das aplicações de banco de dados. No entanto, por usar árvores não ordenadas, o algoritmo não é capaz de detectar operações de movimentação, suportando apenas as operações básicas de inserção, remoção e alteração.

É um algoritmo específico para documentos XML, e encontra o mapeamento mínimo entre os filhos de duas subárvores reduzindo o problema a um problema de fluxo máximo com custo mínimo (PETERS, 2005). O X-Diff é correto e encontra um delta mínimo com complexidade de tempo e memória quadráticos. Os autores propõem algumas heurísticas que melhoram o desempenho, mas com isso, não garante o resultado mínimo. Esta abordagem não tem função de mesclagem entre os documentos, e não suporta a comparação entre várias versões.

Dados dois documentos XML, Doc_1 e Doc_2, considere que T_1 e T_2 são as suas representações em árvore. O X-Diff determina se Doc_2 é diferente de Doc_1. Se os documentos são diferentes, o algoritmo encontra uma correspondência de custo mínimo entre

T_1 e T_2, e gera um *script* de edição de custo mínimo para $(T_1 \rightarrow T_2)$. Para isso, o algoritmo transforma os documentos XML de entrada em estruturas de árvore chamadas *Xtrees*. Em seguida, são executadas três fases.

Primeiramente, é feita a computação de assinaturas. Em ambas as árvores, durante o processo de análise, o algoritmo calcula, através de uma função especial de *hash*, um valor de assinatura *XHash* que identifica cada nó do documento XML. Esta assinatura representa toda a subárvore enraizada no nó.

A segunda etapa consiste em encontrar uma correspondência de custo mínimo entre T_1 e T_2. Primeiro o algoritmo elimina as subárvores equivalentes entre os dois nós raiz, comparando os valores *XHash* de nós filhos de segundo nível. Subárvores com valores *XHash* idênticos podem ser consideradas como equivalentes. Este passo reduz de forma eficaz o tamanho da árvore, evitando cálculos desnecessários nas fases subsequentes do algoritmo.

Posteriormente, o algoritmo calcula a distância de edição, $Dist(T_1, T_2)$, para cada um dos restantes pares de subárvore e obtém correspondências de custo mínimo, $M_{min}(T_1, T_2)$, entre elas. Após esta comparação, é realizado o mesmo procedimento com os nodos pais. Em cada nível, os valores *XHash* dos nós filhos são utilizados para filtrar subárvores equivalentes, a fim de reduzir o espaço correspondente.

A distância de edição entre dois nós folha ou duas subárvores, associados à sua correspondência de custo mínimo, é armazenada em uma tabela de distâncias usada para avaliar os melhores casamentos. Ao calcular a distância de edição entre subárvores, o algoritmo que faz o casamento dos nós usa um algoritmo de fluxo máximo de custo mínimo (ZHANG, 1993) para encontrar o mapeamento bipartido de custo mínimo.

Por fim, na fase de construção do delta, X-Diff gera um *script* de edição de custo mínimo com base no mínimo custo correspondente encontrado na fase anterior. Este procedimento é executado de forma recursiva a partir da raiz até as folhas. Este documento descreve as operações de inclusão, remoção e alteração. Como mencionado anteriormente, não são identificadas operações de movimentação.

3.1.3 DocTreeDiff

O DocTreeDiff (RÖNNAU et al., 2009) é um algoritmo para cálculo de diferença entre documentos XML desenvolvido no contexto do controle de versão de documentos de escritório, principalmente nos formatos do *Open-Document* e do *Office OpenXML*. Neste trabalho, um documento XML é tratado como uma árvore ordenada, com a maior parte do conteúdo armazenado nos nós folha.

Segundo os autores, eles desenvolveram o algoritmo no intuito de que as entradas sigam um padrão esperado para documentos de escritório, que são: (i) o conteúdo está concentrado nas folhas; (ii) as alterações de estrutura são realizadas nos níveis mais altos da arvore; (iii) muitos nós não-folha são iguais devido a marcação idêntica do documento. Os autores também afirmam que o algoritmo poderia ser estendido para contemplar um *3-way merge*, mas o aumento da complexidade previsto seria grande demais para compensar o trabalho.

De acordo com os autores, baseando-se em uma abordagem de programação dinâmica, o DocTreeDiff calcula de forma eficaz as diferenças entre dois documentos XML. É eficiente em termos de tempo e espaço. Sua complexidade de tempo é $O(l(T).D + n)$, onde $l(T)D$ é a multiplicação do número de nós folha das duas árvores pelo número de operações de edição no nível folha e n é o número de nós não folha. A complexidade no espaço é linear $O(T + D)$ onde T é o numero de nós e D número total de operações de edição.

Este algoritmo também segue o modelo de edição árvore a árvore e considera as operações básicas de inserção, alteração e remoção, além da movimentação. No entanto, possui uma definição diferente dessas operações. Uma operação de inserção envolve um nó dentro da árvore, onde uma sequência de árvore deve ser inserida. Uma operação de remoção exclui uma sequência completa da árvore. Operações de alteração envolvem um único nó arbitrariamente dentro da árvore. A operação de movimentação consiste basicamente de um par de operações de inserção e remoção, ligados por um atributo de identificação comum.

Esta abordagem se baseou no algoritmo XyDiff mencionado anteriormente. Contudo, para melhorar a eficiência do algoritmo na transformação dos documentos, e a

qualidade dos deltas gerados, foi introduzido um novo modelo de delta usando as chamadas impressões digitais de contexto. Estas impressões digitais permitem a identificação de operações de uma maneira altamente fiável, levando em conta os nós circundantes.

Através dessas impressões digitais, este algoritmo torna possível a fusão de documentos XML, incluindo a detecção de conflitos. Neste modelo de delta, as operações devem ser invertíveis, principalmente as operações de alteração que devem ter o velho e o novo valor do nó. A inversão de uma operação permite a reconstrução da versão anterior a partir da nova.

A ideia chave do algoritmo é baseada no algoritmo LCS *(Longest Common Subsequence)* (MYERS, 1986), e funciona em três passos.

No primeiro passo, o algoritmo calcula os rótulos de todos os nós folha das árvores a serem comparadas. O rótulo de um nó é dado pelo valor de *hash* do seu conteúdo. O conteúdo do nó depende de seu tipo. Para nós de texto, o conteúdo é o próprio texto. Nós de elementos são representados pela normalização dos nomes de seus atributos. O conteúdo de um nó de processamento de instruções é o nome do nó normalizado.

Estes valores de *hash* são enriquecidos com a profundidade da folha correspondente, isto é, o comprimento do caminho para o elemento de raiz. Então é calculada a maior subsequência comum de todos nós folha. Como resultado, obtém-se uma subsequência comum mais longa de todos os nós folha semelhantes no mesmo nível da árvore XML.

Na segunda etapa, o algoritmo compara os valores *hash* dos pais dos nós já combinados pelo LCS. A incompatibilidade dos valores de *hash* é interpretada como uma operação de alteração, uma vez que eles tiveram sua estrutura preservada. A operação correspondente será criada e adicionada ao delta.

Durante a passagem *bottom-up* da árvore, cada nó é marcado quando é visitado pela primeira vez. Se um nó já visitado é atingido, a travessia é abortada, já que todos os outros nós no caminho para a raiz já foram visitadas. Essa abordagem de programação dinâmica garante que cada nó é visitado apenas uma vez, resultando, portanto numa travessia de complexidade linear.

No terceiro passo são identificadas as operações correspondentes a inserção e

remoção para os nós folha que não foram casados. Todo nó folha da primeira árvore que não faz parte da segunda árvore é considerado como excluído, o inverso é válido para nós inseridos. Para todo nó a ser excluído, o mais próximo nó folha casado e filho do mesmo pai é procurado. Se o nó pai não tem um filho na maior subsequência comum, este também é marcado como excluído. Esta busca é propagada até que um nó filho, parte da maior subsequência comum seja encontrado, ou até atingir o nó raiz. Esta passagem permite a identificação de grandes subárvores a serem excluídas. O mesmo procedimento é realizado para as subárvores inseridas.

Um caso especial acontece se dois nós folha na mesma posição são detectados como removido e inserido, respectivamente. Se os seus nós pais forem idênticos, estas duas operações são substituídas por uma operação do tipo movimentação.

Operações adjacentes de inserção ou remoção que tenham a raiz da subárvore a inserir ou excluir na mesma profundidade são fundidas em uma única operação maior, que contém uma sequência de árvore, e será realizada atomicamente. Durante este procedimento, inserções e exclusões complementares são vinculadas através de um atributo ID, e identificadas como uma operação de movimentação.

3.1.4 CDL

O CDL (Change Detection by Level) (GUTIÉRREZ-SOTO et al., 2010) é um algoritmo para detectar mudanças em documentos XML. Realiza a diferença somente entre duas versões, mas suporta tanto árvores ordenadas quanto não ordenadas. Comporta as operações de inserção, remoção e alteração.

A eficácia do algoritmo reside na aplicação de uma função de *hash* e em como esse valor é utilizado. A complexidade teórica do algoritmo é $O(nlogn + e_G)$, onde n é o número de vértices e e_G é o custo para verificação de todas as arestas.

Este algoritmo transforma cada árvore XML em uma árvore com informações relevantes sobre seus vértices e arestas. Cada árvore pré-processada é decomposta em um sub-caminho mais longo. Se as árvores XML são iguais, então seus respectivos sub-caminhos mais longos devem ser iguais. Caso contrário, não é necessário verificar todos os vértices e arestas das árvores. Esta é a principal vantagem deste modelo.

Este algoritmo pode ser dividido em três partes. Em primeiro lugar, as árvores XML são pré-processadas e são criadas novas árvores. Funções de conversão são aplicadas a cada vértice e aresta da árvore XML, transformando a *string* correspondente em um valor numérico. Além disso, neste passo, a existência de vértices sem rótulos é verificada. Cada vértice indefinido é rotulado, concatenando as informações de seus vértices e arestas adjacentes.

No segundo passo, a ideia principal é a decomposição das árvores pré-processadas em várias subárvores. Para isso, os sub-caminhos mais longos são obtidos e o casamento entre as subárvores é realizado. O algoritmo garante que todas as árvores são decompostas, e que a decomposição é única e completa.

Por fim, o conjunto de subárvores resultante da decomposição feita no passo anterior é a base para um algoritmo eficiente que detecta mudanças em árvores XML. Em vez de comparar todos os vértices das árvores de entrada, é necessário verificar apenas os vértices da subárvore decomposta.

3.1.5 Molhado SPL

Uma pesquisa sobre gerenciamento de configuração feita por THAO (2012), resultou em uma abordagem chamada Molhado SPL, projetado para suportar a evolução de linhas de produtos de software. Para isso, Molhado SPL implementou um algoritmo de mesclagem e versionamento consciente (feito pelo usuário) de documentos XML. O algoritmo não somente mescla dados estruturados dentro da ferramenta, como também pode ser usado como software independente.

O algoritmo de mesclagem implementado segue o modelo *3-way*, e utiliza uma estrutura de dados de árvores especializadas em controle de versão. Esta estrutura auxilia o algoritmo na identificação dos elementos, e das mudanças ocorridas. Suporta as operações de adição, remoção, alteração e movimentação. Além disso, fornece uma interface gráfica para visualização das diferenças e resolução de conflitos. A complexidade de execução e de espaço do algoritmo é quadrática. No pior caso, em que a árvore tem um nível de profundidade grande e existem N nós, então o tempo de execução é $O(N-1)^2$.

Aborda documentos XML gerais e assume que a ordem dos elementos é impor-

tante, isto é, trabalha com o modelo de árvores ordenadas. No entanto, de acordo com a semântica padrão de documentos XML, ele ignora a ordem dos atributos, e com isso trata todas as permutações de atributos como irrelevantes. Esta abordagem pressupõe que existem identificadores únicos para todos os nós presentes no documento base. Com isso, um atributo *mouid* é definido para armazenar o ID único de um elemento. Desta forma todos os nós podem ser casados, ainda que tenham sido submetidos a transformações substanciais.

A estrutura de árvore de versão reduz o uso de memória, de forma que é armazenada apenas uma árvore, a qual contém o documento completo. Todas as alterações são representadas pelos deltas. A árvore de versão suporta um histórico de alterações que permite mesclar somente os nós que foram alterados. Assim, todos os demais nós são ignorados, o que reduz o custo de processamento da árvore. Esta estrutura de árvore versionada foi baseada em um projeto denominado *Fluid* (BOYLAND et al., 2013) , onde um sistema de controle de versão de baixo nível foi desenvolvido utilizando a representação interna do *Fluid IR*. Esta representação é composta por nós, *slots*, atributos e versões.

Os nós são locais de identidade que não contêm nenhuma outra informação. Versões são pontos dentro de um conceito teórico de tempo discreto estruturado em árvore chamada de árvore de versão. As versões são organizadas nesta árvore de forma que a raiz é a versão inicial e versões pais representam estados anteriores ao de seus filhos. Os *slots* são locais que podem armazenar informações, incluindo referências a outros *slots* e nós. Por fim, os atributos são nomes que mapeiam nós a *slots*. Dado um nó e um atributo, pode-se obter o valor de *slot* atribuído a esse nó. Este modelo fornece a ideia de que os nós contém atributos.

Nós, atributos e *slots* podem ser tratados como uma tabela em que as linhas são os nós, as colunas são os atributos e as células são os *slots* que armazenam os valores. Com a adição de versões, torna-se uma tabela tridimensional em que a terceira dimensão é a versão. Com *slots* versionados, dado um nó, um atributo, e uma versão, o valor desta versão pode ser recuperado. Esta estrutura modela a noção de que os atributos de um nó têm valores diferentes, em diferentes versões.

Quando uma nova versão de um documento é criada, ela é representada por uma

tabela vazia de nós, atributos e *slots*, que é a versão atual. Este quadro é preenchido com informações e em algum momento o resultado é salvo como uma versão inicial, v_0. Uma vez salvo, v_0 nunca muda. Outras mudanças na versão atual são permitidas, mas essas mudanças farão parte de uma nova versão. Eventualmente, essas alterações podem ser salvas como v_1. Uma vez que mais de uma versão tenha sido salva no sistema, torna-se possível definir qualquer versão como atual e então começar uma nova versão derivada desta.

Para representar um elemento no documento XML, cada nó recebe um atributo *tagname*, e outro denominado *attributes* que aponta para uma coleção de pares nome-valor que possuem o nome e o valor de um atributo. Desta forma são utilizados dois tipos de atributos (*Fluid* e XML) ao mesmo tempo. Para nós de texto na árvore XML, há um atributo *text* do tipo *Fluid* que contém os dados de texto.

Neste algoritmo, a detecção do nó correspondente e das mudanças ocorridas é feita no momento da análise dos três documentos. O casamento dos nós depende do atributo *mouid* fornecido pelo *Fluid IR*. O documento base é lido pela primeira vez e então uma árvore versionada t_0 é criada e salva como a representação da versão v_0. Uma tabela *hash* é criada mapeando os IDs para elementos em t_0.

