Gerenciando Alterações em Documentos XML

Alexandre Pereira de Oliveira [1]
Alessandreia Marta de Oliveira [2 3]
Vanessa Braganholo [2]
Leonardo Murta [2]

Resumo: Documentos XML estão cada vez mais presentes em projetos de desenvolvimento, e geralmente são colocados em sistemas de controle de versões juntamente com os outros arquivos do projeto. Estes sistemas não consideram o formato específico dos arquivos XML, tratando-os como arquivos comuns de texto. Desta forma, mesclagens automáticas podem gerar documentos mal formados silenciosamente, e exibir, durante comparações, alterações que não possuem qualquer relevância para documentos XML, frequentemente mascarando alterações relevantes. Este artigo propõe uma abordagem para exibição de diferenças e apoio à mesclagem de arquivos XML, utilizando algoritmos clássicos propostos na literatura. A abordagem proposta permite a execução de alterações em paralelo de forma mais controlada, segura e eficiente para documentos XML.

Abstract: XML documents are becoming common in software development projects, and are generally versioned in version control systems together with other files of the project. Such systems do not consider the specificities of the XML format and deals with them in a standard fashion, like any other plain text file. Consequently, automatic merges can silently generate non well-formed documents and display, during comparisons, modifications that are not relevant to XML documents. This paper proposes an approach to display the differences between two XML files, and to merge them by using classical algorithms. The proposed approach allows the execution of modifications in parallel in a more controlled, safe, and efficient way for XML documents.

1 Introdução

Projetos de desenvolvimento de software são, em sua grande maioria, executados por um conjunto de desenvolvedores que trabalham em paralelo sobre um mesmo conjunto de artefatos. A velocidade e a interdependência que pode ocorrer entre os trabalhos de diferentes desenvolvedores torna o processo de controle de versões manual dos artefatos envolvidos custoso e prejudicial à eficiência da equipe. Alterações executadas por um desenvolvedor podem, de forma sutil, comprometer as alterações executadas por outros, levando a um processo de mesclagem (do inglês, *merge* [11]) difícil e propenso a erros.

Este problema tem sido resolvido por sistemas de controle de versões (*Version Control System* - VCS) [21], que visam automatizar o armazenamento, a recuperação, a identificação e a mesclagem de versões de artefatos (também chamadas de revisões [11]). Nesses sistemas, usualmente existe um repositório central, mas cada desenvolvedor trabalha em uma cópia local dos arquivos. Caso uma política otimista seja adotada, a qualquer momento, o usuário pode editar, apagar, criar e mover os arquivos que julgar necessário. Quando achar conveniente, ele envia para o repositório central suas modificações. Este processo é conhecido como *check-in* ou *commit*. O usuário também pode ordenar que o sistema aplique em sua cópia local as alterações enviadas ao repositório por outros usuários. Este processo é conhecido como *update*. Deste modo, os usuários trabalham em cópias diferentes dos arquivos e somente executam *check-ins* e *updates* quando julgarem conveniente. Um *check-in* de um usuário não implica em *updates* automáticos para os outros – eles somente terão essas alterações ao executarem um *update* explícito.

[1]Visagio alexandre.oliveira@visagio.com.br
[2]Instituto de Computação, Universidade Federal Fluminense
{alessandreia,vanessa,leomurta}@ic.uff.br
[3]DCC, Instituto de Ciências Exatas, Universidade Federal de Juiz de Fora

Este cenário cria situações em que um usuário pode executar alterações em uma versão desatualizada de um arquivo. Por exemplo, suponha que dois desenvolvedores, João e Maria, estejam com o projeto devidamente atualizado em suas cópias locais. João, então, executa alterações em um determinado arquivo, e executa um *check-in*. Em paralelo, Maria, sem conhecimento do *check-in* realizado por João, altera o mesmo arquivo, por motivos diferentes. Quando Maria tentar executar o *check-in*, ela será informada que a versão que ela tem do arquivo não é a mais atual, e que ela deve executar um *update* antes. Como o seu arquivo está modificado, será preciso mesclar as alterações executadas por João com as alterações executadas por Maria, de acordo com um processo de mesclagem predeterminado (detalhes na seção 2). Se as alterações afetarem áreas em comum do arquivo e o VCS não conseguir mesclar as alterações, ele acusará um **conflito**, e o usuário será obrigado a mesclar as alterações manualmente antes de executar o *check-in* novamente. O conflito usualmente ocorre quando dois usuários executam alterações em paralelo, de forma independente, sobre um mesmo trecho de um arquivo. O primeiro usuário a enviar as alterações não terá problemas, mas o segundo será obrigado a resolver o conflito, mesclando as alterações executadas pelos dois ou descartando as de um deles.

Atualmente, muitos dos principais *frameworks* para o desenvolvimento de sistemas computacionais utilizam arquivos XML. Tais arquivos podem ser usados, por exemplo, para definir configurações ou mesmo para definir o *layout* da aplicação. Normalmente o código-fonte desses programas é submetido a algum VCS, sendo que os arquivos XML são versionados juntamente com os demais arquivos do sistema.

A maioria dos VCSs disponíveis, como o CVS [2] e o Subversion [7], trata arquivos XML da mesma forma que um arquivo de texto puro, e, portanto, negligencia todo o conhecimento acerca do formato XML. Isso pode acarretar na geração de documentos XML mal formados durante o processo de mesclagem, além de outros problemas. Ao dar ao VCS a informação de que ele está manipulando um arquivo XML, diversas melhorias podem ser implementadas, e problemas podem ser evitados.

Diante disso, o objetivo deste trabalho é estabelecer uma forma mais adequada para o controle de versões de documentos XML, levando em conta o formato estruturado dos documentos ao obter e exibir as diferenças entre versões, e conduzindo um processo de mesclagem mais preciso e controlado, sem, no entanto, prejudicar o controle das versões dos demais tipos de arquivos.

O restante deste artigo está estruturado como segue. A Seção 2 apresenta um estudo das opções disponíveis para executar a detecção de alterações em documentos XML e uma descrição mais detalhada da operação de mesclagem, deixando mais claro o problema tratado neste artigo. A Seção 3 apresenta o algoritmo 3dm em detalhes. Como justificado mais adiante, ele foi o algoritmo base escolhido pela abordagem proposta. A abordagem proposta é detalhada na Seção 4 e a Seção 5 apresenta alguns trabalhos relacionados. A Seção 6 apresenta uma avaliação da abordagem proposta. Finalmente, a Seção 7 discute as conclusões, limitações e trabalhos futuros.

2 Detecção e Mesclagem de Alterações em XML

Para controlar versões de documentos XML, é necessário uma forma de detectar exatamente quais são as diferenças entre duas versões de documentos, ou seja, o que foi modificado de uma versão para a outra.

É necessário estabelecer **Unidades de Versão** e **Unidades de comparação** mais adequadas ao formato, como discutido por Murta et al. [15]. VCS em geral utilizam linhas como Unidades de Comparação para qualquer arquivo texto (para detecção de conflitos), e arquivos como Unidades de Versão. Isso significa que as revisões são referentes aos arquivos, e que ao buscar diferenças entre revisões, a comparação ocorre linha por linha. Esse comportamento se mostrou bastante adequado para arquivos simples de texto puro, mas é inconveniente para arquivos que tenham uma estrutura interna mais elaborada, como é o caso de arquivos XML.

Para ilustrar este problema, analise o fragmento de um documento XML apresentado na Figura 1.

Duas pessoas poderiam fazer modificações em paralelo neste documento e alterar o elemento *dataTable* de formas diferentes, como exibido nas Figuras 2 e 3.

```
<t:dataTable
  id="dataContratosAbertos" var="contrato"
  value="#{projetoVisualizacao.contratosAbertos}"
  rowId="#{contrato.id}"
  rowIndexVar="rowContrato">
```

Figura 1. Fragmento de uma página JSF

```
<t:dataTable
  rendered="#{projetoControlador.exibirContratosAbertos}"
  id="dataContratosAbertos" var="contrato"
  value="#{projetoVisualizacao.contratosAbertos}"
  rowId="#{contrato.id}"
  rowIndexVar="rowContrato">
```

Figura 2. Fragmento de uma página JSF após alteração por um desenvolvedor

```
<t:dataTable
  id="dataContratosAbertos"
  var="contrato"
  value="#{projetoVisualizacao.contratosAbertos}"
  rowId="#{contrato.id}"
  rendered="#{permissoes.visualizaContratosAbertos}"
  rowIndexVar="rowContrato">
```

Figura 3. Fragmento de uma página JSF após alteração em paralelo por um outro desenvolvedor

No caso, dois desenvolvedores inseriram o mesmo atributo em um determinado elemento, porém em linhas distantes, já que o elemento se estende por algumas linhas. Muitos VCS, no entanto, executariam a mesclagem silenciosamente sem detectar qualquer conflito, resultando no documento exibido na Figura 4. Contudo, o documento resultante é malformado, visto que o elemento passa a ter dois atributos com o mesmo nome, o que não é permitido pelas regras de formação de arquivos XML [3]. No caso de arquivos XML, seriam mais adequadas outras unidades de comparação e versão, como elementos e atributos, ao invés de linhas.

```
<t:dataTable
  rendered="#{projetoControlador.exibirContratosAbertos}"
  id="dataContratosAbertos"
  var="contrato"
  value="#{projetoVisualizacao.contratosAbertos}"
  rowId="#{contrato.id}"
  rendered="#{permissoes.visualizaContratosAbertos}"
  rowIndexVar="rowContrato">
```

Figura 4. Resultado da mesclagem executada por um VCS convencional: um documento XML malformado

Esse problema ocorre pois os algoritmos convencionais utilizados em VCS para mesclagem são genéricos, baseados em texto, sem suporte para a sintaxe específica do arquivo [8].

Em alguns casos, como quando uma determinada ferramenta controla diretamente cada alteração executada sobre um artefato, é possível que essa ferramenta armazene exatamente cada passo que foi executado para a alteração. Estas informações poderiam, portanto, ser utilizadas para reconstruir uma das versões do artefato a partir da outra. Entretanto, no caso geral, tais informações não estão disponíveis, sendo necessário detectar as diferenças entre as versões e gerar uma representação dessa diferença, conhecida como delta [8].

As operações representadas pelos deltas geralmente seguem o modelo definido por Tai [20]. Exis-

tem operações de inserção, remoção e atualização. Algumas abordagens ainda utilizam operações de movimentação e cópia.

2.1 Algoritmos de *Diff*

Existem alguns algoritmos dedicados exclusivamente à comparação e detecção de diferenças em XML, que poderiam ser utilizados para comparar arquivos XML. No entanto, somente isso não é suficiente. É preciso também exibir as alterações detectadas de forma mais intuitiva, de modo a auxiliar o usuário no processo de mesclagem. De fato, não foi possível identificar soluções livres ou comerciais que permitam o uso destes algoritmos aliado a uma comparação visual das diferenças entre documentos XML. Uma visualização das alterações de forma gráfica pode trazer grandes benefícios. Ferramentas tradicionais mostram as diferenças em cada linha, o que é inadequado para arquivos XML. Quebras de linha entre atributos de um elemento, ou mesmo troca de ordem entre os atributos de um elemento, são considerados por essas ferramentas como alterações, sendo que na prática são completamente irrelevantes para arquivos XML. Essas alterações serão registradas pelo VCS, o que gera uma sobrecarga desnecessária, e também serão exibidas por ferramentas de comparação de arquivos, gerando sobrecarga cognitiva em relação à visualização das alterações realmente relevantes.

