AGRUPAMENTO E CONSULTA À VERSÕES DE DOCUMENTOS XML EM UM AMBIENTE *PEER-TO-PEER*

Deise de Brum Saccol[1], Felipe dos Santos Giacomel, Renata de Matos Galante, Nina Edelweiss

Instituto de Informática - Universidade Federal do Rio Grande do Sul
UFRGS - Porto Alegre – RS – Brasil

{deise, fsgiacomel, galante, nina}@inf.ufrgs.br

Abstract. O conceito de versão é bastante explorado no gerenciamento de configuração de *software*. Técnicas de versionamento também têm sido utilizadas no gerenciamento de documentos XML. No entanto, a característica da distribuição em um ambiente *peer-to-peer* (P2P) coloca alguns novos desafios à área de versionamento. Documentos versionados podem estar armazenados em um único *peer* ou espalhados por vários *peers*. Neste contexto, gerenciar e acessar versões de documentos é de fundamental importância, principalmente para possibilitar recuperação histórica de documentos em um ambiente distribuído. Este trabalho propõe um mecanismo de agrupamento e consulta à versões de documentos XML, inserido na arquitetura do *DetVX*, um ambiente para detecção de réplicas e versões XML em um contexto P2P.

Palavras-Chave: XML, Peer-to-Peer, Versionamento.

1. Introdução

O paradigma P2P baseia-se nos conceitos de descentralização e compartilhamento de recursos, objetivando evitar gargalos em servidores centrais e distribuir a carga de trabalho [1]. Cada *peer* do sistema atua tanto como cliente quanto como servidor, e fornece parte dos recursos e informações disponíveis no sistema. Embora a idéia de compartilhamento seja comum a todos os sistemas P2P, a diferença entre eles reside, principalmente, no tipo de arquitetura adotada [19].

A arquitetura *parcialmente centralizada* (ou *híbrida*) contém um servidor central responsável pelo mecanismo de busca e pela manutenção da infra-estrutura; o compartilhamento de recursos fica sob a responsabilidade dos *peers*. *Napster*[2] é um exemplo de sistema que pertence à este tipo de arquitetura. O segundo tipo de arquitetura, chamada *pura* ou *distribuída*, apresenta um funcionamento totalmente descentralizado,

[1] Este trabalho é parcialmente financiado pelo CNPQ (processo 142396/2004-4), CAPES (processo 1451/06-5) e projeto PERMXML (CNPq 475743/04-0, Edital CNPq 19/2004 – Universal).

[2] http://www.napster.com

onde cada *peer* é responsável pelo mecanismo de busca e pela manutenção da infra-estrutura. *Gnutella*[3] é um exemplo que utiliza esta arquitetura. O último tipo de arquitetura agrega um subconjunto de *peers* em *super peers*. Neste tipo de arquitetura, a comunicação é estabelecida somente entre os *super peers*, e destes com os seus *peers* agregados, o que pode contribuir para tornar a pesquisa mais rápida. *Kazza*[4] implementa esta arquitetura.

Independente da arquitetura utilizada, um problema que pode ser observado nestes ambientes advém do comportamento evolutivo e dinâmico de alguns recursos disponíveis na rede P2P. Documentos XML compartilhados podem sofrer atualizações de conteúdo e estrutura, criando vários estados possíveis do mesmo recurso ao longo do tempo. A característica de passar por atualizações é um aspecto fundamental de qualquer sistema de informações persistentes. No contexto da pesquisa em banco de dados, o gerenciamento de tempo tem sido bastante estudado nas últimas décadas [2], [10], [11]. Muitos trabalhos foram desenvolvidos para adicionar tempo à modelos de Banco de Dados e fornecer capacidades de armazenamento, consulta e atualização de dados históricos [8], [9], [13].