Em seguida, o analisador lê o primeiro documento modificado, a fim de construir um delta representando t_1. Cada elemento é examinado observando se o mesmo possui um atributo *mouid*. Caso possua, o elemento é comparado com o elemento correspondente em t_0. Se o elemento tiver sido alterado, então as operações correspondentes são adicionadas ao delta de t_1. Alterações em um nó são registrados em um objeto do tipo *ChangeRecord* durante a análise dos documentos de entrada. Usando um objeto *ChangeRecord*, pode-se verificar se um dado nó mudou em relação a v_1 ou v_2. Se um elemento de t_1 não tem um ID, então ele é um elemento novo, neste caso, uma representação deste elemento é adicionado ao delta de t_1. Quando todo o documento tiver sido processado, o delta de t_1 é salvo como a representação de uma nova versão v_1. O mesmo procedimento ocorre para t_2 gerando um delta salvo como a representação de v_2.

Uma vez que t_0, t_1 e t_2 foram construídos, o processo de mesclagem pode acontecer. Mas antes da fusão, t_3 é criada como uma nova versão do documento. Para mesclar

t_1 e t_2, o algoritmo consulta o *ChangeRecord* de t_1 buscando pelos n nós em t_1 marcados como alterados. O algoritmo LCS é computado sob os IDs dos nós, primeiro para a sequência de filhos de n em t_0 e t_1, em seguida para filhos em t_0 e t_2. Para mesclar as sequências de nós resultantes do LCS, o algoritmo *diff3* é aplicado, resultando em uma nova sequência de filhos para o nó n mesclado em t_3.

A sequência de filhos resultante do *diff3* é manipulada e inserida em t_3 como filhos de n. Este processo analisa cada elemento presente na sequência de mesclagem, e verifica se ele está na sequência de filhos de n em t_3. Os elementos que não estão em t_3 são inseridos. Em seguida, elementos presentes em t_3 mas que não fazem parte da sequência de mesclagem são removidos. Uma vez que todos os filhos dos elementos alterados foram mesclados, t_3 é a árvore resultante da mesclagem de t_1 e t_2. Então, finalmente, os valores dos atributos de n em t_1 e t_2 são fundidos, o que é um processo bem simples, uma vez que o algoritmo desconsidera a ordem dos atributos.

Esta abordagem requer que IDs sejam colocados sobre os elementos do documento de base antes de ser compartilhado. E qualquer elemento nos documentos derivados que não têm um ID deve ser introduzido pelo editor como um novo nó, o qual recebe um novo ID. Assim, o algoritmo não funciona com documentos XML produzidos por ferramentas arbitrárias. Além disso, para a utilização desta abordagem, também é necessário que todos os editores utilizados preservem os IDs durante a edição. A vantagem é que esta abordagem é capaz de representar com precisão mudanças radicais nos elementos, o que é muito difícil nos modelos baseados em *hash*.

3.1.6 Project Merge

O Projetc Merge (OSO, 2013) é uma ferramenta proprietária, mas que se mostrou bastante interessante por calcular as diferenças e mesclar documentos XML de maneira eficiente. Outro fato interessante, é que esta ferramenta possui uma interface para a interação com o usuário, onde os documentos XML são apresentados de forma textual. As versões do documento são sobrepostas de maneira que um único documento é exibido, e todas as alterações estão representadas e destacadas por cores diferentes, que indicam a versão em que ocorreu.

Este software realiza tanto o *2* quanto *3-way merge*, ou seja, pode comparar e mesclar duas ou três versões de um mesmo documento. A cada execução, as operações detectadas pela ferramenta são: inserção, remoção e alteração. Uma das funcionalidades mais interessantes é que quando ocorrem conflitos durante alguma operação, o sistema permite ao usuário escolher qual dos elementos permanecerá.

A ferramenta não possui muitas informações disponíveis sobre os algoritmos de utilizados. Contudo, através da documentação e do tutorial de uso do sistema, é possível observar que a especificação do formato do documento XML é importante para o algoritmo. E ainda, é possível perceber que ele usa o atributo ID presente na DTD do documento para fins de identificação dos elementos. Quando esse atributo não é especificado, o sistema exige que o usuário escolha um atributo, o qual será utilizado com identificador. Esta técnica é chamada de identificação por chave de contexto.

3.1.7 Phoenix

O Phoenix (PINTO and CAMPELLO, 2012) é uma ferramenta que foi desenvolvida com a proposta inicial de apoiar a comparação e detecção de diferenças, exclusivamente de documentos XML, e apresentar os resultados de forma gráfica ao usuário. Uma evolução desta proposta, no que diz respeito à mesclagem foi apresentada por MACHADO (2013).

O Phoenix trata mais especificamente da comparação de documentos XML através de árvores não ordenadas, e considera a similaridade entre os nós para fazer a correspondência entre eles. Para isso, são calculadas as diferenças entre dois documentos XML, adicionando o percentual de similaridade de cada nó envolvido no processo. A similaridade total de cada elemento é calculada em função da similaridade entre os seguintes itens: nomes dos elementos, conteúdo textual, atributos e seus valores e sub-elementos.

Cada item possui um método especial de comparação, que resulta em seu percentual de similaridade. O resultado de cada método é multiplicado por um peso estabelecido pelo usuário e somado aos demais itens, resultando na similaridade total entre os documentos. Caso o usuário não estabeleça o peso, um valor padrão pré-determinado pelo algoritmo é utilizado.

O algoritmo primeiramente verifica se os nomes dos elementos são iguais. Se os

nomes são iguais, os demais itens são comparados normalmente. Caso contrário, o algoritmo entende que os elementos são diferentes. Neste caso, é retornado 0% de similaridade, e não há necessidade de comparar os outros itens.

A segunda comparação é em relação ao conteúdo dos elementos. O algoritmo extrai as informações de ambos os elementos correntes e verifica o percentual de similaridade entre elas. Para o cálculo de similaridade de conteúdo textual, foi utilizado o algoritmo LCS. Como este algoritmo trabalha apenas com *strings*, retornando a maior sequência comum, foi preciso transformar o retorno em um valor percentual. Para obter este valor, o tamanho da *string* encontrada pelo LCS é dividido pela metade do somatório dos tamanhos das *strings* comparadas.

Para calcular a similaridade entre os atributos, o Phoenix busca todos os atributos que compõem os elementos e os compara. Se não existirem atributos em ambos, o método retorna 100% de similaridade. Se existirem vários atributos, o algoritmo extrai todos eles junto com os seus respectivos conteúdos e calcula seu percentual através do LCS. O resultado desta etapa é a média aritmética entre as similaridades de cada atributo.

Por fim, é feita a comparação entre os sub-elementos. O método isola todos os filhos de cada nó e os compara, um a um, até encontrar o conjunto com a maior similaridade. A comparação dos sub-elementos é feita de modo recursivo, onde cada um é comparado utilizando os mesmos métodos para comparação de nome, conteúdo e atributos descritos anteriormente. Nesta etapa, o algoritmo monta uma matriz com todos os sub-elementos do nó esquerdo na primeira linha, e todos os sub-elementos do nó direito na primeira coluna.

A comparação dos elementos pode ser vista como um produto cartesiano, onde cada elemento do nó esquerdo é comparado com cada elemento do nó direito. A cada interação, o resultado é armazenado nessa matriz para que futuramente possa ser percorrida com o objetivo de achar o conjunto de elementos com maior similaridade. O algoritmo utilizado nesta última etapa para processar a matriz é o algoritmo Húngaro (KUHN, 2006). Este método retorna um casamento ótimo dos elementos comparados.

A abordagem mais recente do Phoenix, proposta em MACHADO (2013), segue as mesmas características de similaridade do projeto inicial. Porém foi desenvolvida outra

técnica para realizar a diferenciação entre duas árvores com um ancestral comum. O objetivo desta nova funcionalidade é apresentar ao usuário uma árvore, como anteriormente, porém destacando as mudanças ocorridas entre os documentos e o ancestral comum. Dado o ancestral comum, foram observados alguns possíveis casos de relacionamento, como por exemplo, nós que permaneceram inalterados ou nós adicionados na versão 1 ou 2 do documento. Além disso, pode ocorrer alguma situação de conflito caso algum valor tenha sido modificado e esteja presente em ambos.

Com a utilização do Phoenix nesta nova funcionalidade, surgiram algumas dificuldades em relação ao cálculo da similaridade dos elementos filhos. Isso porque o algoritmo Húngaro utiliza uma matriz onde as linhas e as colunas representam elementos pertencentes a documentos distintos. A adição do ancestral poderia gerar a necessidade de uma estrutura de dados complexa o suficiente para continuar a sustentar a abordagem inicial.

Para resolver este problema, na nova versão, os elementos filhos são comparados separadamente, ou seja, é realizado um cálculo de diferenças recursivo entre cada filho e o ancestral comum. O retorno de cada procedimento é uma árvore contendo os respectivos nós envolvidos da diferenciação. Dadas as árvores de retorno, um processo de mesclagem manual mantendo os nós do ancestral comum é utilizado. O mesmo procedimento de mesclagem manual usado nesta etapa é feito quando o usuário solicita a função de mesclagem da ferramenta.

Durante a mesclagem, o algoritmo considera que se um nó presente em uma das versões novas é idêntico ao seu ancestral comum, não houve a intenção de mudança. De maneira oposta, se o mesmo nó na outra versão foi modificado, então este deverá permanecer na versão mesclada. Caso os dois forem modificados, não há uma heurística de precedência, o algoritmo detecta uma situação de conflito, e exibe uma mensagem de alerta.

3.2 Abordagem proposta

Com o intuito de apoiar a detecção de diferenças e a mesclagem de documentos XML, este trabalho propõe o XChangeMerge, uma abordagem para visualização de diferenças e mesclagem de documentos XML no contexto do projeto XChange. Nesta ferramenta, o

usuário pode carregar a versão base, e duas versões modificadas de um documento XML, e obter uma nova versão mesclada destes documentos, bem como visualizar as diferenças entre as versões.

O XChangeMerge consiste da agregação do XMLTreeMerge (OLIVEIRA, 2007), uma abordagem para visualização de alterações e controle da mesclagem de documentos XML, ao XChange. O XMLTreeMerge é um *plugin* para o SCV *Subversion*. Este *plugin* faz com que o *Subversion* se comporte de forma diferente quando confrontado com documentos XML. Com isso, é possível uma diferenciação e mesclagem destes documentos de forma controlada, por algoritmos específicos para documentos XML, evitando que a mesclagem gere um documento mal formado.

A Figura 3.1 apresenta o diagrama que ilustra o fluxo de execução XChangeMerge. Os dados de entrada consistem na versão base dos documentos XML, e duas versões modificadas. Através do algoritmo 3DM, a ferramenta detecta as diferenças entre as versões, bem como os conflitos gerados por alterações concorrentes. Estas informações são salvas pelo algoritmo no *log* de edição e de conflitos, respectivamente. Além disso, o 3DM fornece uma versão mesclada automaticamente, baseada em soluções pré-definidas para cada tipo de conflito.

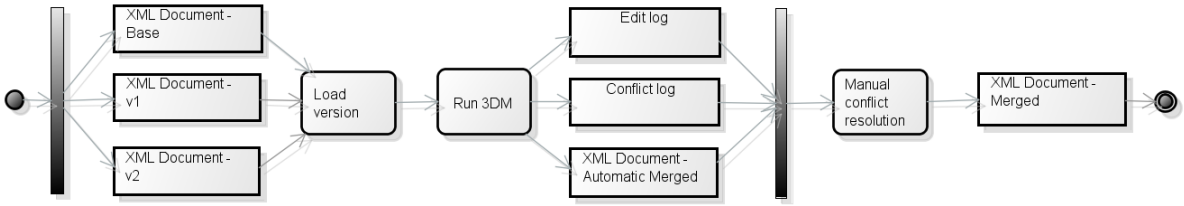

Figura 3.1: Abordagem proposta

O XChangeMerge utiliza os *logs* gerados para exibir os conflitos ao usuário, e permitir que ele resolva-os manualmente. Após a resolução dos conflitos a ferramenta verifica, para cada conflito, se as escolhas feitas pelo usuário são as mesmas tomadas automaticamente. Quando a decisão do usuário é diferente da do algoritmo, o documento que contém a versão mesclada é alterado, de acordo com o que o usuário determinou. Desta forma, a ferramenta gera a nova versão mesclada, que ao final, é exibida ao usuário no formato de árvore.

O algoritmo de diferença e mesclagem utilizado pelo XMLTreeMerge é o 3DM (LINDHOLM, 2001). Este algoritmo, é uma abordagem *3-way* de diferenciação e mesclagem de documentos XML. Como dito anteriormente, dadas três versões de um mesmo documento, uma versão base e duas modificadas, ele retorna a versão mesclada dos documentos, e os *logs* de conflito e diferenças. O 3DM também pode ser utilizado como *2-way*. Neste caso, deve-se reutilizar a base, no lugar de um dos documentos modificados.

O 3DM se baseia em árvores ordenadas, com isso pode detectar operações de inserção, remoção, alteração, movimentação e cópia. Sua execução acontece em duas fases. A primeira é a detecção de alterações. Nesta fase, os deltas entre as versões são gerados pela comparação de cada uma das versões com a versão original. Para isso, é necessário casar os nós da versão original com os nós das versões alteradas. Neste processo de correspondência entre os nós, o 3DM tenta primeiro utilizar os identificadores definidos na DTD do documento. Quando isso é possível, o casamento é feito de forma rápida e eficiente com complexidade $O(n)$, uma vez que para casar todos os nós é preciso passar pela árvore somente uma vez.

Quando o documento não possui IDs definidos na DTD, o algoritmo utiliza uma abordagem heurística composta por uma elaborada função que determina a similaridade entre dois nós. Esta função considera a similaridade entre os conteúdos (no caso de elementos que contêm textos), os nomes dos elementos, atributos, filhos, dentre outras características. Neste caso, o casamento dos nós tem complexidade igual a $O(n^2)$, pois neste caso é preciso calcular a similaridade de cada nó e comparar com os nós da outra árvore, até que um casamento seja encontrado. A complexidade do algoritmo de diferenciação é limitada à complexidade do casamento dos nós $O(n^2)$.

A ideia do casamento heurístico é encontrar a maior subárvore correspondente entre a árvore base T_B e a modificada T_M, e casar os nós de acordo com estas subárvores. O casamento de subárvores é executado através de um algoritmo guloso, que recebe um nó de T_M e a árvore T_B como entradas, e retorna o nó de T_B que é a raiz da maior subárvore correspondente.

Após o casamento inicial, são tomadas duas medidas para melhorar a qualidade desta correspondência entre os nós: casar nós ainda não casados que são semelhantes ou

estão em um contexto semelhante e remover casamentos que envolvem pequenas cópias (colocar um limite na quantidade de dados duplicados para considerar como uma operação de cópia). Por fim, o tipo de correspondência (de conteúdo, estrutural ou completa) é determinado. Desta forma, o algoritmo pode tratar de forma diferenciada cada um dos casos, propagando apenas as mudanças necessárias, gerando resultados mais adequados.

A partir dos casamentos, é possível obter o delta que representa as diferenças entre T_B e T_M. Um nó que não casa com nenhum nó da versão base foi inserido. Um nó que não casa com a versão alterada, foi removido. Nós que casam apenas na estrutura tiveram seu conteúdo alterado, e nós que casam apenas no conteúdo tiveram seu contexto alterado, ou seja, foram movidos ou tiveram filhos retirados.