Cobena et al. [6] listam os principais algoritmos disponíveis para cálculos de deltas (também chamados *diffs*) em documentos XML e estabelecem uma comparação entre eles. A maioria dos algoritmos se aproveita do formato em árvore dos arquivos XML para realizar as comparações. Alguns se baseiam em árvores não ordenadas, como o X-Diff [22], enquanto outros se baseiam em árvores ordenadas, como o XMLTreeDiff [9] e o XyDiff [14].

É preciso lembrar que em documentos XML os elementos são ordenados, e que portanto, a rigor, não seria correto modelar um arquivo XML como uma árvore não ordenada. No entanto, dependendo da aplicação, a ordem dos elementos pode não ser relevante, e tais algoritmos podem ser valorosos. Um exemplo seria um documento XML que represente as tuplas de uma tabela de um banco de dados. As tuplas no banco de dados não são ordenadas, e, portanto, a ordem entre os elementos no documento XML não é relevante. Entretanto, no contexto deste trabalho, não convém limitar a semântica dos documentos suportados. Desta forma, algoritmos baseados em árvores desordenadas não podem ser utilizados.

O X-Diff [22] detecta o mapeamento mínimo entre os filhos de duas sub-árvores, reduzindo o problema a um problema de fluxo máximo com custo mínimo. Esse algoritmo encontra um delta mínimo, em tempo quadrático, porém é baseado em árvores não ordenadas. O XMLTreeDiff [9] trabalha com árvores ordenadas e utiliza *hashes* para comparar sub-árvores. Já o XyDiff [14] intuitivamente remove da comparação árvores idênticas dos dois documentos, também utilizando *hashes*. Ele considera os atributos declarados como ID na DTD do documento para identificar elementos. O algoritmo tem complexidade de tempo linear, mas não necessariamente encontra o delta mínimo.

2.2 Algoritmos de Mesclagem

A operação de mesclagem (*merge*), por sua vez, consiste em fundir modificações executadas em paralelo sobre uma versão base em comum. Se dois ou mais usuários de um VCS obtêm a mesma versão de um determinado arquivo e executam modificações em paralelo, é necessário consolidar as modificações realizadas pelos usuários em uma mesma versão. Na prática, um dos usuários enviaria o arquivo para o VCS, que passaria a considerar aquela como a última versão. Quando os demais usuários tentassem enviar suas versões modificadas, eles seriam informados pelo sistema que aquele arquivo foi modificado por outro usuário, e que eles teriam que atualizar o arquivo. É neste momento que a mesclagem acontece.

Alguns VCS preferem evitar esse caso utilizando bloqueios nos arquivos, de forma que apenas um usuário tem o direito de modificar um arquivo em um determinado instante de tempo. Esta é a política padrão adotada pelo Visual SourceSafe da Microsoft, por exemplo. No entanto, como esse bloqueio limita o trabalho em paralelo e introduz diversos outros problemas administrativos [18], vamos considerar apenas os sistemas de controle de versão que não adotam esta técnica.

O CVS e o Subversion, bem como a maioria dos VCS, utilizam o programa GNU diff3 [19] diretamente, ou adaptações do seu algoritmo. O diff3 é um algoritmo bastante simples para mesclagem de texto. Apesar de ser muito satisfatório para arquivos texto e de ser um algoritmo já consagrado e estável, o diff3 apresenta as desvantagens já apontadas na Seção 2 quando aplicado a documentos XML. Por isso, Lindholm [13] estudou a questão e criou o algoritmo 3dm, voltado especificamente para a mesclagem de arquivos XML, bem como para arquivos XHTML.

O objetivo do 3dm é obter uma mesclagem automática, sem qualquer interação com o usuário. Ele perde, portanto, a oportunidade de permitir ao usuário a resolução manual de conflitos, correndo o risco de gerar arquivos com mesclagens indesejáveis ou incorretas. Além disso, o 3dm não permite que o usuário acompanhe visualmente as alterações.

O 3dm se baseia em árvores ordenadas e funciona em duas fases. Inicialmente, são calculados deltas entre a versão base e cada uma das versões modificadas, fase esta chamada de detecção de alterações. Em seguida os dois deltas são aplicados sobre o documento, em uma fase denominada conciliação.

Na **detecção de alterações**, cada uma das versões é comparada com a versão original de forma independente. É necessário casar os elementos da versão original com os elementos das versões alteradas para descobrir quais elementos foram alterados, inseridos ou removidos. O algoritmo utiliza uma função que determina a similaridade entre dois elementos, levando em conta a similaridade entre os conteúdos (no caso de elementos que contêm textos), o nome dos elementos, atributos, subelementos, dentre outras características [12]. Esta função é aplicada em subárvores podadas – subárvores com todos os filhos de alguns elementos removidos. Desta forma ele consegue casar partes de subárvores que tiveram elementos adicionados ou removidos.

São diferenciados elementos que foram casados apenas no conteúdo (ou seja, com mudança estrutural); elementos que foram casados apenas na estrutura (com mudança de conteúdo); ou que foram completamente casados. Assim, o algoritmo pode tratar de forma diferenciada cada um dos casos, propagando apenas as mudanças que devem ser propagadas, gerando resultados mais adequados. A partir dos casamentos, é possível obter um delta: cada elemento que não casa com nenhum elemento da versão base foi inserido; cada elemento que não casa com a versão alterada, foi removido; elementos que casam apenas na estrutura tiveram seu conteúdo alterado; e elementos que casam apenas no conteúdo tiveram seu contexto alterado – foram movidos ou tiveram filhos retirados.

Na fase de **conciliação**, os casamentos obtidos na fase anterior para as duas versões são utilizados para elaborar a versão final do documento. No 3dm, um conjunto de regras pré-definidas resolve os conflitos automaticamente. Ele apenas, de forma opcional, registra quais conflitos ocorreram. Além do documento com as alterações aplicadas e o registro de conflitos, o algoritmo também pode fornecer uma lista das alterações aplicadas. Além disso, como apontado por Lindholm [13], ele também pode ser utilizado para obter a diferença entre dois documentos XML.

Na próxima seção, apresentamos o 3dm em detalhes. Tais detalhes são importantes para o entendimento da abordagem proposta neste trabalho.

3 Algoritmo 3dm

O 3dm trabalha com 5 tipos de operações, listadas a seguir. Na lista dos tipos de alterações a seguir, entende-se por nó um elemento ou o seu conteúdo – já que o 3dm (e ferramentas e algoritmos para XML em geral) trata ambos como nós em seu modelo de árvore. Atributos são considerados parte integrante do elemento ao qual pertencem.

- *insert* (inserção) – Consiste na inserção de um nó (elemento ou conteúdo textual) em alguma parte do documento;

  delete (remoção) – Consiste na remoção de um nó do documento;

- *update* (atualização) – Significa que o nó teve o seu conteúdo alterado. No caso de elementos, pode ser o seu nome ou um de seus atributos;

- *move* (movimentação) – Significa que um nó foi movido para outra parte do documento;

- *copy* (cópia) – 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.

É preciso destacar que nem sempre é possível detectar qual alteração foi realmente executada pelo usuário, já que não é possível ter acesso à sequência de edições executada. Por exemplo, o usuário poderia modificar o nome de um elemento, e depois movê-lo. Ou então o usuário poderia apagar este elemento e inserir um outro. Apenas analisando o resultado final, é complexo diferenciar os dois casos. Portanto, é importante ressaltar que o *log* de alterações não deve ser considerado como sendo necessariamente o conjunto de alterações executadas pelo usuário, mas sim como um delta que lista as diferenças entre dois arquivos e permite construir um arquivo a partir do outro.

3.1 Formato do Log de Alterações

O *log* gerado pelo 3dm consiste de um arquivo XML cujo elemento raiz se chama "*edits*" e que contém a sequência de alterações detectadas. Cada alteração é representada por um elemento com o formato <operação *path src originTree originList originNode*>, onde *path* é o caminho do nó na árvore mesclada; *src* é o caminho do nó na árvore original; *originList* é o caminho do nó pai envolvido em uma operação de remoção, na versão em que a operação ocorreu (ou seja, o pai que deveria conter o nó, mas não o contém); *originNode* é o caminho do nó na versão em que a alteração ocorreu; e *originTree* representa a versão que gerou a alteração (*branch1* ou *branch2*).

Caminhos são a forma como o *log* identifica os nós. Trata-se do caminho da raiz até o nó, representado por uma sequência de números separados por barra. O número indica a posição do filho naquele nível, sendo a primeira posição representada por zero. Assim, "/0/1" indica o caminho para o segundo filho da raiz. Note que como os arquivos XML sempre têm apenas uma raiz, todos os caminhos sempre iniciam com "/0", que é o caminho até o nó raiz.

Cada operação utiliza apenas parte dos parâmetros. Por exemplo, no caso de uma remoção, não existe o parâmetro *path*, já que o nó foi removido da árvore mesclada. Cada tipo de operação fornece um comportamento específico, manipulando os parâmetros apropriados, como descrito a seguir:

- *insert* – O nó em *originNode* da versão *originTree* foi inserido na árvore final, assumindo o caminho *path*;
- *delete* – O nó em *src* da versão base foi removido na versão *originTree*, sendo que o seu pai na *originTree* é *originList*;
- *update* – O nó em *src* da versão base foi alterado na versão *originTree*, onde seu caminho era *originNode*, e seu caminho na versão mesclada é *path*;
- *move* – O nó em *src* da versão base foi movido na versão *originTree*, onde seu novo caminho era *originNode*, e seu caminho na versão mesclada é *path*;
- *copy* – O nó em *src* da versão base foi copiado na versão *originTree*, onde o caminho da cópia era *originNode*, e na versão mesclada é *path*;

Como exemplo, na Figura 5 é exibida a versão base (a) e duas versões modificadas de forma independente (b) e (c). O log de alterações gerado para este caso é apresentado na Figura 5(d).

3.2 Conflitos e Avisos

O 3dm, que tem como objetivo uma mesclagem sem interação com o usuário, tem regras pré-estabelecidas que resolvem todos os conflitos. Mas, é preciso ter em mente que a resolução automática de conflitos pode nem sempre ser a melhor solução ou ao menos uma solução correta [11].