A característica de temporalidade em documentos XML compartilhados em ambientes P2P pode ser tratada com o uso de versões. Através do uso de versões, pode-se preservar o conteúdo antigo e o conteúdo atual do documento, possibilitando representação da evolução temporal e acesso aos estados passados do documento. No entanto, em um ambiente P2P, versões de documentos são disponibizadas na rede em arquivos físicos separados, os quais podem estar armazenados em um único *peer* ou em *peers* diferentes. Se o usuário submete uma consulta a um *peer* específico solicitando o endereço atual de uma pessoa, o sistema deve retornar somente a última versão desta informação. Supondo que os dados existentes no *peer* onde a consulta foi submetida estejam desatualizados, então a consulta deve ser roteada para outros *peers*. Um outro caso é o usuário submeter consultas que retornam o histórico de documentos. Uma vez que as versões podem estar espalhadas em vários *peers*, recuperar o histórico de documentos implica em executar consultas distribuídas pela rede.

Para evitar o roteamento de consultas desta natureza, este artigo propõe *XVersion*, uma ferramenta para agrupamento e consulta à versões de documentos XML. O mecanismo proposto consiste em agrupar versões de documentos XML em um único arquivo que contém todo o histórico de modificações do documento. Desta forma, um único arquivo pode ser acessado para recuperar toda a história de um determinado recurso. A implementação desta ferramenta está inserida dentro do *DetVX*, um ambiente para detecção de réplicas e versões de documentos XML em contextos P2P.

A seção 2 do artigo apresenta os trabalhos relacionados. A seção 3 apresenta o ambiente *DetVX*, contexto onde o artigo está inserido. A seção 4 apresenta o mecanismo proposto para agrupamento de documentos XML, visando recuperação histórica de recursos espalhados em uma rede P2P. O protótipo da ferramenta é apresentado na seção 5. Conclusões e perspectivas futuras são apresentadas na seção 6.

2. Trabalhos Relacionados

Gerenciamento de tempo tem sido bastante estudado nos últimos anos [13]. Uma possível abordagem para tratar o aspecto evolutivo é através do uso de versões [4]. Algumas técnicas de versionamento foram propostas para gerenciamento de tempo de transação em

[3] http://www.gnutella.com
[4] http://www.kazaa.com

bancos de dados temporais [11], versões de artefatos em bancos de dados orientados a objetos [10] e gerenciamento de mudanças em dados semi-estruturados [23]. O conceito de versão também já é bem conhecido no gerenciamento de configuração de *software* [7],[15]. No entanto, sistemas tradicionais de controle de versões modelam documentos como uma sequência de linhas de texto e utilizam *scripts* para representar diferenças entre versões. Estes sistemas mostram-se inadequados para representar versões de documentos XML [4],[14], uma vez que não preservam a estrutura lógica do documento original [3] e não suportam consultas complexas.

Desta forma, alguns modelos foram propostos para representar o caráter evolutivo de dados XML [20]. A proposta de [32] apresenta algumas técnicas para gerenciamento e consultas temporais de documentos multiversionados. A proposta consiste em representar as sucessivas versões de um documento XML como um outro documento XML que implementa um modelo de dados temporal. A abordagem utiliza algoritmos *diff* para processar os tempos de validade dos elementos no documento [33]. Por fim, a proposta utiliza *XQuery* para formular consultas temporais nos documentos, a partir do uso de funções temporais que podem ser escritas pelo usuário.

Outros trabalhos incluem: em [24] os autores propõem o uso de uma *tag* de marcação *valid* para documentos XML/HTML, a fim de possibilitar visualização temporal em *browsers* utilizando-se XSLT. Um método baseado em dimensões é proposto em [25] para gerenciar mudanças em documentos XML, mas não é mostrado como as consultas temporais poderiam ser tratadas. Em [26] e [27] também é proposto um modelo de dados temporal para representar documentos XML; o suporte à consultas é feito via extensões à API DOM e à linguagem *XPath*. O trabalho de [28] propõe uma outra extensão da linguagem *XPath* para possibilitar consultas temporais. Em [21] é proposta a linguagem tXQuery, uma extensão à linguagem *XQuery* com suporte à temporalidade. O trabalho de [29] propõe uma técnica de versionamento baseada em rótulos temporais e números de nós duráveis para preservar a estrutura e o histórico dos documentos durante a evolução. [14] e [30] propõem um mecanismo para possibilitar consultas baseadas em expressões de caminho para documentos XML versionados; é proposta uma representação de documentos que mantém relacionamentos entre nós de documentos. Por fim, [31] apresenta um mecanismo para gerenciamento temporal de textos em XML através do uso de dimensões temporais e metadados para possibilitar consultas com base em restrições temporais.