A segunda fase é a conciliação, onde os casamentos obtidos na fase anterior para as duas versões são utilizados na elaboração da versão final do documento. Começando pela raiz de cada um dos documentos, o algoritmo obtém os nós correspondentes dentre os filhos de cada uma das raízes. Então os dois deltas são aplicados sobre o documento, adequando eventuais movimentações e verificando quais foram alterados, inseridos ou removidos em uma das versões. Quando possível, as mudanças são propagadas de forma que se obtém uma nova lista de filhos, e o procedimento é recursivamente aplicado para os filhos.

O 3DM resolve automaticamente os conflitos detectados durante a mesclagem, através de regras pré-estabelecidas, não permitindo ao usuário decidir qual das alterações conflitantes utilizar. No entanto, o XChangeMerge é uma abordagem que tem como objetivo principal fornecer ao usuário uma visualização das diferenças e dos conflitos. Para isso, esta ferramenta recorre ao 3DM, utilizando a versão mesclada e os *logs* gerados para apresentar as alterações ao usuário.

Estas alterações são apresentadas a partir da visualização de três árvores, uma com a versão base e as outras com as duas versões modificadas. Além disso, outra opção da ferramenta é permitir ao usuário o tratamento dos conflitos. Isso é feito através de uma lista de conflitos que é exibida na tela de forma que, para cada conflito selecionado, o usuário possa escolher qual versão ele deseja usar. Após tratar os conflitos, o usuário pode salvar a sua versão mesclada, que é apresentada em outra tela.

A aplicação proposta tem um importante papel dentro do XChange, uma vez que através deste novo módulo além de comparar e mesclar diferentes versões de um documento XML, o usuário pode visualizar as diferenças entre as versões, e também resolver manualmente eventuais conflitos ocorridos durante a mesclagem. Outra questão importante desta abordagem é a maneira como os documentos são exibidos para o usuário. No XChangeMerge, além da visualização em formato de texto, já contemplada pelo XChange, o usuário agora também se beneficia da visualização em formato de árvore.

3.3 Análise das propostas

3.3.1 Relacionamento entre as abordagens

A Figura 3.2 mostra o relacionamento entre as abordagens apresentadas neste capitulo.

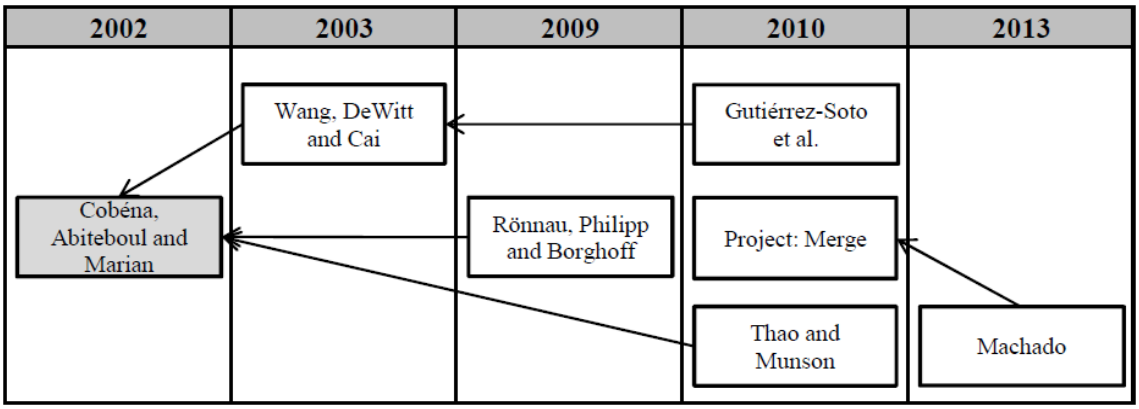

Figura 3.2: Relacionamento entre as abordagens

Como apresentado na Figura 3.2, a abordagem de COBENA et al. (2002), baseada no conceito de chaves e árvores ordenadas para detecção de diferenças em documentos XML é o mais referenciado, dentre os selecionados. Ele serviu como base para que WANG et al. (2003) apresentassem um algoritmo com o mesmo objetivo. Entretanto, WANG et al. (2003) trabalham com o conceito de árvores não ordenadas, a fim de obter resultados mais precisos que o algoritmo proposto por COBENA et al. (2002).

RÖNNAU et al. (2009) e THAO and MUNSON (2010) também se basearam no trabalho de COBENA et al. (2002). Ambos trabalham com árvores ordenadas. No entanto, RÖNNAU et al. (2009), no intuito de melhorar a eficiência do algoritmo na

transformação dos documentos, e a qualidade dos deltas gerados, introduziu um novo modelo de delta usando impressões digitais de contexto calculadas através do algoritmo LCS . Por outro lado THAO and MUNSON (2010) também trabalha com o conceito de chaves, porém as chaves utilizadas nesta abordagem não são os atributos IDs definidos na DTD, como em COBENA et al. (2002). Neste caso, as chaves são definidas pela própria ferramenta, utilizando uma estrutura de dados de árvores especializadas em controle de versão.

A abordagem proposta por GUTIÉRREZ-SOTO et al. (2010) se baseou em WANG et al. (2003), e também optou por utilizar uma função de *hash* na identificação dos nós. A maneira como o algoritmo utiliza o valor do *hashing* é o que o difere da abordagem anterior. Outra diferença desta abordagem, é que ela trata tanto árvores ordenadas como não ordenadas.

Por fim, MACHADO (2013), que é a abordagem mais recente, se baseou na ferramenta de OSO (2013) no que diz respeito à possibilidade de mesclagem dos documentos, bem como à resolução manual de conflitos.

3.3.2 Comparativo entre as abordagens

Esta seção tem por objetivo apresentar um comparativo entre abordagens supracitadas. Para isso, a Tabela 3.1 traz em suas linhas cada um dos algoritmos estudados, e em suas colunas, algumas questões chave para a comparação entre eles.

Tabela 3.1: Comparativo entre as abordagens

Nome	Complex.	Match	Árvore	Operações	Merge	Interface
XyDiff	linear	ID DTD	ordenada	I, R, A, M	não	não
XDiff	quadrática	XHash	não ordenada	I, R, A	não	não
DocTreeDiff	linear	Hash + LCS	ordenada	I, R, A, M	apoio	não
CDL	linear	Hash	ambas	I, R, A	não	não
Molhado SPL	quadrática	ID árvore	ordenada	I, R, A, M	sim	sim
Project Merge	não informada	ID DTD ou Chave de contexto	-	I, R, A	sim	sim
Phoenix	não informada	Similaridade	não ordenada	I, R, A	sim	sim
XChangeMerge	quadrática	Similaridade	ordenada	I, R, A, M, C	sim	sim

Pode-se então observar que os algoritmos mais eficientes em termos de complexidade são XyDiff, DocTreeDiff e CDL. No caso do XyDiff, esta eficiência é devido ao uso dos IDs já definidos na DTD do documento, o que nem sempre é possível. Já nos

casos do DocTreeDiff e CDL, observa-se que o uso de funções de *hash* para o cálculo dos identificadores para os elementos também é uma técnica eficiente.

Todos os algoritmos apresentados suportam as operações básicas de inserção (I), remoção (R) e alteração (A). Entretanto, apenas as abordagens XyDiff, DocTreeDiff, Molhado SPL e XChangeMerge suportam a operação de movimentação (M). Isso acontece devido ao fato de que estas abordagens trabalham com árvores ordenadas, o que é essencial na identificação de movimentações. O XChangeMerge é o mais completo dentre os apresentados, uma vez que suporta também a operação de cópia (C).

O ponto mais importante desta comparação, no contexto deste trabalho, é a mesclagem. Analisando a Tabela 3.1, pode-se verificar que apenas as abordagens Molhado SPL, Project Merge, Phoenix e XChangeMerge possuem a função de mesclagem dos documentos. O DocTreeDiff, apresenta apenas um apoio à mesclagem, isto é, o delta gerado pelo algoritmo de diferença possui as informações necessárias para a mesclagem.

A Tabela 3.2 apresenta um comparativo mais detalhado, apenas das abordagens com função de mesclagem. Desta forma é possível observar que todas as abordagens possuem a opção de edição dos documentos. No entanto, apenas o Project Merge, o Phoenix e o XChangeMerge possuem uma interface de visualização das diferenças entre os documentos. Outro ponto importante é a resolução dos conflitos. As abordagens Molhado SPL, Project Merge, Phoenix e XChangeMerge suportam a resolução manual dos conflitos pelo usuário.

Tabela 3.2: Comparativo entre as abordagens de mesclagem

Nome	Visual. diff	Tipo de visual.	Resolução de conflitos	Edição do docum.
Molhado SPL	não	árvore	manual	sim
Project Merge	sim	texto	manual	sim
Phoenix	sim	árvore	manual	sim
XChangeMerge	sim	árvore	manual e automática	sim

Além disso, pode-se observar que das abordagens apresentadas na Tabela 3.2, apenas o Project Merge segue o formato de visualização textual. Isso sugere uma preferência dos autores pela visualização em árvores.

3.3.3 Análise prática das abordagens

A fim de obter informações práticas sobre o uso das ferramentas descritas, buscou-se encontrar versões das mesmas, para que fossem testadas de forma que alguns requisitos pudessem der observados. A ferramenta Project Merge possui uma versão *trial* disponível por 30 dias para que possa ser testada. Já o Phoenix, não está disponível, entretanto, em contato com os autores, foi possível obter uma versão para testes. Desta forma, este experimento foi efetuado com o Project Merge, o Phoenix e com XChangeMerge, uma vez que as demais ferramentas não estão disponíveis.

A partir dos testes realizados, foram observadas as funcionalidades de cada ferramenta e suas formas de interação com o usuário durante a detecção de diferenças, resolução de conflitos e mesclagem de documentos.

Verificou-se que o Project Merge é a única ferramenta que necessita de informações sobre a estrutura dos documentos para efetuar a mesclagem. Isso acontece, porque esta abordagem utiliza uma chave de contexto definida pelo usuário para a identificação e casamento dos elementos. As demais abordagens, por serem baseadas em funções de similaridade, dispensam a definição deste atributo.

No que diz respeito ao modelo de visualização dos documentos, foi visto na seção anterior, que o XChangeMerge e o Phoenix utilizam o formato de árvores, e o Project Merge utiliza o formato textual. Contudo, é possível observar ainda, algumas particularidades de cada abordagem.

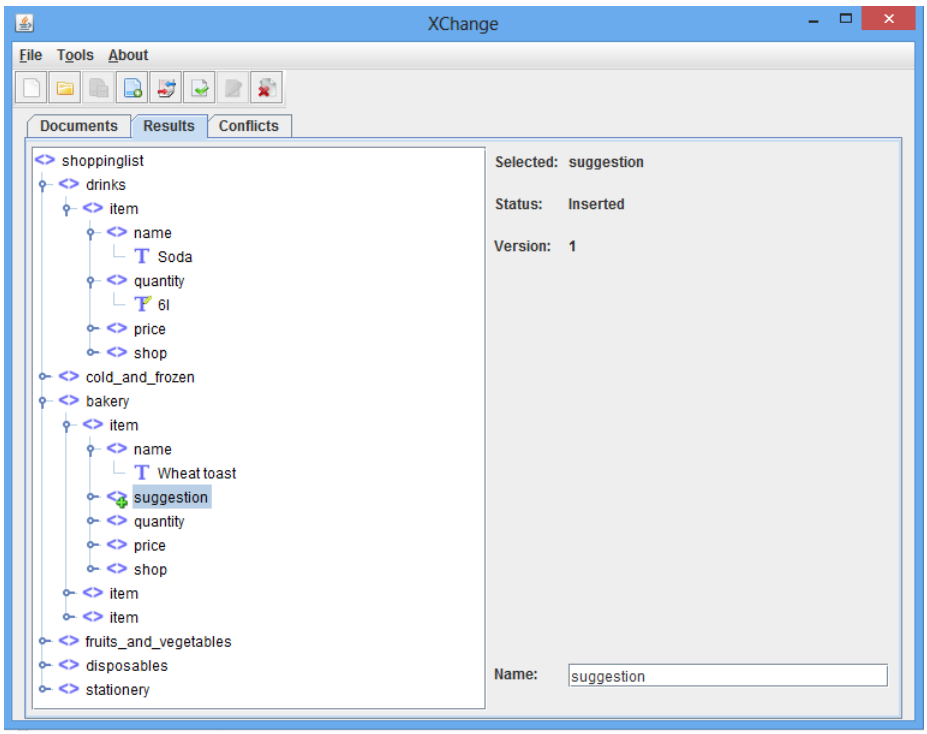

Figura 3.3: Exibição de diferenças no XChangeMerge

No XChangeMerge, a versão base do documento e as duas versões modificadas podem ser visualizadas de forma independente durante a resolução de conflitos. Após a mesclagem, elas são exibidas agrupadas em uma única árvore, que contém todos os elementos das duas versões. Diferentes ícones são utilizados para destacar os nós inseridos, removidos, alterados, movidos ou copiados. A Figura 3.3 apresenta a interface de exibição de diferenças do XChangeMerge.

O Phoenix, também fornece uma visualização independente das duas versões modificadas do documento, mas não da versão base. Na árvore mesclada, os elementos são destacados por cores, de forma que cada cor representa a versão a qual o elemento pertence, versão 1, versão 2, ambos (permaneceu inalterado em ambas as versões) ou conflito. O Phoenix não fornece informações sobre as operações de edição ocorridas. A interface de exibição de diferença do Phoenix é ilustrada pela Figura 3.4.

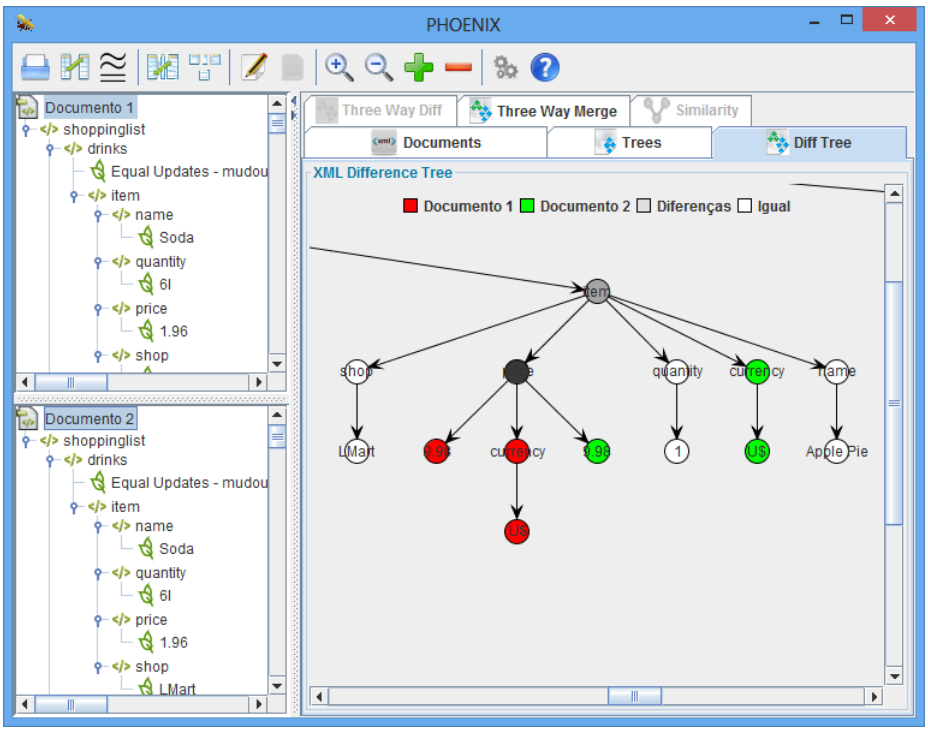

Figura 3.4: Exibição de diferenças no Phoenix

O modelo de visualização do Project Merge, embora textual, também exibe um único documento, que agrupa todas as versões. Desta forma, o documento é exibido como um texto comum, para os elementos que não sofreram alterações. Para elementos alterados, a ferramenta exibe as três versões, uma seguida da outra, destacando por cores diferentes, as informações de cada versão. A igura 3.5 ilustra a interface de exibição de diferenças e resolução de conflitos do Project Merge.