Na Tabela 1 é apresentada a lista de conflitos identificados e registrados pelo 3dm. A classificação dos tipos de conflito não é a originalmente utilizada por Lindholm [12], já que ele utilizava o mesmo nome para diferentes tipos de conflitos, chamando vários deles simplesmente de "*move*". Na tabela,

```
(a)          (b)           (c)          (d)

<R>          <R a="2">     <R>          <edits>
  <a>          <a>           <c>          <update path="/0" src="/0"
    <b/>         <b/>          <c1/>          originTree="branch1" originNode="/0"/>
    <m/>         <m>         </c>         <delete src="/0/0"
  </a>             <i/>      <c>            originTree="branch2" originList="/0"/>
  <c>            </m>          <c1/>      <copy path="/0/1" src="/0/1"
    <c1/>      </a>          </c>           originTree="branch2" originNode="/0/1"/>
  </c>         <c>           <m/>         <move path="/0/2" src="/0/0/1"
</R>             <c1/>     </R>           originTree="branch2" originNode="/0/2"/>
               </c>                       <insert path="/0/2/0" originTree="branch1"
             </R>                           originNode="/0/0/1/0"/>
                                        </edits>
```

Figura 5. Documentos XML e log de alterações: (a) Versão Base; (b) Versão 1; (c) Versão 2; (d) Log de alterações

Tabela 1. Descrição dos conflitos tratados pelo 3dm (adaptado de [12]).

Tipo de Conflito	Descrição	Resolução Padrão
Update/Update	Um nó sofreu alterações diferentes nas duas versões	Usar a atualização executada pela versão 1
Near Move/Near Move	Um nó foi movido para posições diferentes no mesmo pai	Usar a movimentação executada pela versão 1
Far Move/Near Move	Um nó foi movido para outra posição no mesmo pai em uma das versões, e para outro lugar na outra versão	Ignorar a movimentação dentro do mesmo pai
Far Move/Far Move	Um nó foi movido para lugares diferentes nas duas versões, nenhum deles sendo o mesmo pai	Aceitar as duas movimentações – ou seja, haverá uma cópia extra
Far Move/Delete	Um nó foi movido para outro pai em uma das versões, e removido na outra	Ignorar a remoção
Locked/Delete	Um nó é parte do contexto de uma operação em uma das versões, e foi removido da outra.	Remover assim mesmo

"*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.

Além dos conflitos, o 3dm ainda registra "avisos de conflito" (*conflict warnings*), que são, de acordo com Lindholm [12], falsos conflitos, ou conflitos que têm uma resolução óbvia. Na Tabela 2, são apresentados os tipos de avisos gerados pelo 3dm. Note que os dois primeiros casos têm solução trivial, ou seja, se as alterações são iguais, não há problema. As inserções também têm resolução trivial. A única questão é em que ordem elas deveriam acontecer, e o algoritmo se encarrega de estabelecer uma. O último é o mais controverso, mas que também tem uma solução padrão oferecida pelo algoritmo.

Como pode ser observado, as resoluções padrão de conflitos propostas por Lindholm [12] não representam necessariamente o desejo do usuário. Desta forma, é de suma importância a disponibilização de apoio para a resolução manual desses conflitos, através de uma interface visual e intuitiva, integrado a um sistema de controle de versões real.

3.3 Formato do *Log* de Conflitos e Avisos

Os conflitos e avisos, apesar de serem tratados automaticamente pelo algoritmo, também são registrado em um log. Este log consiste de um arquivo XML cujo elemento raiz se chama *conflictlist*, e que contém a sequência dos conflitos e avisos detectados. Caso existam conflitos, haverá um elemento

Tabela 2. Descrição dos avisos (*warnings*) registrados pelo 3dm (adaptado de [12])

Tipo de Aviso	Descrição	Resolução
Equal Updates	Um nó sofreu alterações iguais nas duas versões	Usar a atualização executada pela versão 1
Equal Inserts	Nós foram inseridos ou copiados para o mesmo lugar em ambas as versões, e os nós são iguais	Usar a movimentação executada pela versão 1
Different Inserts	Nós foram inseridos ou copiados em ambas as versões, e elas são diferentes	Inserir os nós da versão 1, seguidos dos nós da versão 2
Delete/Update	Um nó que pertencia a uma subárvore que foi removida em uma das versões, foi alterado na outra versão.	Remover a subárvore, ignorando a alteração

chamado *conflicts*, que tem como filhos os conflitos, e se existirem avisos, haverá um elemento *warnings* contendo os avisos.

Em cada conflito, o nome do elemento pode ser *update*, *move* ou *delete*, e para cada aviso, pode ser *update*, *insert* ou *delete*. O elemento que identifica o conflito contém uma descrição textual em inglês do conflito. Por exemplo, *Node updated in both branches, using branch 1* (Nó atualizado nas duas versões, utilizando a versão 1). Além desta descrição textual, o elemento contém uma sequência de elementos *node*. Os elementos *node* identificam os nós envolvidos no conflito em cada versão, e contêm os atributos *tree*, que indicam a qual versão eles se referem ("*merged*", "*base*", "*branch1*" ou "*branch2*") e *path*, que indica o caminho naquela determinada versão. Um exemplo é mostrado na Figura 6.

```
<update>Node updated in both branches, using branch 1
  <node tree="merged" path="/0/0/1/0" />
  <node tree="base" path="/0/0/1/0" />
  <node tree="branch1" path="/0/0/1/0" />
  <node tree="branch2" path="/0/0/1/0" />
</update>
```

Figura 6. Trecho de *log* de conflitos mostrando um conflito *Update/Update*.

Apesar de ser bastante completo, o 3dm não exibe as alterações para o usuário. Seus logs não podem ser analisados diretamente pelo usuário para este fim, uma vez que não são intuitivos. Além disso, como já ressaltado, a mesclagem totalmente automática pode trazer resultados indesejados. Para contornar estas deficiências, a próxima seção apresenta a abordagem proposta para visualização de alterações e controle de mesclagem para documentos XML.

4 Gerenciando Alterações em Documentos XML

Como discutido anteriormente, os VCS não consideram informações específicas sobre o formato XML no momento de controlar as versões de tais documentos. Isso gera comportamentos incorretos e, invariavelmente, arquivos XML malformados.

Apesar de existirem algoritmos para cálculo de diferenças e mesclagem de documentos XML, estes não são usados nos VCS. Para solucionar este problema, nossa abordagem consiste em estabelecer uma forma mais adequada para controlar as versões de documentos XML sem prejudicar o controle dos demais arquivos. Para isso, foram considerados os seguintes requisitos: (i) a detecção de diferenças deve ser realizada utilizando um algoritmo que leve em conta o formato específico dos documentos XML; (ii) a mesclagem não deve produzir documentos malformados; (iii) o usuário deve ser capaz de acompanhar as alterações aplicadas durante a mesclagem; (iv) arquivos que não contêm documentos XML devem ser suportados, e tratados da mesma forma como são tratados em VCS tradicionais.

Para atender a estes requisitos, é necessário estabelecer uma forma eficiente de comparar versões e executar mesclagens em versões de documentos XML. É preciso ressaltar que a abordagem deve, de forma transparente, continuar versionando com as técnicas padrão os arquivos que não são documentos XML. Futuramente, outros tipos de arquivos – como arquivos de código fonte – poderiam também receber tratamento diferenciado, mas isto não se encaixa no escopo deste trabalho.

Há diversas formas de se atender aos requisitos propostos. Uma possível estratégia seria desenvolver um sistema que utiliza um dos VCS existentes para guardar os arquivos. Desta forma, ele poderia exibir as diferenças e executar a mesclagem como desejado de acordo com o tipo de arquivo, sempre invocando o cliente do VCS para armazenar ou recuperar as versões. Neste caso, seria preciso implementar toda a interface com o usuário, que em sua maior parte poderia simplesmente atuar como uma ponte entre o usuário e a interface do VCS utilizado, mas que seria necessária para os casos em que é preciso decidir entre o uso do comportamento padrão do VCS e o comportamento especial. Uma outra possibilidade seria implementar todo o sistema, aplicando nele os comportamentos especiais desejados para documentos XML. Isso significa implementar não somente o versionamento dos arquivos XML, como também o versionamento de arquivos texto e dos arquivos binários. Considerando a grande gama de VCS disponíveis, não parece uma boa alternativa duplicar os esforços já realizados e implementar todo um sistema de controle de versões. Esta opção, portanto, não foi considerada.

Uma terceira estratégia seria integrar-se a um VCS existente. A maioria dos VCS permite que *plugins* ou programas externos determinem como a detecção de diferenças e a mesclagem devem ser executadas para um determinado tipo de arquivo. Tais *plugins* poderiam ser escritos para se comportarem de forma diferenciada quando confrontados com arquivos XML. Para outros tipos de arquivos, eles dispariam o comportamento padrão do VCS. Ainda que o VCS não suporte tais *plugins*, se o VCS utiliza o paradigma Cliente/Servidor, é possível alterar apenas o cliente, implementando nele este comportamento especial, sem realizar qualquer alteração no servidor. Desta forma, a abordagem não precisaria lidar com algumas tarefas de grande complexidade, como o armazenamento e a recuperação das versões, que não são relevantes para os requisitos propostos.

Além disso, esta opção ainda permite o uso de VCS consagrados, já adotados em massa pela indústria, amplamente testados e conhecidos, facilitando, portanto, a adoção da abordagem proposta e melhorando sua robustez, sem impor a necessidade de se reimplementar toda a interface com o usuário, como no caso da primeira opção. Assim, esta opção tem clara vantagem sobre as demais, e foi a escolhida.

Para viabilizar essa escolha (ilustrada na Figura 7), optou-se por utilizar os algoritmos existentes para *diff* e *merge* de documentos XML. Deste modo, o principal desafio da abordagem consiste em adaptar tais algoritmos para que o usuário possa visualizar as diferenças entre dois arquivos XML, e decidir como mesclá-los.

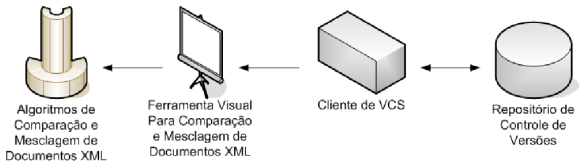

Figura 7. Abordagem proposta

As próximas seções detalham as escolhas realizadas para cada componente da Figura 7.

4.1 O Repositório

Os dois principais VCS livres centralizados disponíveis atualmente são os já citados CVS e Subversion. Ambos são sistemas sólidos, amplamente testados e aprovados, e seriam portanto boas escolhas para representar o papel do repositório na abordagem proposta. No entanto, o Subversion, que é um candidato a substituir o CVS, de fato apresenta vantagens frente a este. Várias operações que dependiam do servidor no CVS – como a visualização das alterações – são operações locais no caso do Subversion. Além disso, o subversion trata arquivos binários de forma mais eficiente que o CVS, tem diversas ou-

tras operações implementadas de forma eficiente (e.g., criação de ramos e etiquetas), controla também a versão de diretórios, enquanto o CVS apenas controla arquivos, dentre outras vantagens. Todas estas vantagens tornam o Subversion a escolha natural para o VCS a ser utilizado neste trabalho, e portanto, ele foi adotado como o repositório.