A maioria das propostas propõem extensões ao modelo XML para a representação do aspecto evolutivo e algumas estensões à linguagens de consulta existentes [21] ou funções definidas pelo usuário [22], [33] para possibilitar recuperação temporal. Além disso, as propostas apresentadas focam na representação e gerenciamento de versões já conhecidas; o processo de detecção de versões não é tratado. A característica distribuída inerente aos ambientes P2P torna o problema de gerenciamento e consulta à dados versionados mais complexo. Sistemas P2P geralmente utilizam técnicas tradicionais de busca em largura e profundidade, mas não tratam o problema da existência de recursos versionados.

O ambiente em desenvolvimento, *DetVX*, foca nesta lacuna e propõe um ambiente para detecção, gerenciamento e consulta à versões de documentos XML em um contexto distribuído. O ambiente proposto não faz qualquer extensão ao modelo de dados XML para a representação do aspecto temporal dos documentos. Consultas temporais podem ser expressas atráves do linguagem *XQuery* padrão, sem o uso de extensões e sem o uso de funções temporais. Desta forma, operadores temporais básicos, como *after*, *before e now*, podem ser utilizados no acesso aos documentos versionados através de filtros nos rótulos temporais, o que torna o uso do sistema e da linguagem *XQuery* bastante intuitivos na submissão de consultas.

3. Ambiente DetVX

No ambiente *DetVX* (**Det**ecção de Réplicas e **V**ersões de Documentos **X**ML), cada *peer* atua como cliente e servidor. Documentos compartilhados estão armazenados em *peers*, segundo uma arquitetura *super peer*. Neste tipo de arquitetura, o usuário submete uma consulta a um *peer* específico. Se a consulta não puder ser respondida localmente, ela é roteada para o *super peer* do *peer* solicitante, o qual verifica seus *peers* disponíveis que estão aptos a responder a consulta e reenvia a consulta a estes *peers*. Os *peers* processam a consulta e retornam os resultados ao *super peer*, o qual os envia ao usuário final.

O critério de agrupamento de *peers* em *super peers* baseia-se no domínio de conhecimento ao qual os documentos do *peer* pertencem. Para a representação dos conceitos e relações, utiliza-se uma ontologia. Assume-se que cada *super peer* possui pelo menos uma ontologia associada a ele. Desta forma, um *super peer* pode agregar *peers* que contenham documentos pertencentes a vários domínios. Assume-se que existe uma entidade central responsável pelo gerenciamento do ambiente (comum em sistemas P2P que adotam a arquitetura parcialmente centralizada), conforme mostrado na Figura 1.

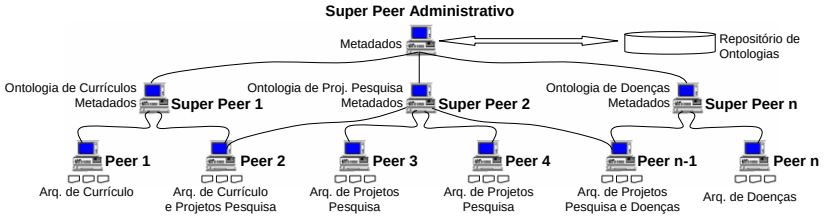

Fig. 1. Ambiente *DetVX*

Cada *super peer* possui metadados com informações sobre seus *peers* agregados e seus respectivos documentos registrados, tais como *data de registro* e *última data de modificação* de cada arquivo no *peer*. O *super peer* administrativo funciona como responsável pelo gerenciamento da rede. Os metadados do *super peer* administrativo descrevem os *super peers* existentes e seus respectivos domínios de aplicação, além de outras informações necessárias ao gerenciamento do ambiente.