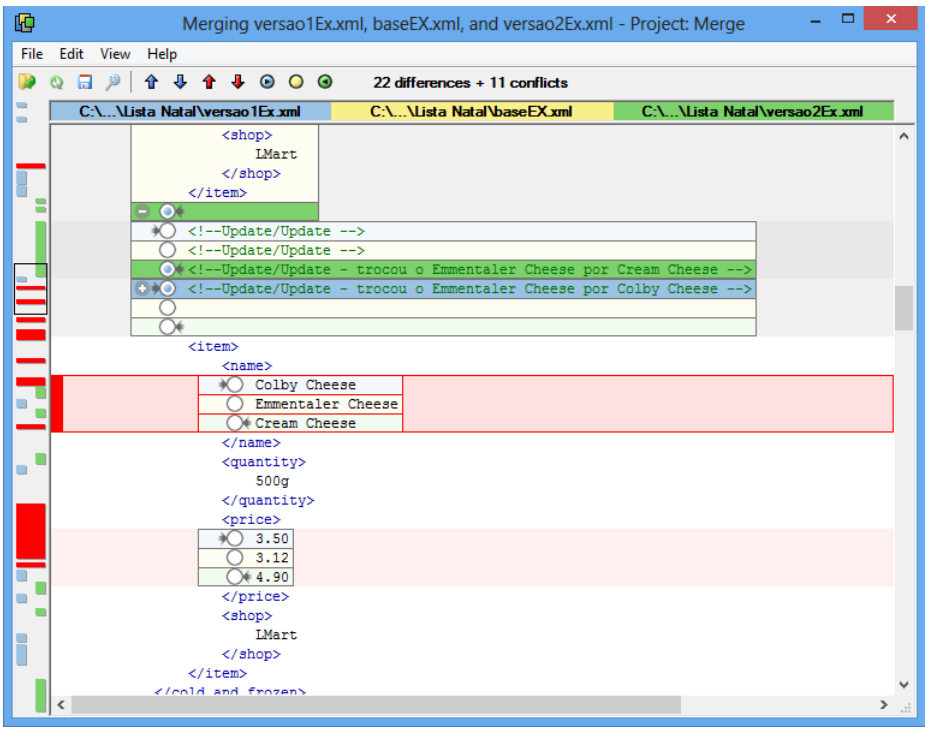

Figura 3.5: Exibição de diferenças e resolução de conflitos no Project Merge

Todas as abordagens testadas oferecem a visualização completa das diferenças, destacando todos os elementos que sofreram alterações, independente de haver conflitos ou não. Além disso, todas elas são capazes de mostrar os elementos modificados que geraram conflitos. O XChangeMerge, além de destacar os conflitos na árvore, exibe uma lista de conflitos e avisos de conflitos, classificados pelo tipo de operação.

Com relação à resolução de conflitos, o Phoenix não possui uma maneira automatizada de se realizar esta tarefa. Esta abordagem apenas identifica a ocorrência de conflitos, destaca os nós conflitantes na árvore, e exibe uma mensagem solicitando que o usuário resolva os conflitos no documento original. Desta forma, o usuário precisa selecionar a aba que exibe os documentos completos, encontrar o elemento conflitante, e alterá-lo em uma das versões, de maneira que fique igual em ambas. O XChangeMerge e o Project Merge permitem que ao selecionar um conflito o usuário escolha, através de um *radio button*, qual versão deseja manter no documento mesclado.

No XChangeMerge, os conflitos podem ser resolvidos automaticamente por regras pré-definidas de acordo com o tipo de conflito. Portanto, nesta abordagem, o usuário pode salvar a versão mesclada sem ter que resolver todos os conflitos, deixando que os demais

conflitos tenham a solução padrão.

A resolução automática de conflitos não se aplica ao Project Merge. Nesta ferramenta a versão mesclada dos documentos não pode ser salva antes que usuário resolva manualmente todos os conflitos. Entretanto, uma questão interessante do Project Merge, é que ele permite também a mesclagem de comentários, inclusive detecta conflitos em comentários permitindo a escolha do usuário. As demais abordagens não mesclam comentários.

O Project Merge permite que o usuário escolha por manter a versão base do elemento, mesmo quando não ocorreu nenhum conflito. Já o XChangeMerge entende que não faz sentido manter o versão base quando existem versões novas. Por outro lado, para auxiliar o usuário na resolução dos conflitos, o XChangeMerge classifica cada um deles de forma que ao selecionar um conflito na lista, o sistema exibe uma descrição das operações que geraram os conflitos. Além disso, informações sobre o resultado obtido com escolha de cada versão, são exibidas ao usuário abaixo do botão de seleção das versões. Estas informações auxiliam o usuário na resolução dos conflitos, uma vez que mostram claramente o que gerou o conflito, e o que acontece caso ele escolha manter a primeira ou a segunda versão.

Ainda com relação à resolução de conflitos pode-se observar que o Project Merge possui um contador de conflitos, que indica inicialmente a quantidade total de conflitos, e é decrementado a cada conflito resolvido. Além do contador, esta ferramenta possui botões de navegação entre os conflitos, permitindo que o usuário selecione o conflito anterior ou o próximo. Estas funcionalidades melhoram significativamente a usabilidade da ferramenta, desta forma, seria interessantes a implementação destas funcionalidades no XChangeMerge.

No XChangeMerge, a navegação entre os conflitos é feita somente através da lista de conflitos. Esta lista é numerada, permitindo que o usuário saiba o número total de conflitos e avisos de conflitos. Entretanto, os conflitos não são removidos desta lista após serem resolvidos, uma vez que o usuário pode selecioná-lo novamente, caso queira mudar a escolha feita.

A visualização da versão mesclada, tanto no XChangeMerge quanto no Phoenix

é feita através de árvores. A árvore apresentada pelo XChangeMerge contém ainda informações sobre as mudanças ocorridas. Além disso, esta abordagem permite a edição do documento mesclado. No Phoenix, apenas as versões anteriores podem ser alteradas. Já o Project Merge não permite a edição de nenhuma das versões, e também não exibe a versão final, podendo esta ser salva e visualizada em outra ferramenta.

3.4 Conclusão

Neste capítulo foi possível analisar alguns trabalhos presentes na literatura que implementaram algoritmos de diferenciação e mesclagem de versões diferentes de um documento XML. Pode-se verificar a existência de diversas técnicas desenvolvidas para este fim, entretanto, a maioria das abordagens ou não possuem a funcionalidade de mesclagem, ou não possuem uma representação visual adequada que permita a interação do usuário.

4 XChangeMerge: um exemplo de utilização

Após apresentar os principais conceitos relativos ao gerenciamento de mudanças em documentos XML, bem como os principais trabalhos relacionados a esta atividade, este capitulo tem por finalidade explicar detalhadamente a abordagem proposta neste trabalho. Para isso, a seção 4.1 apresenta uma breve descrição da ferramenta XChange, a seção 4.2 descreve as novas funcionalidades implementadas no XChangeMerge, e finalmente a seção 4.3 traz um exemplo de utilização da ferramenta.

4.1 XChange

O XChange é um sistema que de um modo geral visa a compreensão de mudanças ocorridas em diferentes versões de um documento XML. Esta abordagem utiliza o conceito de *diff* semântico onde, a partir do processamento de duas versões de um documento, é possível compreender as mudanças ocorridas entre as versões, bem como inferir um sentido para as mesmas. A inferência é feita através de um motor de inferência Prolog que aplica um conjunto de regras, escritas pelo usuário, aos fatos derivados das versões do documento (GAZZOLA, 2011).

Em GARCIA (2012) foi proposto um módulo de construção de regras para a inferência em documentos XML. Este trabalho teve como objetivo principal tornar a criação da regra-base um processo semi automático e fazer com que a construção destas regras seja uma tarefa mais simples, sem a necessidade de conhecer a linguagem Prolog.

A Figura 4.1 ilustra o processo de inferência de dados, através do XChange dividido em três etapas. A primeira etapa é a construção da base de informações, formada por fatos Prolog. Nesta etapa, o usuário deve selecionar duas versões de um documento XML, um arquivo base e um modificado, sobre os quais se deseja inferir as mudanças semânticas. Para a construção dos fatos, a ferramenta submete os documentos à uma biblioteca, que transforma elementos em predicados e o conteúdo em constantes (MARTINS et al., 2013).

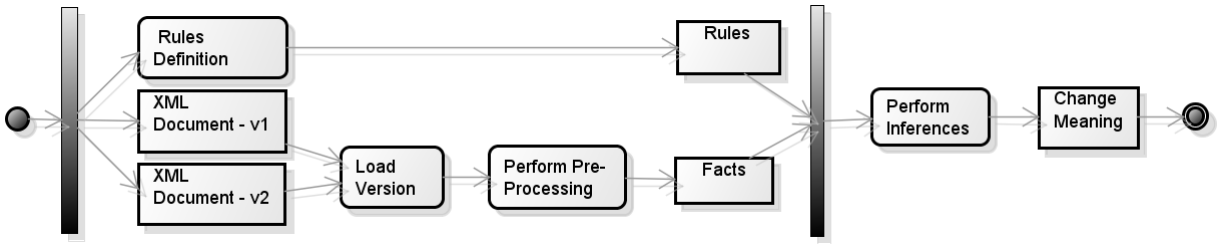

Figura 4.1: XChange (MARTINS et al., 2013)

A segunda etapa consiste da análise do contexto de negócio e construção de regras de inferência. Estas regras estabelecem o que pode ser inferido dos documentos escolhidos. Nesta etapa, o usuário deve carregar o documento contendo as regras, ou criá-las através da interface de construção de regas (MARTINS et al., 2013).

A última etapa consiste em aplicar as regras criadas anteriormente na base de fatos fornecida pelo tradutor de XML em Prolog. Esta etapa utiliza a biblioteca tuProlog (DENTI et al., 2001), para criar um motor de inferência. A inicialização deste motor de inferência é feita, atribuindo a este, a teoria composta pelos fatos e pelo conjunto de regras. A consulta é realizada através dos nomes das regras selecionadas. Em cada consulta é gerado um conjunto de respostas que atendem a condição presente na regra. Finalmente estas repostas são apresentadas ao usuário no formato de texto, contendo o nome dado à regra e os elementos que as atendem (MARTINS et al., 2013).

Na abordagem apresentada, a inferência é feita através de um atributo chave do contexto de negócio que o documento XML representa. A Chave de contexto deve ser um atributo do documento que identifica o elemento em todas as instâncias do mesmo, independente da versão. Portanto, espera-se que o usuário, conhecedor do domínio de negócio em questão, escolha um atributo que possa ser utilizado para este fim.

Em OLIVEIRA (2013), uma nova forma de identificação dos elementos foi proposta. Esta abordagem utiliza uma função de similaridade para identificação dos elementos nas diferentes versões. Com isso, é possível verificar a equivalência entre os elementos com maior precisão, sem a necessidade do usuário definir um atributo de chave de contexto.

Desta forma, o XChange é composto por duas abordagens *"Context Key"* e *"Similarity"*. Em ambas, é possível atribuir significados semânticos às mudanças sintáticas

encontradas entre duas versões de um documento XML.

4.2 XChangeMerge

O XChangeMerge, tem como principal objetivo a mesclagem *3-way* de documentos XML, mas também pode ser utilizado para a mesclagem *2-way*, ou para verificar as diferenças entre as versões do documento. Estas funcionalidades são executadas a partir da tela de mesclagem ilustrada na Figura 4.2, e são compostas basicamente pelos seguintes passos:

1. abrir a versão base do documento;
2. abrir a primeira versão modificada do documento;
3. abrir a segunda versão do documento (caso o usuário deseje a modalidade *2-way*, basta carregar a versão base novamente);
4. clicar no botão *merge*.

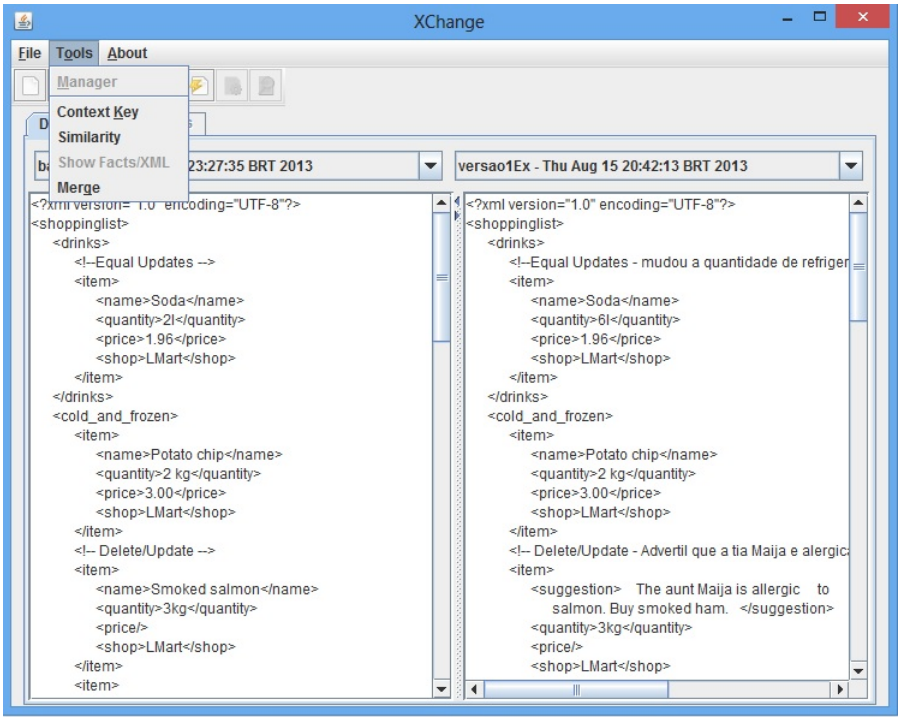

Figura 4.2: XChange

Com a execução dos passos descritos, a ferramenta analisa os documentos através do algoritmo 3DM, e gera os *logs* de edição e de conflitos, bem como a versão mesclada dos

documentos. Esta versão mesclada é gerada automaticamente pelo algoritmo mesmo que ocorram conflitos durante a mesclagem. Para isso, um conjunto de regras pré-definidas decidem para cada tipo de conflito gerado, qual das versões deve ser mantida após a mesclagem. Os arquivos resultantes do 3DM são utilizados na composição das telas de exibição dos conflitos e das diferenças entre os documentos.