Vale ressaltar que outras escolhas razoáveis seriam VCS distribuídos, como o Git [5] e o Mercurial [16]. Contudo, no momento da realização dessas escolhas (2007), esses sistemas ainda se apresentavam imaturos e suscetíveis a grandes mudanças arquiteturais. Caso uma nova implementação venha a ser feita no futuro, esses sistemas seriam fortes candidatos a exercer o papel de repositório.

4.2 O Cliente

O cliente padrão do Subversion permite que se configure uma ferramenta externa para executar a detecção de alterações e a mesclagem. Portanto, poderia ser desenvolvida uma aplicação simples que detecta se o arquivo a ser manipulado é um documento XML, e, em caso afirmativo, executa o comportamento especial, ou, caso contrário, utiliza as ferramentas padrão (*diff* e *diff3*). Esta ferramenta seria configurada no cliente padrão do Subversion como a ferramenta de detecção de alterações e de mesclagem.

No entanto, após a análise de outras opções de clientes para o Subversion, foi possível concluir que não é necessário desenvolver tal ferramenta, pois clientes já disponíveis englobam esta funcionalidade, permitindo que se configure uma ferramenta diferente para cada extensão de arquivo.

O TortoiseSVN [4] é um cliente gráfico para o Subversion que se integra ao Windows, permitindo que as operações de versionamento sejam executadas utilizando-se os menus de contexto do Windows Explorer. O TortoiseSVN já engloba ferramentas gráficas para a detecção de alterações e para a mesclagem. Tais ferramentas são práticas para arquivos de texto, mas não consideram as características específicas do XML, como já discutido. Contudo, o programa permite que se configurem outras ferramentas de detecção de alterações e mesclagem para extensões de arquivo específicas. Pode-se, então, utilizar essa funcionalidade para configurar uma ferramenta específica para tratar os arquivos XML, e os outros arquivos não serão afetados pelo comportamento dessa ferramenta.

4.3 Algoritmos

Dentre os algoritmos descritos na Seção 2, o X-Diff não foi considerado por utilizar um modelo de árvores não ordenadas, incompatível com o escopo deste trabalho. Dentre os restantes, o XyDiff se destaca por ter uma baixa complexidade de tempo, e também por ser melhor documentado que o XMLTreeDiff. Deste modo, o XyDiff seria uma boa opção a ser utilizada neste trabalho para obter as diferenças entre duas versões. Um ponto negativo é que o XyDiff não traz qualquer suporte à mesclagem de arquivos, e por isso seria necessário desenvolver um algoritmo à parte para a mesclagem.

Para a mesclagem entre arquivos XML, o 3dm se sobressai como a opção mais interessante. Como vantagens, podem ser citadas: (i) é específico para XML, e portanto, utiliza uma Unidade de Comparação adequada; (ii) é um algoritmo elaborado, capaz de detectar até mesmo movimentações de nós que tiveram descendentes alterados; (iii) possui uma implementação livre já disponível e utilizável.

A principal desvantagem é a completa falta de iteração com o usuário durante a mesclagem. O 3dm resolve todos os conflitos utilizando regras pré-estabelecidas, perdendo a chance de permitir que o usuário decida qual das alterações conflitantes utilizar. Este comportamento se encaixa perfeitamente nos objetivos do 3dm, voltado para ferramentas automáticas de mesclagem, mas impõe obstáculos no contexto deste trabalho. Como discutido anteriormente, dadas 3 versões de um documento – a versão base e duas versões que sofreram alterações de forma independente – o 3dm gera a versão mesclada, gravando-a em um arquivo. Este algoritmo também gera um *log* das alterações aplicadas e uma lista dos conflitos encontrados, como representado no diagrama da Figura 8.a.

Além disso, também é possível utilizar o 3dm para obter as diferenças entre os arquivos. Para tal, basta utilizar a versão base como uma das versões alteradas. Como uma das versões é igual à base, esta

[4] http://tortoisesvn.net

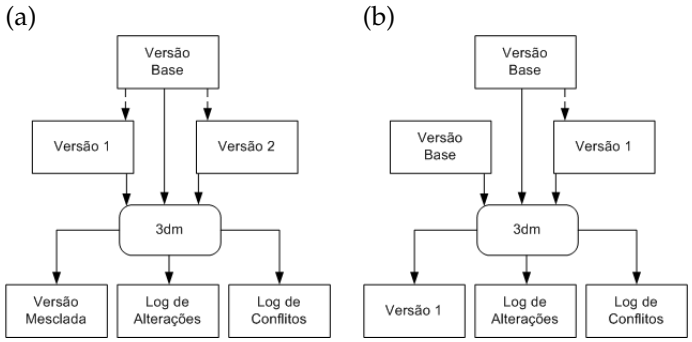

Figura 8. Relação entre as versões durante a mesclagem (a) e relação entre as versões durante a detecção de alterações (b)

versão não implicará em mudança alguma. Todas as alterações serão referentes à outra versão. Desta forma, o log de alterações gerado pelo 3dm conterá as operações que transformam uma versão na outra, exatamente o necessário para exibir as diferenças entre os arquivos. O diagrama da Figura 8.b mostra como as versões são enviadas para o 3dm quando o objetivo é detectar alterações. Note que a versão mesclada gerada pelo 3dm é igual à versão 1, já que esta foi a única que efetuou alterações, e que isso não é um resultado relevante – apenas os *logs* serão utilizados.

Na abordagem proposta, a opção foi por manter o 3dm como uma ferramenta externa, executado de forma transparente pela aplicação. Isso abre a interessante possibilidade de se trocar o 3dm por alguma outra opção no futuro, que gere os *logs* em formatos semelhantes.

A Tabela 3 apresenta uma visão geral dos algoritmos e suas propriedades.

Tabela 3. Visão geral dos algoritmos

Nome	Complexidade	Árv. Ordenada?	Delta Mínimo	Operações
X-Diff	Quadrática	Não	Sim	Básicas
Xy-Diff	Linear	Sim	Não	Básicas, Movimentação
XMLTreeDiff	Quadrática	Sim	Não	Básicas
3dm	Linear	Sim	*	Básicas, Movimentação

Legenda:
* Não Informado

4.4 Ferramenta Visual XMLTreeMerge

Para a visualização de diferenças e controle de mesclagem, não foi possível encontrar nenhuma ferramenta com comportamento aceitável. Portanto, optou-se por desenvolver a XMLTreeMerge – Ferramenta de Exibição de Diferenças e Controle de Mesclagem, para suprir esta importante lacuna. A ferramenta utiliza as versões do documento e os *logs* gerados pelo 3dm, como exibido no diagrama da Figura 8, para apresentar as diferenças para o usuário de uma forma gráfica e intuitiva.

4.4.1 Exibição de Diferenças Uma questão de extrema relevância está em como exibir as diferenças encontradas nos arquivos para que o usuário possa validá-las. A maioria dos programas gráficos com este objetivo, voltados para arquivos texto em geral, exibem 2 (ou 3, no caso de uma mesclagem) painéis lado a lado, um com cada versão, e destacam as linhas que são diferentes entre as versões. Linhas inseridas em uma das versões geralmente produzem um espaço em branco na representação gráfica da

outra versão, para que as partes em comum estejam sempre lado a lado, facilitando a comparação.

Como a estrutura dos arquivos XML é uma árvore e os algoritmos para detecção de diferenças utilizam modelos baseados em árvores, exibir as diferenças em um formato de árvore se mostra uma escolha natural. No entanto, a filosofia de mostrar árvores lado a lado como forma de comparar edições não parece ser a mais adequada. Uma opção mais interessante seria exibir as duas árvores como uma só, onde nós em comum são exibidos normalmente, e os outros nós são exibidos de uma forma especial. Se considerarmos que há uma versão antiga e uma nova, e que a versão nova pode ser gerada a partir de alterações aplicadas sobre a versão antiga, é possível representar os nós diferentes com uma marca que indique qual operação aquele nó sofreu. Um exemplo de como isso pode ser feito é exibido na Figura 9, onde os nós "p2" e "s2" foram removidos, o nó "bah" foi inserido, e o nó "p1a" editado. Uma forma análoga de exibição das diferenças dentro do próprio documento é adotada pelo Microsoft Word em seu controle de revisões.

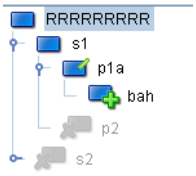

Figura 9. Diferenças entre árvores exibidas como uma única árvore

Da mesma forma, ao executar uma mesclagem, pode-se mostrar de forma comum os nós que pertencem às 3 versões, e marcar na árvore de forma diferente as operações oriundas de cada versão.

4.4.2 Controle das Alterações Ao exibir as alterações detectadas considerando as características específicas dos documentos XML de uma forma adequada, a abordagem já apresenta uma grande vantagem frente ao versionamento atualmente existente para documentos XML. Entretanto, outro ponto fundamental é permitir ao usuário interagir com o documento final, desfazendo ou consertando alterações não adequadas. Por isso, um requisito implementado na XMLTreeMerge foi a possibilidade de edição do documento final. Note que não é objetivo da ferramenta ser um editor completo para XML, mas sim prover algum suporte à edição para permitir pequenos reparos.

Outro ponto importante, no caso da mesclagem, é o tratamento dos conflitos. Nem sempre é possível conciliar as alterações realizadas em cada uma das versões. Por exemplo, se as duas versões mudam o conteúdo textual de um mesmo elemento para valores diferentes, existe uma situação de conflito.

Na abordagem proposta, a mesclagem foi dividida em duas fases: resolução de conflitos e edição. Primeiro, se houver conflitos, eles serão exibidos junto a opções que permitam ao usuário escolher qual das versões deve prevalecer em cada um dos casos. Após esta fase, o documento mesclado, já agregando eventuais mudanças efetuadas pelo usuário durante a fase de resolução de conflitos, é exibido para eventual edição com todas as alterações destacadas, como já discutido.

4.4.3 Exibição das Alterações Para exibir as diferenças, o XMLTreeMerge utiliza diretamente o *log* de alterações gerado pelo 3dm. Inicialmente, ela carrega para uma estrutura em árvore cada uma das versões envolvidas, ou seja, a versão base, as duas versões modificadas (que chamaremos de versão 1 e versão 2, enquanto o 3dm as denomina *branch1* e *branch2*) e a versão mesclada, gerada pelo 3dm. É importante notar que o objetivo não é a construção da versão mesclada, tarefa já executada pelo 3dm, mas sim exibir graficamente as alterações aplicadas para chegar a esta versão mesclada.