Versões e réplicas de documentos podem estar localizadas em um único *peer* ou em *peers* diferentes. Se o usuário submete uma consulta a um *peer* específico solicitando o endereço atual de uma pessoa, o sistema deve retornar somente a última versão desta informação. Supondo que os dados existentes no *peer* onde a consulta foi submetida estejam desatualizados, então a consulta deve ser roteada para outros *peers*. Para descobrir qual(is) documento(s) é(são) necessário(s) acessar para responder uma determinada solicitação, o sistema consulta os metadados disponíveis no *super peer*.

A fim de fornecer as funcionalidades para o ambiente de detecção e gerenciamento de versões, este trabalho propõe a arquitetura mostrada na Figura 2. De maneira geral, o usuário interage com o sistema via uma interface, a qual possibilita registrar *peers* e documentos no ambiente (através do gerenciador de *peers*) e submeter consultas posteriores aos recursos compartilhados (através do processador de consultas).

Os módulos propostos na arquitetura possuem as seguintes funcionalidades:

- gerenciador de *peers*: responsável por conectar e reconectar *peers* ao sistema, além de verificar periodicamente a ocorrência de modificações nos arquivos compartilhados;
- gerenciador de ontologias: responsável pela manutenção do repositório de ontologias e pela associação de ontologias a *super peers*;

- gerenciador de réplicas e versões: responsável por identificar e representar réplicas e versões de documentos;
- processador de consultas: responsável por verificar à qual domínio uma consulta pertence e roteá-la ao *peer* adequado, com base nos metadados disponíveis.

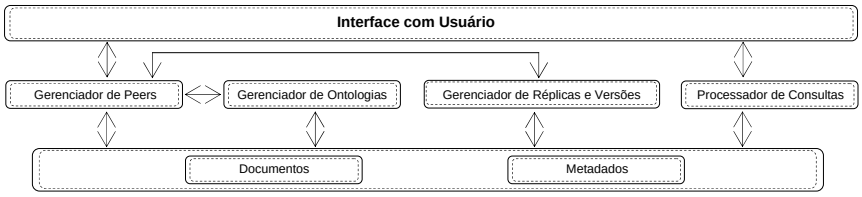

Fig. 2. Arquitetura do ambiente *DetVX*

Os módulos fazem uso intensivo de metadados para representação da informação. Metadados são representados como documentos XML e são classificados em dois níveis: metadados do *super peer* administrativo e metadados do *super peer*. Os metadados basicamente descrevem: *super peers* existentes e respectivos *peers* agregados; identificadores locais, tempos de registro e última data de atualização de cada arquivo no *peer; peers* agregados e seus respectivos documentos registrados; versões e réplicas disponíveis em um determinado *super peer;* e rótulos temporais correspondentes a cada elemento (*TS,TE*) encontrado em um dado documento de um *peer*. Rótulos temporais são derivados a partir das datas de modificação dos arquivos, e visam representar a evolução temporal de cada elemento, juntamente com seus respectivos tempos de validade. Este artigo foca nos módulos de gerenciamento de versões e processamento de consultas, descritos em mais detalhes nas seções a seguir.

4. Gerenciamento de Versões de Documentos

O trabalho proposto assume que a evolução dos documentos cria uma seqüência linear de versões temporalmente ordenadas: V_1, V_2, ..., V_n, onde a última versão V_n é a atual. Uma versão pode possuir apenas uma versão antecessora e outra sucessora. A fim de ordenar as versões de arquivos em uma linha de tempo, leva-se em consideração os tempos de modificação destes arquivos. Assume-se que o arquivo com tempo de modificação mais recente é uma derivação de uma versão anterior. Supondo duas versões de arquivos, *F1* e *F2*, com as respectivas datas de modificação *10/10/2005* e *03/04/2006*, conclui-se que o arquivo *F2* é uma versão modificada do arquivo *F1*.