Ao introduzir esta nova funcionalidade, a interface do XChange foi modificada, acrescentando uma nova opção no menu *tools*. Desta forma, ao selecionar esta opção, o usuário é direcionado a uma nova tela implementada para esta abordagem. A interface inicial do XChange após carregar os documentos é apresentada na Figura 4.2, onde pode-se observar no menu *tools* as opções *Context Key*, *Similarity* e *Merge*.

A interface do XChangeMerge é composta por três abas. A primeira, chamada de *documents* é exibida na Figura 4.3. É nesta aba que o usuário carrega os documentos para a mesclagem, visualizando-os em forma de texto. Esta aba está sempre habilitada dentro do módulo de mesclagem.

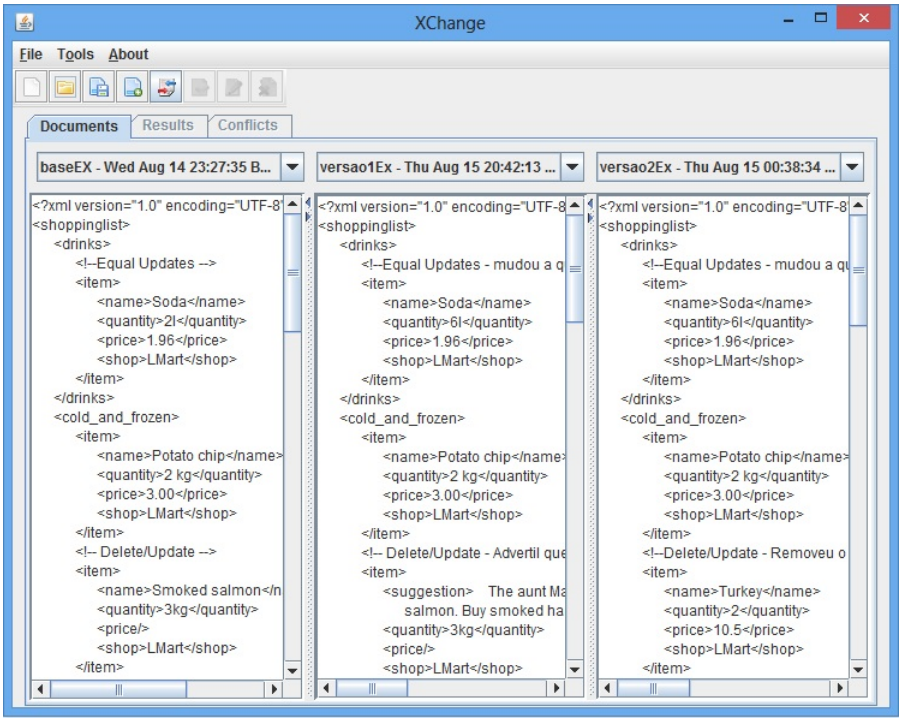

Figura 4.3: Mesclagem de documentos

A segunda aba, chamada de *results*, exibe a versão mesclada dos documentos. Nesta aba, exemplificada na Figura 4.4, o documento é apresentado no formato de árvore e os elementos que sofreram alterações são marcados por ícones que identificam a operação

de mudança ocorrida. A Tabela 4.1 descreve o significado de cada ícone.

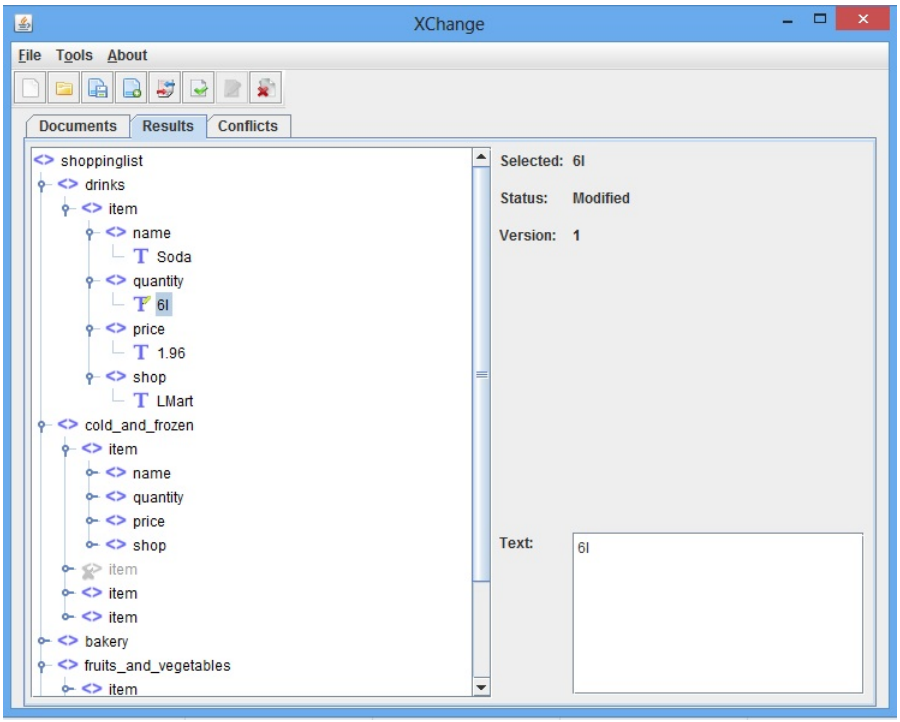

Figura 4.4: Árvore de resultados da mesclagem

Tabela 4.1: Descrição dos ícones da árvore de diferenças

Operação	Elemento	Texto	Atributo
-	<>	T	@
Insert	<>	T	-
Delete	<>	T	-
Near Move	<>	T	-
Far Move	<>	T	-
Update	<>	T	-
Copy	<>	T	-

Durante a exibição das alterações, também é permitido que o usuário faça alterações na versão final do documento. Ao clicar em um elemento da árvore, caso o nó tenha sofrido alguma alteração, é possível verificar o nome da operação, bem como a versão na qual a mudança ocorreu. O usuário também pode editar o nome e o texto dos elementos. Clicando com o botão direito do mouse, o usuário tem acesso a outras opções como remover ou inserir um elemento e cancelar uma remoção. Além disso, arrastando os nós, o usuário pode mover os elementos dentro da árvore.

A árvore exibida na segunda aba, a princípio é o resultado da mesclagem automática do 3DM. Entretanto, caso tenha ocorrido algum conflito, o usuário pode resolver manualmente os conflitos. Com isso, esta árvore é atualizada toda vez que o usuário altera a solução de algum conflito e clica no botão *Apply Choice*.

Desta forma, a terceira aba, chamada de *conflicts*, somente é habilitada se ocorrer algum conflito durante a mesclagem. Nesta tela, exibida na Figura 4.5, as versões do documento são apresentadas lado a lado no formato de árvore, e os conflitos gerados são exibidos em uma lista. Ao selecionar um conflito, os elementos envolvidos são destacados na árvore e uma descrição do conflito é exibida. Além disso, um *radio button* é exibido permitindo ao usuário escolher qual das versões será mantida.

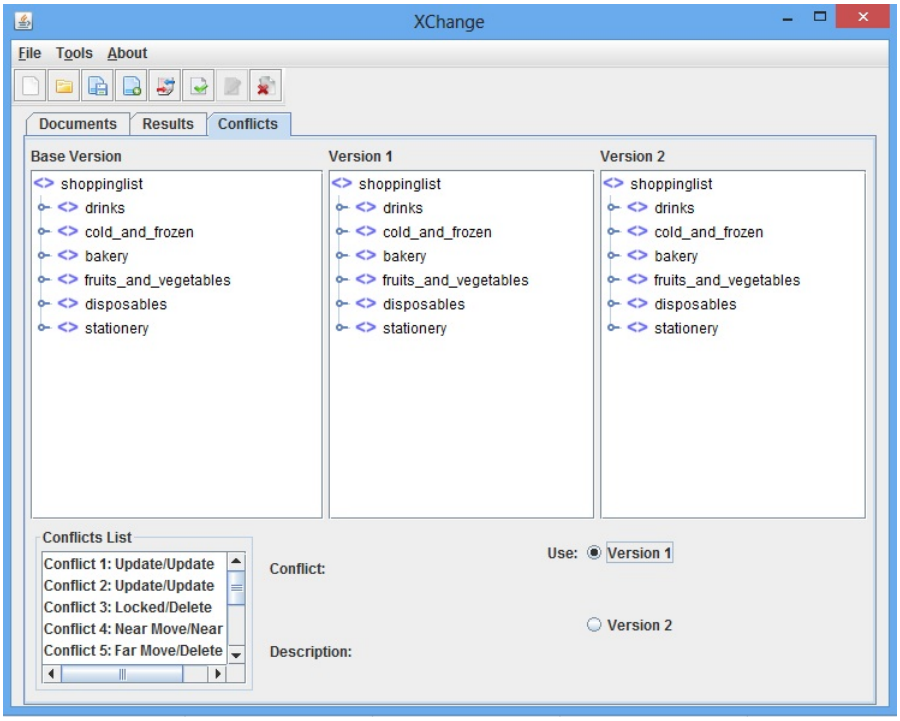

Figura 4.5: Resolução de conflitos

4.3 Exemplo de utilização - Shopping List

Para exemplificar o funcionamento da ferramenta, a seguir é apresentado um exemplo de utilização que simula uma lista de compras compartilhada. A lista consiste em um documento XML onde cada item é um elemento representado pela tag $< item >$, e as informações sobre o item são representadas por sub-elementos como por exemplo a *tag*

$< name >$. Este exemplo foi proposto por LINDHOLM (2001) e adaptado para ilustrar o presente trabalho.

Um sistema inteligente gerencia esta lista de forma que cada usuário possui em seu dispositivo (computador, *tablet* ou celular) a lista completa dos itens a serem adquiridos. Os usuários podem adicionar ou remover itens, bem como fazer comentários. O sistema deve ser capaz de sincronizar a lista de todos os dispositivos, mesclando as alterações.

Tomando como exemplo a lista de natal da família Smith, observa-se que a Sra. e o Sr. Smith, empenhados na organização da ceia de natal, fizeram diversas alterações na lista ao longo do dia. A Sra. Smith, como também estava envolvida na preparação da casa, utilizou o seu *tablet* para inserir os itens na lista. Paralelamente, seu esposo, que ainda estava no trabalho, registrou suas alterações através do seu *notebook*.

Para facilitar as compras, eles optaram por comprar todos os itens em um mesmo supermercado, e criaram uma lista já subdividida em categorias, de forma que cada categoria da lista representa a seção do supermercado onde o item se encontra. A versão base utilizada por eles está na listagem L0.xml, enquanto as versões da Sra. e do Sr. Smith estão respectivamente nas listagens L1.xml e L2.xml, todas no Apêndice.

Ao final da tarde, o Sr. Smith, encarregado de fazer as compras, precisava imprimir a lista. Entretanto, para garantir que a versão impressa seria a mais recente, antes disso, foi necessário sincronizar a lista com o repositório. Ao sincronizar, ele percebeu que sua esposa também havia alterado a lista, e que algumas alterações geraram conflitos.

A fim de obter o melhor resultado possível, ele preferiu resolver manualmente cada um dos conflitos usando o XChangeMerge. Para isso, ele abriu o XChange e selecionou a opção *Merge* no menu *tools*. Em seguida, carregou o documento base utilizado por ele e por sua esposa, e cada uma das novas versões. Ao clicar no botão *Merge*, ele foi direcionado à aba de conflitos, onde pode visualizar todos os conflitos ocorridos.

A seguir os conflitos são listados juntamente com uma breve descrição de suas causas. Também são apresentadas a solução automática dada pelo XChangeMerge, e a solução escolhida pelo Sr. Smith.

O primeiro conflito listado é o *Update/Update*. Este conflito ocorreu pois a Sra. Smith resolveu servir *Colby Cheese* ao invés de *Emmentaler Cheese*, mas seu marido

também quis trocar o tipo de queijo, alterando este mesmo item para *Cream Cheese*. Observe que este conflito aparece duas vezes na lista, pois além do nome, o preço do queijo também foi alterado em ambas as versões. Ao resolver este conflito, o Sr. Smith preferiu não contrariar sua esposa, e escolheu utilizar a versão 1 do documento. A mesma escolha foi feita para o conflito nos preços, já que não faz sentido comprar um queijo pelo preço de outro. Para este tipo de conflito, a solução automática é sempre utilizar a alteração da versão 1, portanto, este resultado foi mantido. A Figura 4.6 ilustra o conflito *Update/Update*.

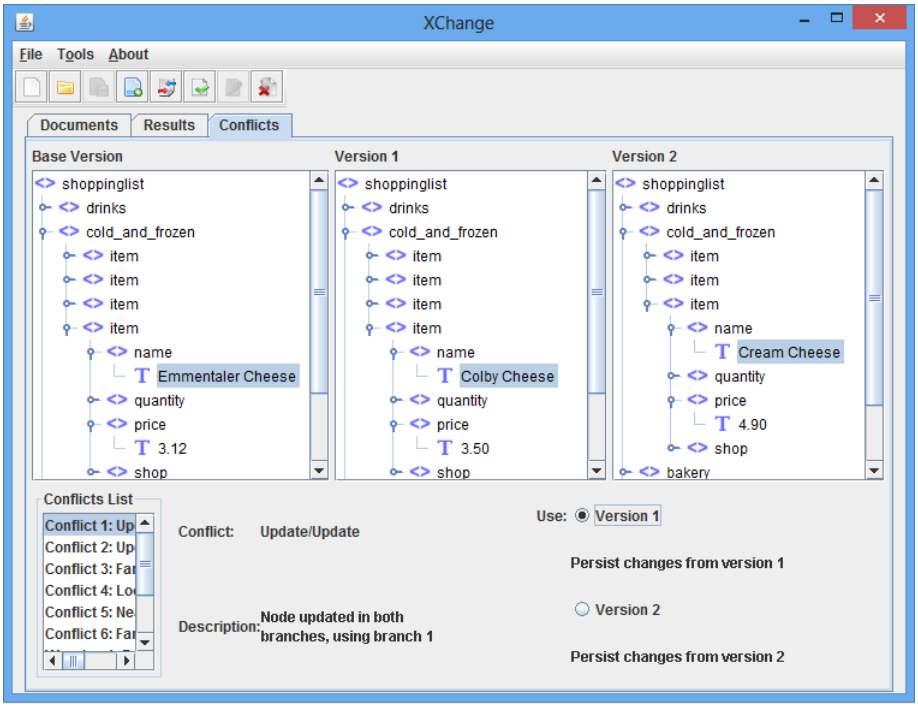

Figura 4.6: Resolução do conflito Update/Update

O conflito *Far Move/Near Move* apresentado na Figura 4.7, ocorreu devido uma diferença de formatação entre as duas listas. Na versão da Sra. Smith a ordem entre o elemento *currency* e o valor dentro do *price* foi alterada, enquanto na outra versão, o elemento *currency* foi movido para fora do *price*. Como este tipo de formatação não interfere nas compras, o Sr. Smith deixou que este conflito fosse resolvido automaticamente. Por padrão o XChangeMerge mantém a movimentação externa e ignora a movimentação dentro do mesmo pai, logo, a segunda versão foi mantida.