Para as **inserções** obtidas do *log*, o XMLTreeMerge procura na árvore mesclada o nó inserido, identificado pelo parâmetro *path*, e o marca como tendo sido inserido. Para **atualizações** obtidas no *log*, o processo é análogo. Procura-se na árvore mesclada o nó alterado e ele é marcado como tendo sido alterado. Para representar **cópias**, o XMLTreeMerge também busca o nó utilizando o *path* e o marca como sendo uma cópia de outro nó. Seria interessante se o log tivesse também o caminho do nó que foi copiado na árvore mesclada, já que essa informação é útil para exibir na interface o nó original de alguma

forma associado à cópia. Para representar uma **remoção**, é necessário inserir o nó que foi removido, juntamente com toda a sub-árvore abaixo do nó. Apesar de aparentar ser contraditório, este procedimento é fundamental, pois o nó foi removido pelo 3dm. Portanto, é necessário inseri-lo novamente para poder marcá-lo como removido. O XMLTreeMerge copia o nó cujo caminho é *src* na versão base, colocando-o como filho de *originNode* na árvore mesclada, e marcando-o como removido. Finalmente, para representar uma **movimentação**, o processo é análogo ao de uma inserção e uma remoção, ou seja, o XMLTreeMerge encontra na árvore mesclada o nó em sua nova posição, representada por *path*, e o marca como o destino de uma movimentação. Em seguida, ela copia esse nó para a sua posição antiga na árvore, identificada por *src*, marcando esse nó como a origem de uma movimentação.

Os documentos apresentados na Figura 5, cujo log de alterações está na Figura 5(d), geram a árvore apresentada na Figura 10. Na árvore, o XMLTreeMerge exibe com ícones diferentes elementos que não sofreram alteração, elementos adicionados, modificados, movidos, copiados e removidos. Os nós removidos – ou que são origem de uma movimentação – aparecem desabilitados, para tornar evidente que eles não estarão realmente presentes no documento final.

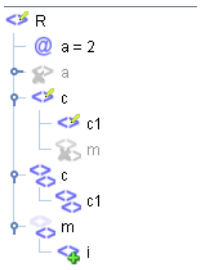

Figura 10. Árvore gerada a partir do log de alterações.

4.4.4 Conflitos Como já discutido, o processo de mesclagem pode gerar conflitos nos casos em que é impossível conciliar alterações das duas versões sem efetuar escolhas que implicam em ignorar mudanças executas por uma delas.

Para alguns dos casos, a resolução do conflito proposta pelo algoritmo 3dm pode ser acertada. Por exemplo, no caso do *Far Move/Near Move*, o 3dm assume que a movimentação local pode ser ignorada em favor da movimentação mais distante. Da mesma forma, o 3dm duplica os nós quando ambos são movidos para locais diferentes. Contudo, mesmo nesses casos, essa pode não ser a intenção do usuário. No entanto, para outros casos, o algoritmo simplesmente adota as alterações da versão 1, o que muito provavelmente não será uma solução aceitável em vários momentos.

A descrição textual fornecida pelo 3dm (Tabela 1) não é adequada para ser interpretada por um programa, já que é direcionada para os usuários. No entanto, a abordagem proposta neste trabalho necessita interpretar esse texto, pois esta é a única forma disponível para diferenciar os tipos de conflitos relacionados a movimentações. Além disso, a descrição do conflito *Locked/Delete* não é precisa (*Moved or copied node deleted. Moving on by allowing the delete*). Na verdade, este conflito não ocorre quando o nó que foi movido em uma versão é apagado em outra, e sim quando um nó que servia de contexto para alguma operação – nós que ajudam a identificar o destino de um outro nó, por exemplo – é removido na outra versão.

4.4.5 Exibição dos Conflitos No XMLTreeMerge, optou-se por mostrar os conflitos encontrados antes de mostrar todas as diferenças, dando ao usuário a opção de escolher qual ação tomar para cada conflito. Somente após a fase de resolução dos conflitos o usuário é levado à tela de visualização das diferenças.

Esse procedimento contrasta com o normalmente utilizado em ferramentas de mesclagem, que mostram todas as alterações, conflitantes ou não, ao mesmo tempo. A decisão de resolver os conflitos um a um anteriormente se deve à escolha da exibição das diferenças em apenas uma árvore. A árvore ficaria confusa se em meio às demais alterações, os conflitos também fossem ali representados. Além disso, exibi-los separadamente é também uma forma de destacar sua importância. Quando uma ferra-

menta que não leva em conta o formato do XML é utilizada, conflitos muitas vezes são gerados por mudanças sem nenhuma relevância, como por exemplo mudanças de formatação no documento. Exagerar a importância deles tornaria o processo de mesclagem cansativo. No caso do XMLTreeMerge, conflitos representam situações que realmente necessitam da ação e intervenção do usuário, então é primordial apresentá-los com bastante destaque. Esse tipo de estratégia, também conhecida como resolução de conflitos interativa [18], também é utilizada por outros sistemas de controle de versão modernos, como o Subversion.

A interface proposta para exibir os conflitos consiste em um painel lateral que exibe a lista de conflitos e alguns detalhes sobre o conflito selecionado na lista. Além disso, 3 árvores são apresentadas, uma com a versão base, e as outras com as duas versões modificadas. Ao selecionar um dos conflitos, o XMLTreeMerge destaca em cada uma das árvores o nó mencionado no conflito, como pode ser visto na Figura 11.

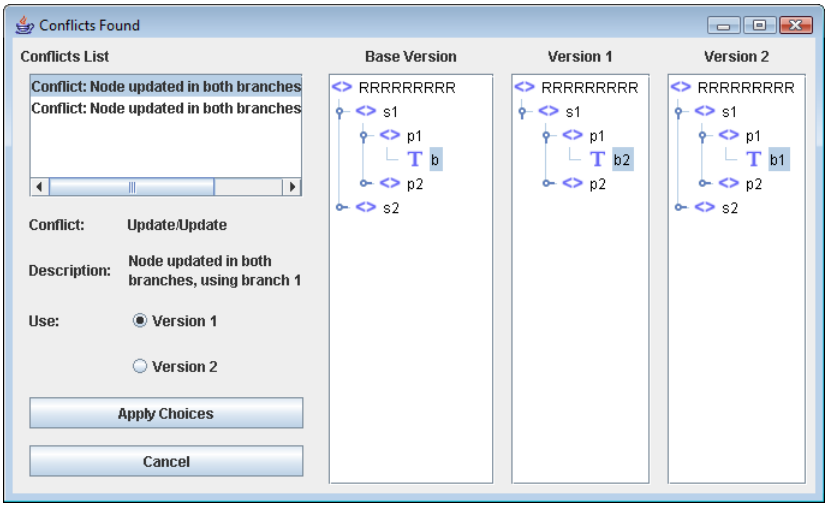

Figura 11. XMLTreeMerge exibindo um conflito *Update/Update*

No caso dos conflitos *Update/Update*, é exibido para o usuário a tela ilustrada na Figura 11. Nela, o usuário pode verificar as alterações que o nó sofreu em cada uma das versões e escolher a alteração mais adequada. Se o usuário escolher a alteração da versão 1, nenhuma ação precisa ser tomada, pois o 3dm já a utilizou. Se o usuário escolher a alteração da versão 2, é preciso aplicar esta alteração. Para tal, é retirado do *log* de conflitos o caminho do nó na árvore mesclada para a sua remoção. Então, o caminho do nó na versão 2 é obtido e este é copiado para a árvore mesclada.

No caso dos conflitos *Near Move/Near Move* e *Far Move/Near Move*, o usuário também pode escolher qual das movimentações ele deseja. Se o usuário selecionar a movimentação utilizada pelo 3dm, o XMLTreeMerge não precisa tomar nenhuma ação. Contudo, se ele seleciona a outra movimentação, o destino originalmente adotado pelo 3dm é marcado como removido, e em seguida o nó em questão é copiado para o novo destino e marcado como movido.

No caso dos conflitos *Far Move/Far Move*, o XMLTreeMerge apresenta a tela exibida na Figura 12. Como a ação padrão do 3dm para esse caso é aplicar as duas movimentações, duplicando o nó, o XMLTreeMerge apresenta uma opção extra: aplicar ambas as operações. Se o usuário escolhe a opção referente a manter as duas alterações, nada precisa ser feito. Se ele escolhe apenas a movimentação de uma das versões, o nó equivalente ao da outra versão é marcado como removido.

No caso dos conflitos *Far Move/Delete*, a ação padrão do 3dm é ignorar a remoção. Caso o usuário decida pelo outro caminho, o nó é recuperado e marcado como removido.

Os conflitos do tipo *Locked/Delete* não fazem sentido para o usuário final, mas com o intuito de tornar a ferramenta mais completa, esse caso também é tratado. Para estes conflitos é exibida a tela apresentada na Figura 13. Neste exemplo, o elemento "c" era utilizado como o contexto do destino da movimentação do elemento "d" em uma das versões, e foi removido na outra versão. A ação padrão

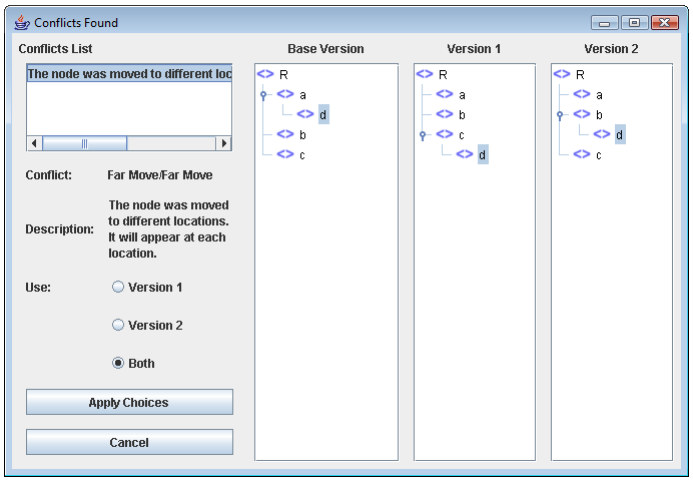

Figura 12. XMLTreeMerge exibindo um conflito *Far Move/Far Move*

do 3dm para o caso é remover o nó. Caso o usuário decida por mantê-lo, o nó é copiado para a árvore novamente.

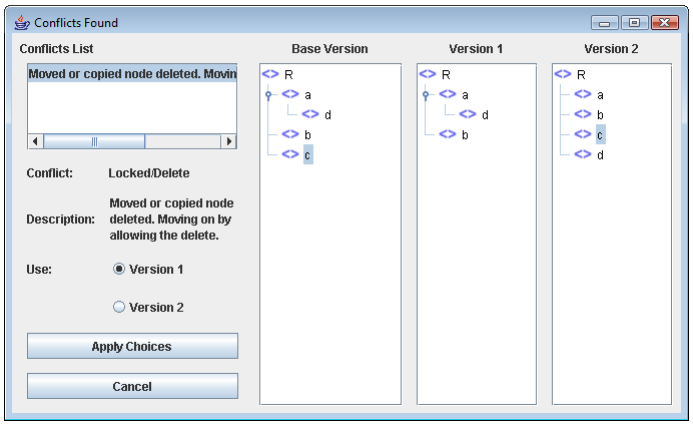

Figura 13. XMLTreeMerge exibindo um conflito *Locked/Delete*

4.4.6 Exibição dos Avisos Os avisos são exibidos juntamente com os conflitos, com a diferença que para alguns casos de aviso não faz sentido escolher qual versão utilizar, visto que as duas versões executaram a mesma alteração, como no caso dos *Equal Updates* e *Equal inserts*. Nestes casos, o XMLTreeMerge apenas exibe o aviso, sem oferecer qualquer opção. Na verdade, esses dois avisos poderiam ser ignorados pela ferramenta, já que não trazem informação útil. Eles foram mantidos no XMLTreeMerge apenas para a completude do tratamento dos conflitos e avisos gerados pelo 3dm.