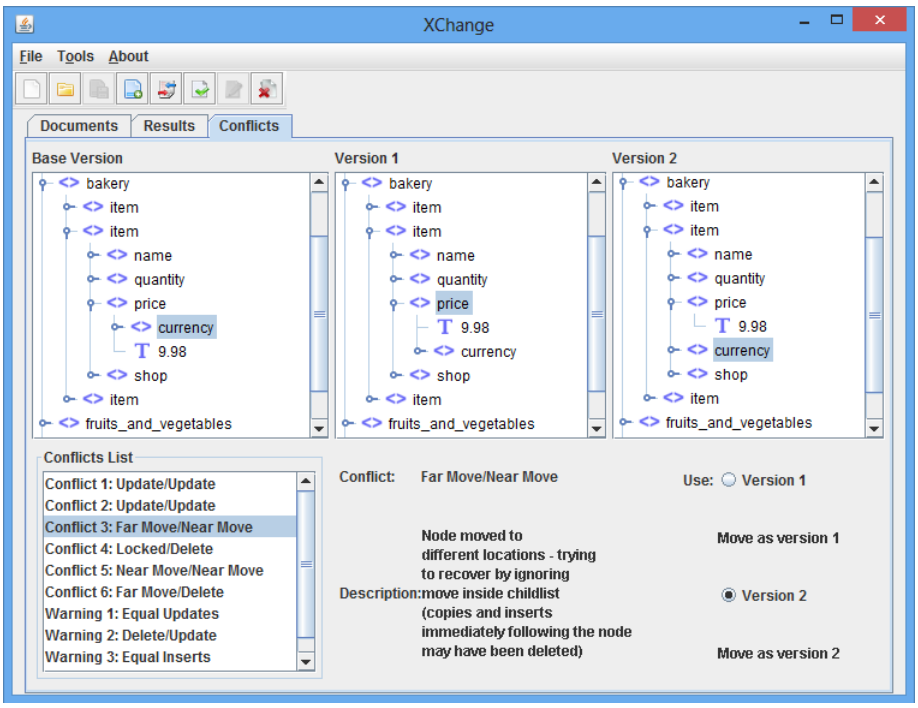

Figura 4.7: Resolução do conflito Far Move/Near Move

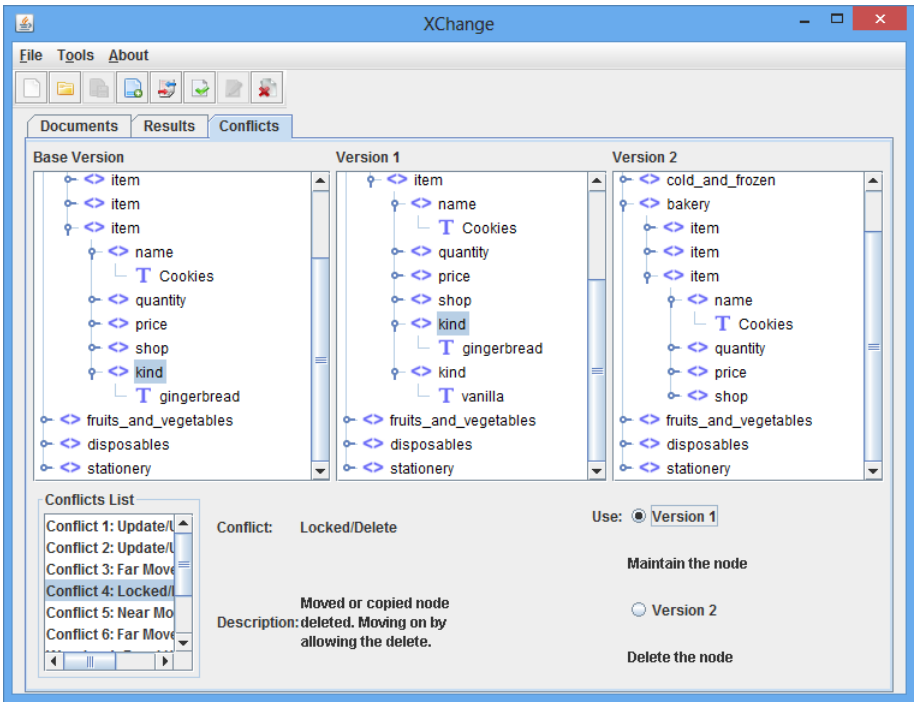

Figura 4.8: Resolução do conflito Locked/Delete

O próximo conflito da lista, *Locked/Delete*, aconteceu pois o Sr. Smith, que não gosta dos biscoitos de gengibre, removeu da sua lista o elemento *kind* que sugeria a compra destes biscoitos. Porém, a Sra. Smith já sabendo da preferência do seu marido

pelos biscoitos de baunilha, inseriu outro elemento *kind* em sua lista, sugerindo que estes também fossem comprados. O Sr. Smith satisfeito com a sugestão de sua esposa, preferiu manter os dois tipos de biscoitos na lista. Desta forma, diferente da solução padrão do XChangeMerge, ele optou por não remover o elemento *kind*, escolhendo a versão 1, conforme mostra a Figura 4.8.

O quinto conflito registrado, é o *Near Move/Near Move*, e assim como o *Far Move/Near Move* foi gerado por uma diferença de formatação. O elemento *quantity* foi movido nas duas versões para diferentes posições dentro do mesmo pai. Novamente, o Sr. Smith deixou que o sistema resolvesse o conflito automaticamente. Neste caso, o XChangeMerge opta simplesmente por utilizar a versão 1. Desta forma a versão mantida foi a da Sra. Smith. A Figura 4.9 ilustra este conflito.

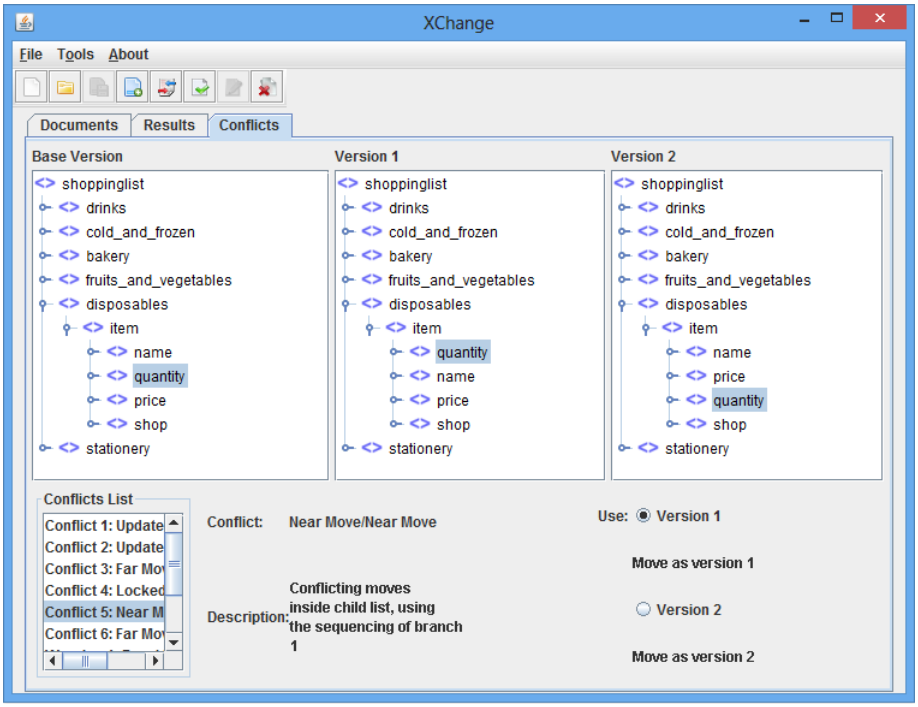

Figura 4.9: Resolução do conflito Near Move/Near Move

O último conflito da lista é o *Far Move/Delete* ilustrado pela Figura 4.10. Este conflito ocorreu pois a Sra. Smith temendo que seu marido não visse sua sugestão, moveu o elemento *types*, o qual contém os tipos de cartões que ela deseja, para fora do elemento *quantity*. Mas o Sr. Smith não gosta de cartões musicais, e também acha que não seria interessante distribuir cartões iguais para todos, então ele preferiu escolher os cartões na hora da compra, e removeu ambas as sugestões de sua esposa . Durante a mesclagem dos

documentos, ele manteve sua posição, e escolheu usar a versão 2, portanto o elemento *types* foi removido. Sua escolha foi diferente da solução automática, uma vez que neste caso, o 3XChangeMerge ignora a remoção, e usa sempre a outra versão.

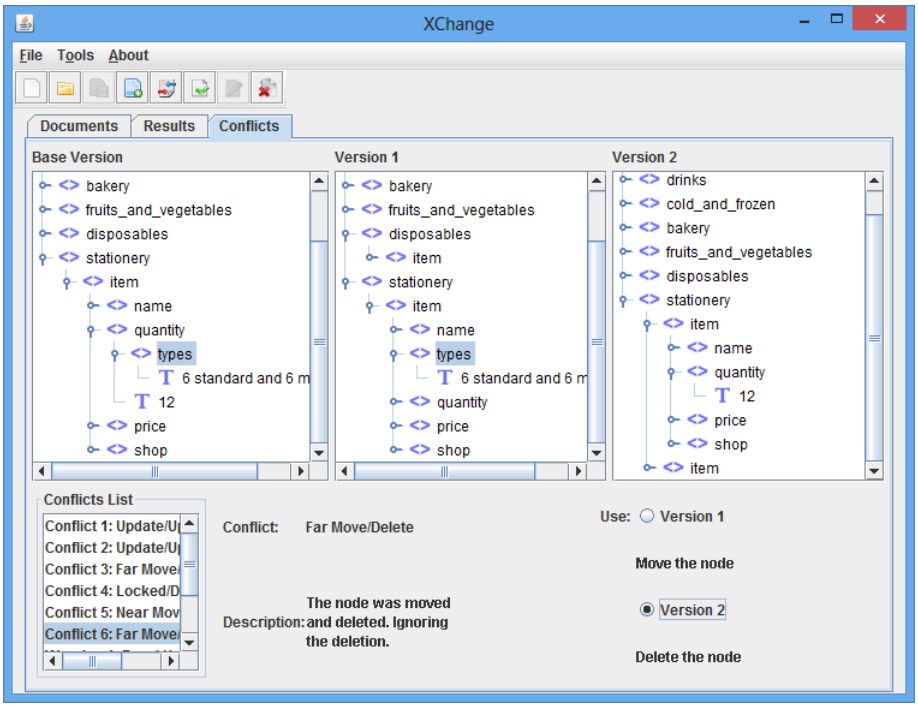

Figura 4.10: Resolução do conflito Far Move/Delete

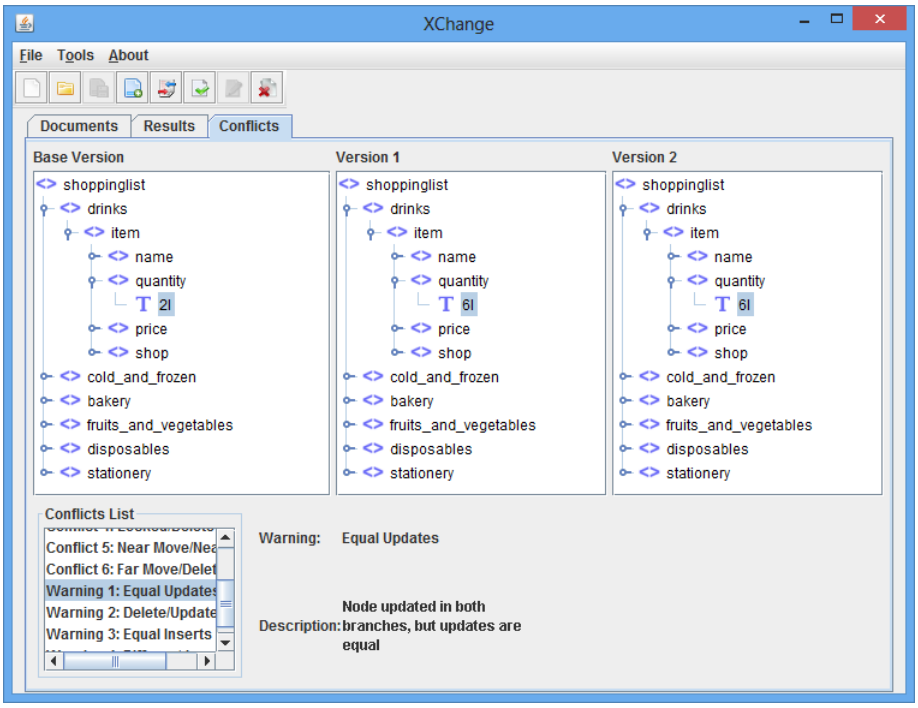

Figura 4.11: Warning Equal Updates

Além dos conflitos, ao mesclar os documentos, o Sr. Smith observou que foram apresentados quatro *warnings*. O primeiro, *Equal Updates*, ocorreu pois ele alterou a quantidade de refrigerante de 2l para 6l, porém, sua esposa sabendo que teria mais alguns convidados, já havia aumentado esta quantidade. Neste caso, ele percebeu que não havia necessidade de escolher qual versão utilizar, pois elas eram iguais. Na Figura 4.11 é possível observar que não há necessidade da escolha do usuário para este aviso.

O aviso *Delete/Update* ocorreu pois a Sra. Smith alterou o item *Smoked salmon* advertindo que a sua tia Maija é alérgica a salmão, e que seria melhor substituí-lo por presunto. Porém, o Sr. Smith já sabia do problema de Maija, então ao ver na lista de convidados que ela estaria presente, ele removeu o salmão da lista. Apesar de ser um aviso, conforme mostra a Figura 4.12, a ferramenta permitia ao Sr. Smith escolher qual versão manter na lista. Então ele concordou com a sugestão de sua esposa, e escolheu a versão 1. Desta forma, sua escolha foi diferente da solução padrão do XChangeMerge, que neste caso opta por ignorar a alteração e remover o elemento.

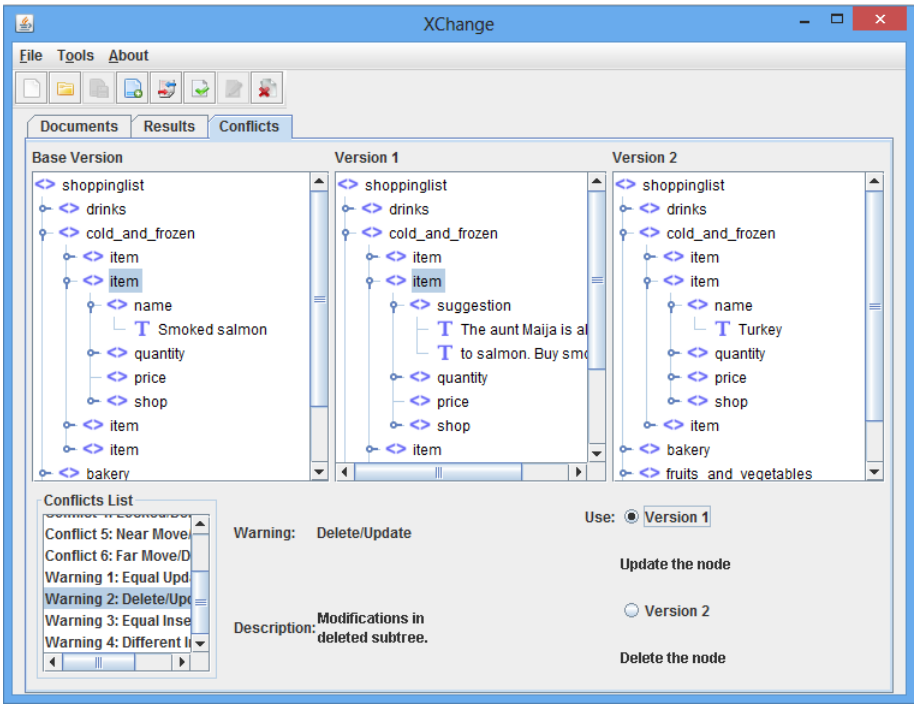

Figura 4.12: Warning Delete Update

O terceiro aviso é o *Equal Inserts*, gerado pois os dois sugeriram o mesmo tipo para as torradas, inserindo o elemento *suggestion* dentro do item *wheat toast*. De maneira similar ao *Equal Updates*, neste caso também não é necessário que o Sr. Smith escolha

uma versão a ser utilizada. A Figura 4.13 ilustra este aviso de conflito.

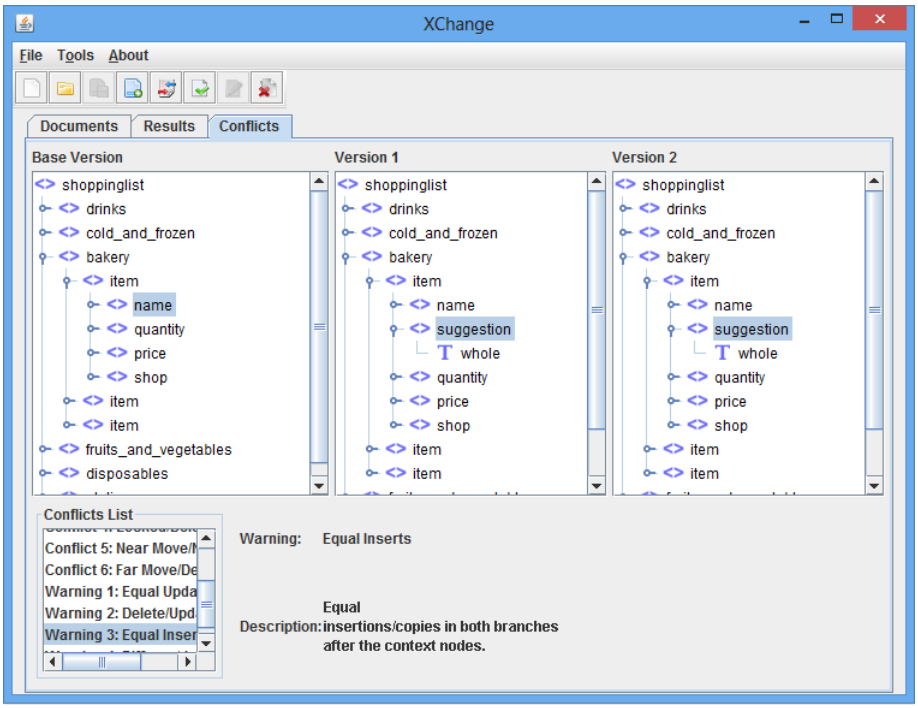

Figura 4.13: Warning Equal Inserts

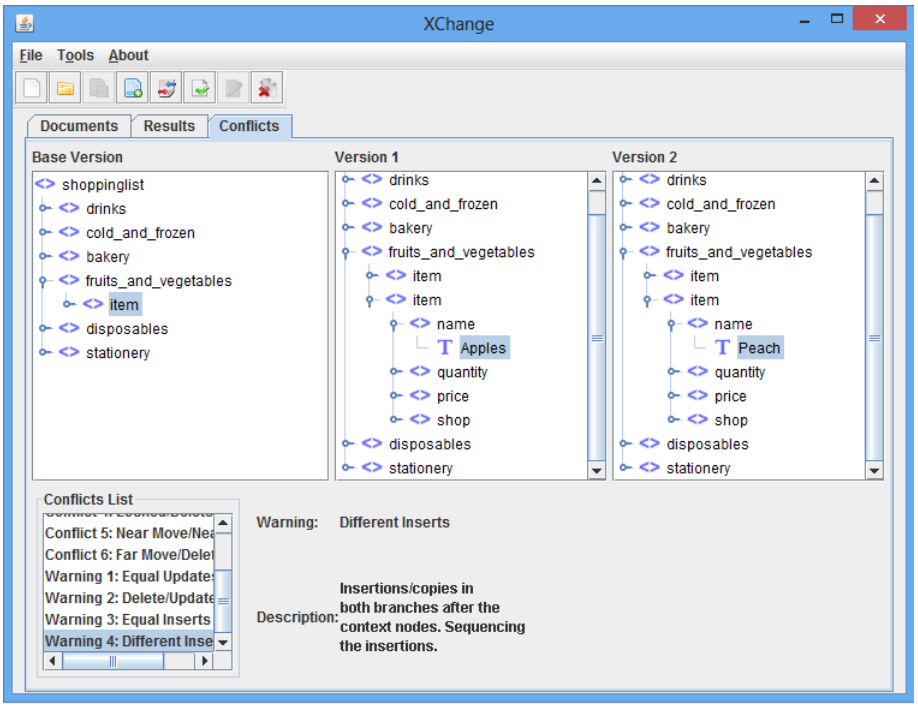

Figura 4.14: Warning Different Inserts

Por fim, o último aviso listado é o *Different Inserts*, apresentado na Figura 4.14. Este aviso ocorreu pois o Sr. Smith inseriu pêssegos na lista, e nesta mesma posição da

lista, sua esposa havia inserido maçãs. Este aviso não apresenta opção para que o usuário escolha o que deseja usar, pois, uma vez que os elementos são diferentes, o XChangeMerge interpreta que são operações independentes. Neste caso a solução automática é inserir o elemento novo da versão 1 seguido do da versão 2, persistindo ambas as alterações.

Após resolver todos os conflitos, o Sr. Smith clicou no botão *Apply Choice* para que a nova versão do documento fosse gerada. O sistema o direcionou pra a aba de resultados, onde ele pode visualizar o documento mesclado no formato de árvore. Ele examinou pacientemente cada item da lista, verificando os itens inseridos por ele e por sua esposa, bem como cada item alterado. Também pode fazer alterações no nome e no texto de alguns elementos. Visto que o documento mesclado estava correto, ele clicou no botão *Write Merged*, para salvar a nova versão do documento em um diretório escolhido por ele. A versão mesclada do documento está descrita na listagem L3.xml no Apêndice.

4.4 Conclusão

O presente capítulo apresentou um exemplo de uso do XChangeMerge, possibilitando o entendimento da ferramenta, e do processo de mesclagem de documentos através da mesma. As figuras apresentadas mostram que o formato de árvore proporciona ao usuário uma melhor visualização dos conflitos, possibilitando sua resolução de maneira eficiente.

O exemplo apresentado mostra que o usuário pode resolver manualmente cada conflito, ou utilizar a solução automática do XChangeMerge quando desejar. Além disso, através da tela de resultados, é possível visualizar informações sobre as mudanças ocorridas, bem como editar, inserir, remover ou mover elementos da versão mesclada.

5 Considerações Finais

O presente trabalho apresentou as principais atividades acerca do gerenciamento de mudanças de documentos XML, e mostrou a necessidade de se tratar estes documentos de maneira diferente dos documentos de texto. Feito isso, mostrou como é feita a diferenciação e a mesclagem de documentos XML de forma específica, considerando o conhecimento disponível acerca do formato dos dados.

Através do estudo comparativo feito no capítulo 3, foi possível verificar a existência de diferentes técnicas desenvolvidas para este fim, e que a maioria das abordagens livres ou não possuem a funcionalidade de mesclagem, ou não possuem uma representação visual adequada que permita a interação do usuário.

Com isso, a abordagem XChangeMerge foi apresentada, trazendo como vantagens uma exibição mais adequada das alterações e um processo de mesclagem mais controlado, e menos propenso a erros. O exemplo de utilização apresentado na seção 4.3 mostrou que a resolução manual dos conflitos melhora a qualidade dos documentos mesclados, uma vez que é o usuário quem escolhe quais alterações devem ser mantidas na nova versão. Esta monografia também mostrou como a interface em formato de árvore ordenada traz vantagens frente a uma interface baseada em arquivos de texto.

A abordagem proposta, ainda possui algumas limitações, muitas delas, estão relacionadas ao algoritmo 3DM. Por exemplo, a eliminação de comentários e de declarações da DTD da versão mesclada. Além disso, o 3DM também não executa uma boa mesclagem em alguns casos. LINDHOLM (2001) cita exemplos que fazem o algoritmo entrar em *loop*, e outros para os quais o casamento dos nós não acontece como seria natural prever, e, portanto, ocorrem resultados inesperados. Desta forma, pretende-se como trabalho futuro, corrigir estas falhas, e também estudar uma possível inclusão de outras abordagens de mesclagem.

Outro trabalho futuro, consiste na integração da abordagem XChange com o Phoenix, no que diz respeito à mesclagem e similaridade de documentos XML. A princípio o XChangeMerge utiliza o algoritmo 3DM, desenvolvido por LINDHOLM (2001). O Phoe-

nix, desenvolvido por PINTO and CAMPELLO (2012) foi aperfeiçoado por MACHADO (2013) incluindo a funcionalidade de mesclagem 3-way. Com isso, ao contrário do 3DM, esta abordagem possui uma interface própria de visualização e controle da mesclagem. Além disso, seu modelo de visualização em árvores é diferente do modelo utilizado no XChangeMerge. Nesta abordagem, as árvores das versões são sobrepostas, de forma que uma única árvore representa o documento completo com todas as alterações. O principal objetivo dessa integração é obter mais uma opção para realizar a mesclagem entre os documentos XML. Além disso, é possível ter acesso ao código desta ferramenta, o que o torna flexível a mudanças, adequado-o para novas funcionalidades.

No que diz respeito à inferência do XChange, foi visto que as regras de inferência em Prolog são fornecidas ou criadas pelo usuário usuário através da ferramenta. Um trabalho futuro, seria evoluir o XChange propondo uma forma de extrair essas regras automaticamente, analisando as mudanças ocorridas nos documentos XML ao longo do tempo. Com técnicas de mineração de dados, pode-se utilizar algoritmos de regras de associação para encontrar relacionamentos e padrões frequentes no conjunto de dados (VASCONCELOS and CARVALHO, 2004), e apresentar ao usuário esses elementos para que ele possa determinar se é uma regra válida e identificá-la, gerando assim uma regra Prolog semi-automaticamente.

Outro trabalho futuro diz respeito a mesclagem semântica de documentos XML Este tipo de mesclagem vem sendo utilizada em vários campos, como por exemplo, em detecção de conflitos em modelagem de dados (COSTA et al., 2013; MAOZ et al., 2011) e códigos fonte (JACKSON and LADD, 1994). A ideia é perceber que alguns conflitos detectados sintaticamente, não existem na realidade, são os chamados falsos positivos. Desta forma, também é possível perceber alguns conflitos que não são detectados pela mesclagem sintática (falsos negativos). Alguns trabalhos já estão disponíveis na literatura, tais como abordagens lógicas para arquivos com informações incertas (HUNTER and LIU, 2006), ou mesclagem semântica através de abordagens que utilizam ontologias (NOY and MUSEN, 2000; CRUZ et al., 2004).

A Apêndice

```
<shoppinglist>
<drinks>
<!-- Equal Updates -->
  <item>
    <name>Soda</name>
    <quantity>2l</quantity>
    <price>1.96</price>
    <shop>LMart</shop>
  </item>
</drinks>
<cold_and_frozen>
  <item>
    <name>Potato chip</name>
    <quantity>2 kg</quantity>
    <price>3.00</price>
    <shop>LMart</shop>
  </item>
<!-- Delete/Update -->
  <item>
    <name>Smoked salmon</name>
    <quantity>3kg</quantity>
    <price>10.60</price>
    <shop>LMart</shop>
  </item>
  <item>
    <name>Turkey</name>
    <quantity>2</quantity>
    <price>10.5</price>
    <shop>LMart</shop>
  </item>
<!-- Update/Update -->
  <item>
    <name>Emmentaler Cheese</name>
    <quantity>500g</quantity>
    <price>3.12</price>
    <shop>LMart</shop>
```

```
37    </item>
38  </cold_and_frozen>
39  <bakery>
40  <!-- Equal Inserts -->
41    <item>
42      <name>Wheat toast</name>
43      <quantity>2 packets</quantity>
44      <price>1.32</price>
45      <shop>LMart</shop>
46    </item>
47  <!-- Far Move/Near Move -->
48    <item>
49      <name>Apple Pie</name>
50      <quantity>1</quantity>
51      <price>
52        <currency>U$</currency>
53        9.98
54      </price>
55      <shop>LMart</shop>
56    </item>
57  <!-- Locked/Delete -->
58    <item>
59      <name>Cookies</name>
60      <quantity>50</quantity>
61      <price>1.5</price>
62      <shop>LMart</shop>
63      <kind>gingerbread</kind>
64    </item>
65  </bakery>
66  <fruits_and_vegetables>
67    <item>
68      <name>Lettuce</name>
69      <quantity>1 head</quantity>
70      <price>0.85</price>
71      <shop>LMart</shop>
72    </item>
73  <!-- Different inserts -->
74
75
76  </fruits_and_vegetables>
```

```
77 <disposables>
78 <!-- Near Move/Near Move -->
79   <item>
80     <name>Napkin</name>
81     <quantity>1 packet</quantity>
82     <price>0.40</price>
83     <shop>LMart</shop>
84   </item>
85 </disposables>
86 <stationery>
87   <item>
88     <name>Christmas cards</name>
89     <quantity>
90       <types>
91         6 standard and 6 musical
92       </types>
93       12
94     </quantity>
95     <price>1.99</price>
96     <shop>LMart</shop>
97   </item>
98 </stationery>
99 </shoppinglist>
```