No caso dos avisos do tipo *Inserts*, no qual nós diferentes foram inseridos no mesmo elemento em cada versão, o 3dm toma como ação padrão aceitar as duas inserções, sendo que as da versão 1 aparecem acima das da versão 2. Poderia fazer sentido para o usuário decidir ignorar as inserções de uma das versões, ou ao menos determinar outra ordem para os elementos. Infelizmente, no *log* gerado pelo 3dm para este caso não há informação suficiente para desfazer a operação executada. Por isso, o XMLTreeMerge também se limita a exibir o aviso, como pode ser visto na Figura 14.

Finalmente, para o último tipo de aviso, o *Delete/Update*, o *log* contém as informações necessárias para desfazer a remoção, que é a ação padrão tomada pelo 3dm para o caso. O usuário tem, então, a opção de escolher entre a versão que removeu a subárvore e a versão que aplicou alterações na subárvore removida, como pode ser visto na Figura 15. Caso o usuário decida pelas alterações, a subárvore em questão é copiada para a versão mesclada.

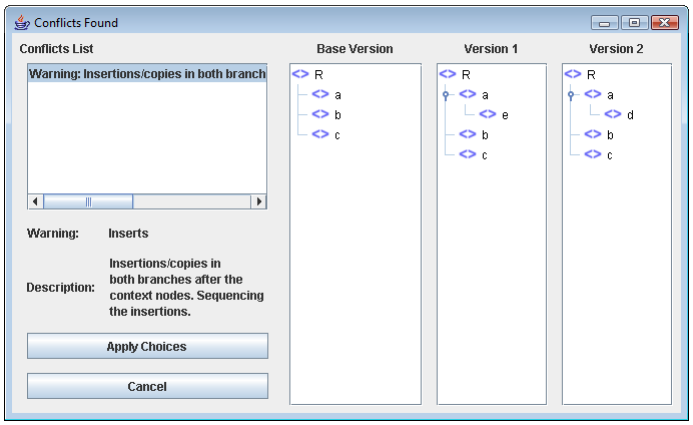

Figura 14. XMLTreeMerge exibindo um aviso *Inserts*

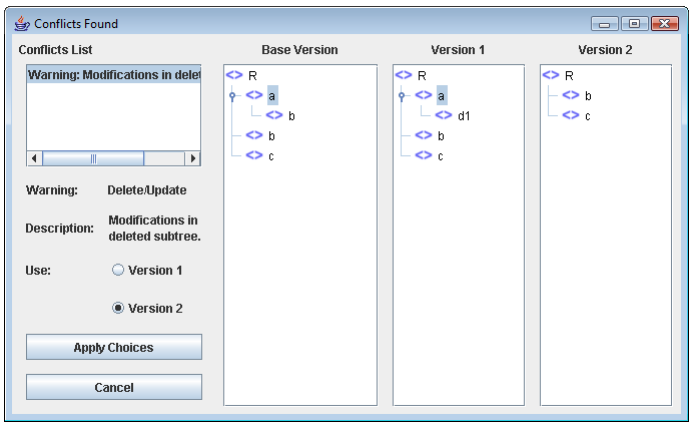

Figura 15. XMLTreeMerge exibindo um aviso *Delete/Update*

4.4.7 Alterações Manuais Durante a exibição das alterações, o XMLTreeMerge também permite que o usuário execute alterações na versão final do documento. Como pode ser visto na Figura 16, há um painel que mostra informações relevantes ao nó selecionado, e que permite a edição do nome, no caso de um elemento; do nome e do valor, no caso de um atributo; e do texto, no caso do conteúdo textual de um elemento.

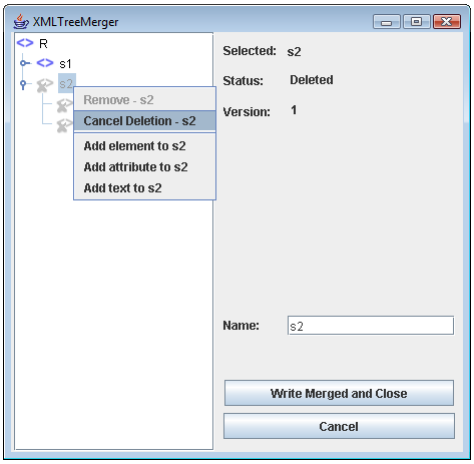

Figura 16. Opções de edição de um elemento

Além disso, é possível remover nós, o que os torna desabilitados, da mesma forma como são exibidos os nós que foram removidos por uma das versões. Também é possível cancelar uma remoção, tenha ela sido feita manualmente, usando o XMLTreeMerge, ou executada por uma das versões, fazendo com que o nó volte a constar no resultado. Finalmente, é possível mover os nós, utilizando a funcionalidade de *drag and drop*, ou seja, arrastando os nós, como exibido na Figura 17.

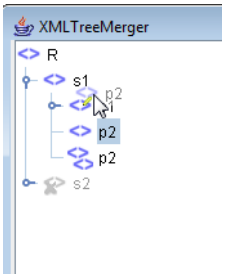

Figura 17. *Drag and drop* dos nós na árvore de visualização das diferenças.

Após as mudanças desejadas, se o usuário clicar no botão *Write Merged and Close*, o programa gravará o arquivo resultante na saída passada como parâmetro – o quarto parâmetro – e encerrará. Se o programa foi chamado como ferramenta de mesclagem por um sistema de controle de versões, isso significa que este arquivo gravado passará a ser considerado como o arquivo que foi mesclado.

4.4.8 Alterações no 3dm Foi necessário realizar duas alterações no 3dm para viabilizar o funcionamento do XMLTreeMerge. Quando as duas versões alteravam os atributos de um elemento sem ocasionar um conflito – por exemplo, se duas versões inserem atributos diferentes – nenhum *log* de alteração era gerado. O arquivo era gerado corretamente, com as alterações devidamente aplicadas, porém não havia registro no *log* referente à atualização daquele elemento. Como o formato especificado para o *log* de alteração não contemplava entradas com referências concomitantes a duas versões, optamos por alterar o 3dm para que ele gerasse duas entradas separadas no *log*, uma para cada versão. Esta foi a forma menos intrusiva para resolver o problema.

Além disso, ao detectar um conflito do tipo *Far Move/Far Move*, o algoritmo executava algumas verificações extras, e, apesar de apresentar o comportamento descrito para o conflito neste caso, ou seja, aceitar as duas alterações, não registrava essa situação como conflito. No caso do 3dm, cujo objetivo era funcionar sem iteração com o usuário, deixar de registrar esse conflito já poderia ser considerado um problema grave. Porém, é ainda mais grave para uma ferramenta que se propõe a utilizar as escolhas do usuário. Neste caso, é preferível lidar com falsos positivos a deixar de detectar um conflito. Portanto, esta verificação automática foi desabilitada.

5 Trabalhos Relacionados

Existem algumas ferramentas de visualização de diferenças entre documentos XML. O DiffDog [1] é uma ferramenta proprietária de visualização de diferenças e mesclagem de documentos. A ferramenta permite para comparar arquivos, diretórios e, especialmente, documentos XML. A ferramenta apresenta os documentos em formato de texto, realçando as diferenças, e cria uma ligação visual entre os elementos casados em uma versão e outra do documento XML. Uma desvantagem do DiffDog diz respeito a portabilidade (disponível somente para Windows).

O XSDelta [17] é uma ferramenta gráfica capaz de comparar versões de esquemas XML e que utiliza o algoritmo de detecção de XyDiff. O XSDelta expõe ao usuário todas as operações envolvidas na transformação de um esquema original em sua nova versão. O XSDelta é capaz de exibir textualmente os esquemas, realçando através de cores distintas as partes modificadas e identificando as mudanças de esquema realizadas. Como saída, o XSDelta gera arquivos XML contendo as diferenças identificadas, o que auxilia na gerência do processo de evolução de esquemas XML. Como limitações, o XSDelta não permite a edição de documentos nem apresenta a opção de visualização dos casamentos.

O XVersion [10] é uma ferramenta gráfica de comparação entre arquivos XML e que é capaz de verificar as alterações que ocorreram entre duas versões de documentos XML, além de agrupar um conjunto de documentos XML em um novo documento. O algoritmo para detecção de diferenças utilizado é o XyDiff. A ferramenta apresenta três estruturas principais: um comparador de documentos XML, um agrupador de documentos XML e uma função de consulta, todas integradas por uma interface que é utilizada para abrir ou salvar documentos XML. Entretanto, a ferramenta não permite, por exemplo, a navegação entre as diferenças.

O VisualX [4] é uma interface gráfica, que interpreta e apresenta as diferenças encontradas pela execução de um algoritmo de diff para XML. Sua interface apresenta duas versões de um documento XML, realçando, através de cores, as modificações detectadas pelo algoritmo. A ferramenta dá suporte aos algoritmos de diferença: XyDiff, JXyDiff, XKeyMatch; permitindo ao usuário escolher qual dos algoritmos deseja utilizar. Uma limitação da ferramenta é a não apresentação visual dos casamentos realizados pelo algoritmo de detecção.

A Tabela 4 apresenta um comparativo entre estas ferramentas. A ferramenta XMLTreeMerge, que é o objeto deste artigo, oferece diversas funcionalidades encontradas nas demais ferramentas, bem como efetua a mesclagem dos documentos XML e a visualização dos casamentos, características estas encontradas somente na ferramenta comercial DiffDog.

Tabela 4. Ferramentas de Visualização de Diferenças

	DiffDog	VisualX	XSDelta	XMLTree Merge	XVersion
Algoritmos	*	XyDiff, JXyDiff, XKey-Match	XyDiff	3dm	JXyDiff
Portabilidade	Windows	Linux, Windows	Linux, Windows	Linux, Windows	Linux, Windows
Merge	Sim	Não	Não	Sim	Não
Diff	Sim	Sim	Sim	Sim	Sim
Diferenciação por cores	Sim	Sim	Sim	Não	Não
Navegação por diferença	Sim	Sim	Sim	Sim	Não
Edição do Documento	Sim	Sim	Não	Não	Sim
Linguagem	**	Java	Java	Java	Java
Distribuição	Comercial	Acadêmica	Acadêmica	Acadêmica	Acadêmica
Visualização dos Casamentos	Sim	Não	Não	Sim	Não
Referência	[1]	[4]	[17]		[10]

Legenda:
* Algoritmo proprietário
** Não Informada

6 Avaliação

A abordagem XMLTreeMerge descrita neste artigo, foi alvo de uma avaliação envolvendo 10 voluntários, alunos do curso de Ciência da Computação da Universidade Federal de Juiz de Fora (UFJF). Para participar, os voluntários deveriam basicamente possuir experiência em algum sistema de controle

de versão. Não houve nenhum tipo de compensação para os participantes. Duas ferramentas foram utilizadas neste estudo: XMLTreeMerge e TortoiseSVN.