Listing A.1: L0.xml

```
1  <shoppinglist>
2  <drinks>
3  <!-- Equal Updates - mudou a quantidade de refrigerante para 6l -->
4    <item>
5      <name>Soda</name>
6      <quantity>6l</quantity>
7      <price>1.96</price>
8      <shop>LMart</shop>
9    </item>
10 </drinks>
11 <cold_and_frozen>
12   <item>
13     <name>Potato chip</name>
14     <quantity>2 kg</quantity>
15     <price>3.00</price>
```

```
16      <shop>LMart</shop>
17    </item>
18  <!-- Delete/Update - Advertil que a tia Maija e alergica a peixe -->
19    <item>
20      <suggestion>
21        The aunt Maija is allergic
22        to salmon. Buy smoked ham.
23      </suggestion>
24      <quantity>3kg</quantity>
25      <price>10.60</price>
26      <shop>LMart</shop>
27    </item>
28    <item>
29      <name>Turkey</name>
30      <quantity>2</quantity>
31      <price>10.5</price>
32      <shop>LMart</shop>
33    </item>
34  <!-- Update/Update - trocou o Emmentaler Cheese por Colby Cheese -->
35    <item>
36      <name>Colby Cheese</name>
37      <quantity>500g</quantity>
38      <price>3.50</price>
39      <shop>LMart</shop>
40    </item>
41  </cold_and_frozen>
42  <bakery>
43  <!-- Equal inserts - sugeriu torradas integrais -->
44    <item>
45      <name>Wheat toast</name>
46      <suggestion>whole</suggestion>
47      <quantity>2 packets</quantity>
48      <price>1.32</price>
49      <shop>LMart</shop>
50    </item>
51  <!-- Far Move/Near Move - Trocou a posicao do atributo currency com o  -->
52  <!-- preco no elemento Apple Pie -->
53    <item>
54      <name>Apple Pie</name>
55      <quantity>1</quantity>
```

```
56      <price>
57        9.98
58        <currency>U$</currency>
59      </price>
60      <shop>LMart</shop>
61    </item>
62  <!-- Locked/Delete - Adicionou uma nova sugestao de sabor para o -->
63  <!-- elemento Cookies -->
64    <item>
65      <name>Cookies</name>
66      <quantity>50</quantity>
67      <price>1.5</price>
68      <shop>LMart</shop>
69      <kind>gingerbread</kind>
70      <kind>vanilla</kind>
71    </item>
72  </bakery>
73  <fruits_and_vegetables>
74    <item>
75      <name>Lettuce</name>
76      <quantity>1 head</quantity>
77      <price>0.85</price>
78      <shop>LMart</shop>
79    </item>
80  <!-- Different inserts - inseriu macas na lista -->
81    <item>
82      <name>Apples</name>
83      <quantity>8</quantity>
84      <price>1.26</price>
85      <shop>LMart</shop>
86    </item>
87  </fruits_and_vegetables>
88  <disposables>
89  <!-- Near Move/Near Move - Moveu a quantidade de guardanapos colocando -->
90  <!-- antes do nome -->
91    <item>
92      <quantity>1 packet</quantity>
93      <name>Napkin</name>
94      <price>0.40</price>
95      <shop>LMart</shop>
```

```
96    </item>
97  </disposables>
98  <stationery>
99    <item>
100     <name>Christmas cards</name>
101     <types> 6 standard and 6 musical </types>
102     <quantity>12</quantity>
103     <price>1.99</price>
104     <shop>LMart</shop>
105   </item>
106 </stationery>
107 </shoppinglist>
```

Listing A.2: L1.xml

```
1   <shoppinglist>
2   <drinks>
3   <!-- Equal Updates - mudou a quantidade de refrigerante para 6l -->
4     <item>
5       <name>Soda</name>
6       <quantity>6l</quantity>
7       <price>1.96</price>
8       <shop>LMart</shop>
9     </item>
10  </drinks>
11  <cold_and_frozen>
12    <item>
13      <name>Potato chip</name>
14      <quantity>2 kg</quantity>
15      <price>3.00</price>
16      <shop>LMart</shop>
17    </item>
18  <!-- Delete/Update - Removeu o salmao da lista -->
19
20    <item>
21      <name>Turkey</name>
22      <quantity>2</quantity>
23      <price>10.5</price>
24      <shop>LMart</shop>
25    </item>
26  <!-- Update/Update - trocou o Emmentaler Cheese por Cream Cheese -->
```

```
27    <item>
28      <name>Cream Cheese</name>
29      <quantity>500g</quantity>
30      <price>4.90</price>
31      <shop>LMart</shop>
32    </item>
33  </cold_and_frozen>
34  <bakery>
35  <!-- Equal Inserts - sugeriu torradas integrais -->
36    <item>
37      <name>Wheat toast</name>
38      <suggestion>whole</suggestion>
39      <quantity>2 packets</quantity>
40      <price>1.32</price>
41      <shop>LMart</shop>
42    </item>
43  <!-- Far Move/Near Move - Moveu o atributo currency do elemento -->
44  <!-- Apple Pie para fora do atributo price -->
45    <item>
46      <name>Apple Pie</name>
47      <quantity>1</quantity>
48      <price> 9.98 </price>
49      <currency>U$</currency>
50      <shop>LMart</shop>
51    </item>
52  <!-- Locked/Delete - Removeu a sugestao de sabor para o elemento Cookies -->
53    <item>
54      <name>Cookies</name>
55      <quantity>50</quantity>
56      <price>1.5</price>
57      <shop>LMart</shop>
58    </item>
59  </bakery>
60  <fruits_and_vegetables>
61    <item>
62      <name>Lettuce</name>
63      <quantity>1 head</quantity>
64      <price>0.85</price>
65      <shop>LMart</shop>
66    </item>
```

```
<!-- Different inserts - inseriu pessegos na lista -->
  <item>
    <name>Peach</name>
    <quantity>12</quantity>
    <price>4.99</price>
    <shop>LMart</shop>
  </item>
</fruits_and_vegetables>
<disposables>
<!-- Near Move/Near Move - Moveu a quantidade de guardanapos colocando -->
<!-- abaixo do preco -->
  <item>
    <name>Napkin</name>
    <price>0.40</price>
    <quantity>1 packet</quantity>
    <shop>LMart</shop>
  </item>
</disposables>
<stationery>
  <item>
    <name>Christmas cards</name>
    <quantity>12</quantity>
    <price>1.99</price>
    <shop>LMart</shop>
  </item>
  <item>
    <name>
      Calvin and Hobbes Comic (for Kalle's Xmas party!)
    </name>
    <quantity>1</quantity>
    <price>4.99</price>
    <shop>LMart</shop>
  </item>
</stationery>
</shoppinglist>
```

Listing A.3: L2.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<shoppinglist>
  <drinks>
```

```
4     <item>
5       <name>Soda</name>
6       <quantity>6l</quantity>
7       <price>1.96</price>
8       <shop>LMart</shop>
9     </item>
10  </drinks>
11  <cold_and_frozen>
12    <item>
13      <name>Potato chip</name>
14      <quantity>2 kg</quantity>
15      <price>3.00</price>
16      <shop>LMart</shop>
17    </item>
18    <item>
19      <name>Smoked ham</name>
20      <quantity>3kg</quantity>
21      <price>8.50</price>
22      <shop>LMart</shop>
23    </item>
24    <item>
25      <name>Turkey</name>
26      <quantity>2</quantity>
27      <price>10.5</price>
28      <shop>LMart</shop>
29    </item>
30    <item>
31      <name>Colby Cheese</name>
32      <quantity>500g</quantity>
33      <price>3.50</price>
34      <shop>LMart</shop>
35    </item>
36  </cold_and_frozen>
37  <bakery>
38    <item>
39      <name>Wheat toast</name>
40      <suggestion>whole</suggestion>
41      <quantity>2 packets</quantity>
42      <price>1.32</price>
43      <shop>LMart</shop>
```

```
44      </item>
45      <item>
46        <name>Apple Pie</name>
47        <quantity>1</quantity>
48        <price>9.98</price>
49        <currency>U$</currency>
50        <shop>LMart</shop>
51      </item>
52      <item>
53        <name>Cookies</name>
54        <quantity>50</quantity>
55        <price>1.5</price>
56        <shop>LMart</shop>
57        <kind>gingerbread</kind>
58        <kind>vanilla</kind>
59      </item>
60    </bakery>
61    <fruits_and_vegetables>
62      <item>
63        <name>Lettuce</name>
64        <quantity>1 head</quantity>
65        <price>0.85</price>
66        <shop>LMart</shop>
67      </item>
68      <item>
69        <name>Apples</name>
70        <quantity>8</quantity>
71        <price>1.26</price>
72        <shop>LMart</shop>
73      </item>
74      <item>
75        <name>PeachApples</name>
76        <quantity>128</quantity>
77        <price>4.991.26</price>
78        <shop>LMart</shop>
79      </item>
80    </fruits_and_vegetables>
81    <disposables>
82      <item>
83        <quantity>1 packet</quantity>
```

```
84        <shop>LMart</shop>
85        <name>Napkin</name>
86        <price>0.40</price>
87      </item>
88    </disposables>
89    <stationery>
90      <item>
91        <name>Christmas cards</name>
92        <quantity>12</quantity>
93        <price>1.99</price>
94        <shop>LMart</shop>
95      </item>
96      <item>
97        <name>Calvin and Hobbes Comic (for Kalle's Xmas party!)</name>
98        <quantity>1</quantity>
99        <price>4.99</price>
100       <shop>LMart</shop>
101     </item>
102   </stationery>
103 </shoppinglist>
```

Listing A.4: L3.xml

Referências Bibliográficas

BaseX, http://docs.basex.org/wiki/Statistics, acessado em 04/09/2013.

BOYLAND, J.; GREENHOUSE, A. ; SCHERLIS, W. L. **The Fluid IR: An internal representation for a software engineering environment.** http://www.fluid.cs.cmu.edu:8080/Fluid, acessado em 5/08/2013.

BRAY, T.; PAOLI, J.; SPERBERG-MCQUEEN, C. M.; MALER, E. ; YERGEAU, F. **Extensible markup language (XML) 1.0 (fifth edition)**. World Wide Web Consortium, Recommendation REC-xml-20081126, November 2008.

COBENA, G.; ABITEBOUL, S. ; MARIAN, A. **Detecting changes in XML documents**. In: Data Engineering, 2002. Proceedings. 18th International Conference on, 2002.

COBENA, G.; ABDESSALEM, T. ; HINNACH, Y. **A comparative study for XML change detection.** Verso report number 221, INRIA 2002 (updated 2004), http://www.deltaxml.com/support/documents/articles-and-papers/is2004.pdf, 2004.

COSTA, V. O.; JUNIOR, J. M. B. O. ; MURTA, L. G. P. Semantic conflicts detection in model-driven engineering. **Available as SEKE 2013 technical program**, 2013.

CRUZ, I.; XIAO, H. ; HSU, F. **An ontology-based framework for XML semantic integration**. In: Database Engineering and Applications Symposium, 2004. IDEAS'04. Proceedings. International, p. 217–226. IEEE, 2004.

Portal Brasileiro de Dados Abertos, http://dados.gov.br/dataset/, acessado em 04/09/2013.

DENTI, E.; OMICINI, A. ; RICCI, A. **tuProlog: A light-weight Prolog for internet applications and infrastructures**. In: Practical Aspects of Declarative Languages. 2001.

ESTUBLIER, J.; LEBLANG, D.; HOEK, A. V. D.; CONRADI, R.; CLEMM, G.; TICHY, W. ; WIBORG-WEBER, D. Impact of software engineering research on the practice of software configuration management. **ACM Trans. Softw. Eng. Methodol.**, v.14, n.4, p. 383–430, oct 2005.

GARCIA, P. A. **EDX: Uma abordagem de apoio ao controle de mudanças de documentos XML**. Trabalho de Conclusão de Curso - UFJF, 2012.

GAZZOLA, P. O. L. **Inferência em documentos XML utilizando Prolog**. Trabalho de Conclusão de Curso - UFJF, 2011.

GUTIÉRREZ-SOTO, C.; BARRA, A.; LANDAETA, A. ; URRUTIA, A. **Change detection by level (CDL): An efficient algorithm to detect change on XML documents**. In: Computer Sciences and Convergence Information Technology (ICCIT), 2010 5th International Conference on, 2010.

HUNTER, A.; LIU, W. Merging uncertain information with semantic heterogeneity in xml. **Knowledge and Information Systems**, v.9, n.2, p. 230–258, 2006.

JACKSON, D.; LADD, D. A. **Semantic diff: A tool for summarizing the effects of modifications**. In: Proceedings of the International Conference on Software Maintenance, ICSM '94, p. 243–252, Washington, DC, USA, 1994. IEEE Computer Society.

KUHN, H. W. The Hungarian method for the assignment problem. **Naval Research Logistics Quarterly**, 2006.

LINDHOLM, T. **A 3-way merging algorithm for synchronizing ordered trees - the 3DM merging and differencing tool for XML**. Dissertação de Mestrado - Helsinki University of Technology, 2001.

LINDHOLM, T. A three-way merge for XML documents. **ACM**, 2004.

MACHADO, L. C. **Phoenix - cálculo de similaridade de documentos XML - XML diff e merge**. Trabalho final de disciplina - UFF, 2013.

MAOZ, S.; RINGERT, J. O. ; RUMPE, B. **Addiff: semantic differencing for activity diagrams**. In: Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering, ESEC/FSE '11, p. 179–189, New York, NY, USA, 2011. ACM.

MARTINS, G.; JUNIOR, C. L.; OLIVEIRA, A.; MURTA, L. ; BRAGANHOLO, V. XChange: Compreensão de mudanças em documentos XML Demos e Aplicações. **Simpósio Brasileiro de Banco de Dados (SBBD)**, 2013.

MURTA, L. G. P. **Gerência de configuração no desenvolvimento baseado em componentes**. Dissertação de Mestrado - COPPE, UFRJ, 2006.

MYERS, E. W. An O(ND) difference algorithm and its variations. **Algorithmica**, 1986.

NOY, N. F.; MUSEN, M. A. **Algorithm and tool for automated ontology merging and alignment**. In: Proceedings of the 17th National Conference on Artificial Intelligence (AAAI-00). Available as SMI technical report SMI-2000-0831, 2000.

OLIVEIRA, A. P. D. **Gerenciando alterações em documentos XML**. Trabalho de Conclusão de Curso - UFRJ, 2007.

OLIVEIRA, A. P. D.; OLIVEIRA, A. M. D.; BRAGANHOLO, V. ; MURTA, L. Gerenciando alterações em documentos XML. **Revista de Informática Teórica e Aplicada - RITA**, v.17, n.3, 2010.

OLIVEIRA, D. V. D. D. **XChangeSim: Compreensão de mudanças de documentos XML baseada em similaridade**. Trabalho de Conclusão de Curso - UFJF, 2013.

OSO CORPORATION - Project Merge, http://www.projectmerge.com/index.php, acessado em 20/08/2013.

PETERS, L. Change detection in XML trees: a survey. **3rd Twente Student Conference on IT**, 2005.

PINTO, B. F.; CAMPELLO, F. **Cálculo de similaridade de documentos XML**. Trabalho de Conclusão de Curso - UFF, 2012.

RÖNNAU, S.; PHILIPP, G. ; BORGHOFF, U. M. **Efficient change control of XML documents**. In: Proceedings of the 9th ACM symposium on Document engineering, 2009.

SILVA, R. B. **XPerseus: Uma interface gráfica para detecção de diferenças entre documentos XML**. Trabalho de Conclusão de Curso - UFJF, 2011.

THAO, C.; MUNSON, E. V. **Using versioned tree data structure, change detection and node identity for three-way xml merging**. In: Proceedings of the 10th ACM symposium on Document engineering, DocEng '10, p. 77–86, New York, NY, USA, 2010. ACM.

THAO, C. **A configuration management system for software product lines**. Dissertação de Mestrado - University of Wisconsin-Milwaukee, 2012.

VASCONCELOS, L. M. R. D.; CARVALHO, C. L. D. **Aplicação de regras de associação para mineração de dados na web**. Technical report, Instituto de Informática - UFG, 2004.

WANG, Y.; DEWITT, D. J. ; CAI, J.-Y. **X-Diff: An effective change detection algorithm for XML-documents**. In: Data Engineering, 2003. Proceedings. 19th International Conference on, 2003.

XMLData Repository, http://www.cs.washington.edu/research/xmldatasets/, acessado em 04/09/2013.

ZHANG, K. **A new editing based distance between unordered labeled trees**. In: Combinatorial Pattern Matching, p. 254–265. Springer, 1993.