Os alunos foram divididos em dois grupos (Grupo 1 - P1 a P5 e Grupo 2 - P6 a P10), com base no formulário de caracterização preenchido anteriormente. O grupo 1 utilizou na primeira etapa a ferramenta XMLTreeMerge para detectar as diferenças e resolver os conflitos e na segunda etapa, o TortoiseSVN. O grupo 2 utilizou na primeira etapa o TortoiseSVN e na segunda, a XMLTreeMerge.

Na situação proposta, o participante foi informado que havia sido convidado por seu professor orientador a participar do II Workshop de Trabalhos de Graduação e Pós Graduação (WTGPG) da UFJF. Para isso deveria submeter um resumo relacionado aos resultados obtidos até então em seu trabalho de iniciação científica e que deveria começar esta tarefa imediatamente, editando um documento XML que continha o *template* do resumo, disponibilizado para o Workshop.

Inicialmente, o participante foi informado sobre o estudo por meio de um formulário de consentimento. Caso concordasse em participar, ele preenchia um questionário de caracterização, que avaliava o nível de conhecimento e experiência do participante em diferentes temas relacionados ao estudo, como, por exemplo, desenvolvimento de software, XML e sistemas de controle de versão. Estas informações foram utilizadas para garantir que os participantes estavam aptos a executar o estudo de avaliação; na ditribuição dos participantes nos grupos que usariam primeiro uma ferramenta e depois a outra; e na interpretação dos resultados obtidos.

Após o preenchimento do questionário de caracterização, foi feita ao participante uma apresentação de aproximadamente 5 minutos sobre Gerência de Configuração de Software (GCS), além da definição de alguns conceitos que seriam utilizados posteriormente nos documentos fornecidos. Em seguida, foi entregue ao participante o documento de descrição geral da tarefa e a especificação de sua tarefa neste estudo. A partir desse momento, o participante dava início à execução da tarefa, usando para isso, sua cópia local do documento XML.

Ao terminar a edição do documento, o participante era informado que seu orientador também estava escrevendo parte do resumo e que já havia submetido a versão atualizada para o repositório central. O participante então deveria fazer o mesmo, ou seja, enviar as modificações efetuadas em sua cópia local para o repositório. Neste caso, ao fazer a atualização, o participante precisaria resolver alguns conflitos, em função das mudanças efetuadas por seu orientador, a partir de uma das duas ferramentas indicadas para o estudo. Em seguida, era disponibilizado ao participante o documento de avaliação da tarefa executada que deveria ser preenchido.

Logo após, o participante deveria executar a mesma tarefa efetuada na primeira etapa, sendo que a outra ferramenta seria utilizada ao final deste processo para identificar as diferenças entre os documentos, resolver os conflitos e efetuar a mesclagem correspondente. Concluída essa segunda etapa, o participante preenchia o questionário de avaliação da segunda tarefa e em seguida o formulário de avaliação comparativa, por meio do qual deveria expressar sua opinião a respeito das tarefas executadas no estudo.

A Tabela 5 apresenta um resumo da caracterização desses participantes. Para algumas questões, os participantes deveriam indicar se tinham experiência na indústria (I) ou apenas no curso (C). Sobre os sistemas de controle de versão utilizados, adotou-se Hg para Mercurial, SVN para Subversion e SSafe para SourceSafe. Como se pode observar, os voluntários envolvidos tinham em média 3 anos de experiência em desenvolvimento de sistemas, trabalhavam na indústria em sua maioria, e todos conheciam o Subversion.

A Tabela 6 mostra o tempo gasto pelos participantes do grupo 1 na execução de cada uma das 2 etapas. Conforme mencionado anteriormente, o grupo 1 utilizou na primeira etapa a ferramenta XMLTreeMerge e na segunda, o TortoiseSVN. A Tabela 7 apresenta o tempo gasto pelos participantes do grupo 2 na execução das 2 etapas (Etapa 1 - TortoiseSVN e Etapa 2 - XMLTreeMerge).

Vale mencionar que esse tempo não leva em consideração o tempo gasto com as fases iniciais do estudo como apresentação de GCS e preenchimento dos formulários. Em média, o tempo total deste estudo, para cada participante, foi de 55 minutos. Observa-se que em função da mesma tarefa ter sido realizada duas vezes (edição do documento XML, resolução de conflitos), todos os participantes cum-

Tabela 5. Caracterização dos Participantes

	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10
Experiência Desenvolv.	I	I	I	I	C	C	I	I	C	I
Experiência (anos)	5	2	6	3	2	1	3	4	2	5
Tamanho Máx. Equipe	4	30	5	3	-	-	6	20	-	8
Experiência em XML	C	I	I	C	C	C	C	C	C	I
Experiência Modelagem	I	C	I	I	C	C	I	C	C	I
Experiência em VCS	C	I	I	I	C	C	I	I	C	I
VCS Usados	SVN, Git, Hg	SVN, Hg	SVN, Hg, Git	SVN	SVN, Hg	SVN	SVN, Hg	SVN, Hg, SSafe, Git	SVN, Git, Hg	SVN, Hg

Tabela 6. Tempo de Execução das Tarefas do Grupo 1, em minutos

Participante	Etapa 1	Etapa 2	Tempo Total
P1	35	13	48
P2	26	15	41
P3	29	10	39
P4	49	23	72
P5	45	22	67
Média	36,8	16,6	
Desvio Padrão	8,91	5,08	

Tabela 7. Tempo de Execução das Tarefas do Grupo 2, em minutos

Participante	Etapa 1	Etapa 2	Total
P6	43	24	67
P7	40	17	57
P8	46	13	59
P9	40	25	65
P10	24	12	36
Média	38,6	18,2	
Desvio Padrão	7,63	5,42	

priram a segunda etapa em um menor tempo do que o que foi gasto na primeira, independente da ferramenta utilizada.

A Tabela 8 apresenta um resumo da avaliação dos participantes sobre as ferramentas. Inicialmente, cada participante avaliou isoladamente a ferramenta usada em cada etapa e deveria responder, por exemplo, se ficou satisfeito com o resultado final das tarefas (Q1 - XMLTreeMerge e Q2 - TortoiseSVN) e se utilizaria tal ferramenta para exibição de diferenças e apoio à mesclagem de arquivos XML (Q3 - XMLTreeMerge e Q4 - TortoiseSVN). Sobre a avaliação comparativa das ferramentas (T - TortoiseSVN e X - XMLTreeMerge), os participantes deveriam indicar, entre outros pontos, qual das ferramentas apresentava as diferenças entre os arquivos, no caso de detecção de conflitos, de forma mais intuitiva (Q5) e qual das ferramentas apresentava um processo de mesclagem mais controlado e menos propenso a erros (Q6).

Estes estudos preliminares foram executados a fim de analisar a viabilidade e a aplicação da abordagem proposta neste trabalho de pesquisa no que diz respeito à detecção de diferenças e mesclagem de

Tabela 8. Avaliação das Ferramentas

Part.	Q1	Q2	Q3	Q4	Q5	Q6
P1	Sim	Parcialmente	Sim	Parcialmente	X	X
P2	Sim	Parcialmente	Sim	Não	T	X
P3	Sim	Sim	Sim	Sim	X	X
P4	Parcialmente	Sim	Sim	Sim	T	X
P5	Sim	Sim	Sim	Parcialmente	X	X
P6	Parcialmente	Sim	Sim	Sim	T	T
P7	Parcialmente	Parcialmente	Sim	Sim	X	X
P8	Sim	Sim	Sim	Sim	X	X
P9	Sim	Sim	Sim	Sim	X	T
P10	Sim	Sim	Sim	Sim	X	T

Legenda:
Q1: Você ficou satisfeito com o resultado final das tarefas? (ref. XMLTreeMerge)
Q2: Você ficou satisfeito com o resultado final das tarefas? (ref. TortoiseSVN)
Q3: Você utilizaria a XMLTreeMerge para exibição de diferenças e apoio à mesclagem de arquivos XML?
Q4: Você utilizaria a TortoiseSVN para exibição de diferenças e apoio à mesclagem de arquivos XML?
Q5: Qual das ferramentas apresenta as diferenças entre os arquivos de forma mais intuitiva?
Q6: Qual das ferramentas apresenta um processo de mesclagem mais controlado e menos propenso a erros?
Nas questões Q5 e Q6, as respostas possíveis eram T - TortoiseSVN ou X - XMLTreeMerge

documentos XML. A leitura dos resultados obtidos e da avaliação dos participantes ajudou a entender alguns pontos que precisam ser melhorados na abordagem, assim como outros benefícios que podem ser alcançados com a sua utilização. No entanto, devido às restrições deste estudo (como por exemplo, número de voluntários e perfil dos envolvidos), estes resultados não podem ser generalizados.

De acordo com os envolvidos, a XMLTreeMerge cumpre, de modo satisfatório sua finalidade e, quando comparada com a ferramenta TortoiseSVN, os voluntários, em sua maioria afirmaram que ela era a mais adequada para a tarefa proposta (independente da ordem em que a tarefa foi cumprida). Como justificativas para esta escolha, foram apresentadas por exemplo: o fato da ferramenta ser específica para documentos XML; a possibilidade de visualizar o arquivo base e as duas versões modificadas do arquivo, o que facilita o processo de mesclagem; e a visualização utilizando a estrutura em árvore, que é mais adequada que a visualização textual utilizando cores.

De acordo com alguns participantes, a XMLTreeMerge aqui proposta, executa a mesclagem dos documentos e exibe as diferenças de uma forma bem mais intuitiva, como pode ser visto na Figura 18.

Utilizando o TortoiseSVN, alguns voluntários concluiram que, o desenvolvedor não tem uma visão muito intuitiva do que aconteceu, já que as alterações na ordem tornam a visualização extremamente confusa, como apresentado na Figura 19.

7 Considerações Finais

Este artigo mostrou como documentos XML podem ter um tratamento diferenciado em um VCS, obtendo como vantagens uma exibição mais adequada das alterações e um processo de mesclagem mais controlado, mais confortável, e menos propenso a erros. Essas características têm o potencial de aumentar a produtividade e a segurança no uso do VCS, já que não serão executadas mesclagens automáticas que gerem arquivos malformados. Além disso, a infraestrutura oferecida ao usuário visa ajudá-lo a executar uma mesclagem mais segura.

O artigo também mostrou como uma interface voltada especificamente para documentos XML traz vantagens frente a uma interface voltada para arquivos de texto em geral. Mostrou também como o uso de algoritmos já existentes na literatura para detecção de alterações e mesclagem de documentos XML apresenta vantagens frente aos algoritmos usualmente utilizados pelos VCS, por se basearem em

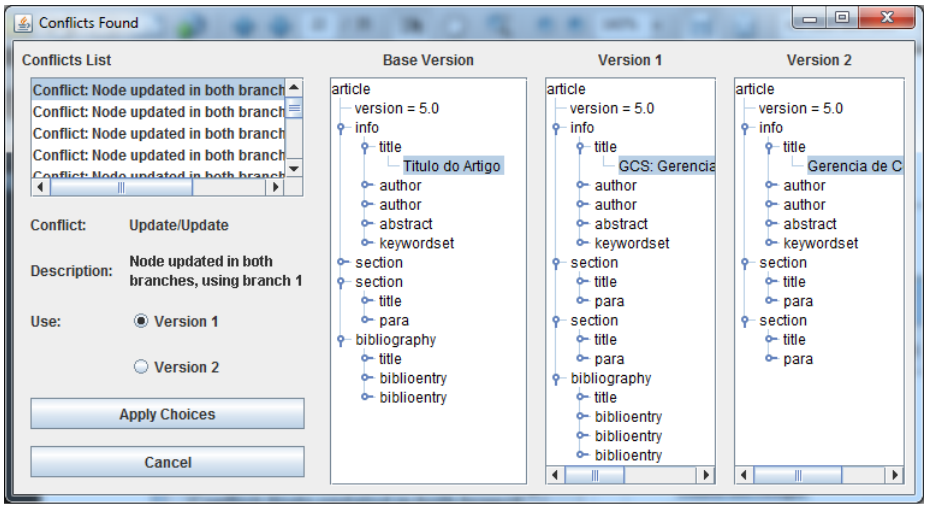

Figura 18. XMLTreeMerge: Exibição de diferenças

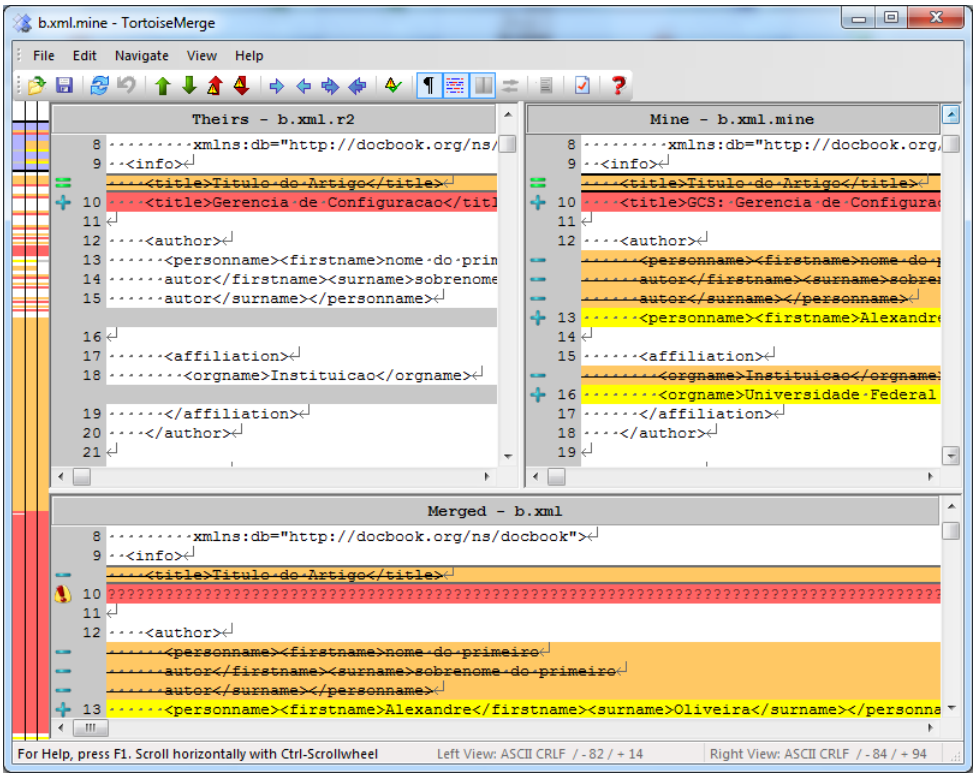

Figura 19. TortoiseSVN: Exibição de diferenças

características específicas do formato dos arquivos em sua concepção.

Ao decidir pela integração ao *Subversion* e ao *TortoiseSVN*, já conhecidos, testados e adotados em massa pela indústria, o trabalho gera uma abordagem mais robusta e de fácil adoção, pois quem já os utiliza pode facilmente testar a abordagem em seus repositórios. A abordagem proposta ainda permite que sejam desenvolvidas ferramentas para exibição de alterações e controle de mesclagem para outros tipos de arquivos, que utilizem algoritmos mais adequados a cada situação.

Infelizmente, o 3dm impõe algumas limitações ao objetivo proposto. As principais se referem ao fato da declaração de DTD e comentários serem removidos dos arquivos XML pelo 3dm. Como alguns

dos principais beneficiários de uma ferramenta neste estilo seriam desenvolvedores de software, e os arquivos XML utilizados por eles frequentemente contêm declarações de DTD e comentários que podem ser considerados de suma importância, trata-se de uma grave limitação. No entanto, como destacado por Lindholm [12], trata-se de uma limitação da implementação criada por ele, e não do algoritmo. É importante destacar que esta implementação pode ser facilmente substituída por outra que não sofra desta limitação.

O 3dm também não executa uma boa mesclagem em alguns casos. Lindholm [12] cita exemplos de casos que fazem o algoritmo entrar em repetição infinita, e outros para os quais o casamento dos nós não acontece como seria natural prever. Portanto, resultados inesperados ocorrem.

Também é importante destacar que quando as alterações efetuadas são numerosas ou de grande impacto, a visualização escolhida não se mostra confortável. No entanto, esta questão não é exclusiva da abordagem proposta, visto que as ferramentas atuais também sofrem do mesmo problema.

Como já citado, outros tipos de arquivo também podem se beneficiar de uma abordagem específica, e estudar tais abordagens para outros formatos seria um trabalho futuro interessante. Por exemplo, uma abordagem específica para arquivos de código Java poderia detectar corretamente movimentações de métodos, e ignorar mudanças de formatação, evitando que as alterações realmente importantes sejam exibidas em meio a uma grande quantidade de mudanças sem nenhuma relevância.

Outra possibilidade, ainda considerando os documentos XML, é melhorar a forma como o histórico de alterações é disponibilizado ao usuário, aplicando também a este caso algoritmos e ferramentas específicos para XML. Por exemplo, seria de grande valia poder executar consultas quanto à evolução de um elemento específico ao longo das diversas revisões de um documento. Um desenvolvedor poderia usar este histórico especial para, por exemplo, descobrir quando uma certa opção que causou problemas foi adicionada, ou mesmo quais valores esta opção já teve ao longo do tempo de vida do documento, sem precisar analisar manualmente diversas versões que não executaram alterações especificamente neste elemento.

Agradecimentos

Gostaríamos de agradecer ao CNPq e à FAPERJ pelo financiamento parcial deste trabalho.

Referências

[1] Altova. Diffdog. disponivel em: <http://www.altova.com/diffdog/diff-merge-tool.html>, 2011.

[2] B. Berliner. CVS II: Parallelizing software development. In *USENIX Conference*, pages 341–352, Berkeley, CA, 1990. USENIX Association.

[3] Tim Bray, Eve Maler, François Yergeau, C. M. Sperberg-McQueen, and Jean Paoli. Extensible markup language (XML) 1.0 (fourth edition). W3C recommendation, W3C, August 2006. Disponível em: <http://www.w3.org/TR/2006/REC-xml-20060816>. Acesso em Dezembro, 2007.

[4] Frantchesco Cecchin and Carmem Hara. Uma interface gráfica para algoritmos de detecção de diferenças entre documentos xml. *Escola Regional de Banco de Dados (ERBD)*, 2009.

[5] Scott Chacon. *Pro Git*. Apress, 2009.

[6] Grégory Cobena, Talel Abdessalem, and Yassine Hinnach. A comparative study for XML change detection . In *Bases de Données Avancées (BDA)*. INRIA, 2002.

[7] Ben Collins-Sussman. The subversion project: buiding a better cvs. *Linux Journal*, 2002(94):3, 2002.

[8] Reidar Conradi and Bernhard Westfechtel. Version models for software configuration management. *ACM Comput. Surv.*, 30(2):232–282, 1998.

[9] Francisco Curbera and David Epstein. Fast difference and update of XML documents. In *XTech*, San Jose, California, March 1999.

[10] Felipe dos Santos Giacomel. Xversion - uma ferramenta gráfica para gerenciamento e consulta de versões de documentos xml, 2006.

[11] Alexis Leon. *A guide to software configuration management*. Artech House, Inc., Norwood, MA, USA, 2000.

[12] Tancred Lindholm. A 3-way merging algorithm for synchronizing ordered trees — the 3DM merging and differencing tool for XML. Master's thesis, Helsinki University of Technology, Dept. of Computer Science, September 2001. Disponível em: <http://www.cs.hut.fi/~ctl/3dm/thesis.pdf>. Acesso em Dezembro, 2007.

[13] Tancred Lindholm. A three-way merge for xml documents. In *Symposium on Document Engineering, ACM*, pages 1–10, New York, NY, USA, 2004. ACM.

[14] Amélie Marian, Serge Abiteboul, Gregory Cobena, and Laurent Mignet. Change-centric management of versions in an xml warehouse. In *International Conference on Very Large Data Bases, VLDB*, pages 581–590, San Francisco, CA, USA, 2001. Morgan Kaufmann Publishers Inc.

[15] Leonardo Murta, Hamilton Oliveira, Cristine Dantas, Luiz Gustavo Lopes, and Cláudia Werner. Odyssey-scm: An integrated software configuration management infrastructure for uml models. *Science of Computer Programming*, 65(3):249 – 274, 2007. Special Issue on: Software Configuration Management (SCM).

[16] Bryan O'Sullivan. *Mercurial: The Definitive Guide*. O'Reilly Media, 2009.

[17] Augusto Belotto Perini, Vincent Nelson Kellers da Silveira, and Renata Galante. Xsdelta: Uma ferramenta visual para comparação de esquemas xml. *SBBD (Brazilian Symposium on Databases)*, 2006.

[18] C Pilato, Ben Collins-Sussman, and Brian Fitzpatrick. *Version Control with Subversion*. O'Reilly Media, Inc., 2008.

[19] Randy Smith. Gnu diff3 version 2.8.1, 2002. Disponível em: <http://www.gnu.org/software/diffutils/>. Acesso em Dezembro, 2007.

[20] Kuo-Chung Tai. The tree-to-tree correction problem. *Journal of the ACM, JACM*, 26(3):422–433, 1979.

[21] Walter F. Tichy. Design, implementation, and evaluation of a revision control system. In *international conference on Software engineering, ICSE*, pages 58–67, Los Alamitos, CA, USA, 1982. IEEE Computer Society Press.

[22] Yuan Wang, David J. DeWitt, and Jin-Yi Cai. X-diff: an effective change detection algorithm for xml documents. In *International Conference on Data Engineering*, pages 519–530, 2003.