A detecção de versões baseia-se na similaridade entre documentos. Em uma abordagem simples, se dois arquivos possuem um conjunto pequeno de diferenças significativas, então eles são considerados duas versões do mesmo documento. Se eles possuem um conjunto grande de diferenças significativas, então são considerados dois documentos diferentes. Para definir o ponto de corte do conjunto de diferenças, propõe-se utilizar um limiar de similaridade. Arquivos que apresentam similaridade acima deste limiar serão considerados versões de um mesmo documento. Arquivos que apresentem similaridade abaixo deste limiar serão considerados dois documentos distintos. No entanto, a detecção de versões não é o foco deste artigo e não será apresentada.

A maioria dos sistemas de gerenciamento utilizados no suporte à configuração de *software* consideram arquivos como objetos básicos de controle de versão. Estes sistemas

armazenam diferenças entre duas versões de um arquivo, utilizando deltas (processados através de algoritmos *diff*), e utilizam os deltas para reconstruir versões, anteriores ou posteriores, dos arquivos [12]. A proposta deste trabalho não armazena versões prévias (ou posteriores) como mudanças delta sobre o estado corrente de um certo documento. Ao invés disso, versões são armazenadas na rede como arquivos físicos separados.

Para identificar as mudanças ocorridas entre as versões, processa-se as diferenças com um algoritmo *diff* que constrói o delta representativo destas mudanças. A entrada de um algoritmo *diff* consiste de dois documentos e sua saída é um documento delta que representa as diferenças entre os dois documentos de entrada. Vários algoritmos podem ser encontrados na literatura [5],[18]. Alguns são mais eficientes em termos de tempo de execução, mas nem sempre garantem a solução ótima nas diferenças detectadas [6]. Até o presente momento, o algoritmo *XyDiff* [5] tem sido utilizado nos testes realizados. Entretanto, a arquitetura proposta permite a troca por outro algoritmo *diff*.

Considere dois arquivos XML contendo informações relacionadas a currículos, mostrados na Figura 3 (a) e Figura 3 (b). Nota-se que algumas modificações foram realizadas no segundo arquivo: o elemento *phone* foi atualizado, o elemento *fax* foi removido e o elemento *maritalSt* foi inserido. Utilizando-se o algoritmo *XyDiff*, obtém-se as seguintes operações, mostradas na Figura 3 (c). As operações 2 e 3 correspondem à atualização do elemento *phone*. A operação 1 corresponde à remoção do elemento *fax*. A operação 4 corresponde à inserção do elemento *maritalSt*. Estas operações são estruturadas em um documento XML, mostrado na Figura 3 (d).

a) Arquivo Original

```
<curriculum>
   <personalData>
        <name>Ana Silveira</name>
        <phone>5665458</phone>
        <address>Rua A</address>
        <fax>2888888</fax>
   </personalData>
</curriculum>
```

b) Arquivo Modificado

```
<curriculum>
    <personalData>
        <name>Ana Silveira</name>
        <phone>333000</phone>
        <address>Rua A</address>
        <maritalSt>single</maritalSt>
   </personalData>
<curriculum>
```

c) Diferenças detectadas com XyDiff

```
1.DeleteNode:
  <fax>2888888</fax> path
  0:0:0:3
2.DeleteNode: 5665458 path
  0:0:0:1:0 (update)
3.InsertNode: 333000 path
  0:0:0:1:0 (update)
4.InsertNode: <mari-
  talSt>single</maritalSt> path
  0:0:0:3
```

d) Arquivo XML com as diferenças detectadas

```
<delta>
   <Deleted pos="0:0:0:3"> <fax>2888888</fax>
   </Deleted>
   <Deleted update="yes"
      pos="0:0:0:1:0">5665458</Deleted>
   <Inserted update="yes"
      pos="0:0:0:1:0">333000</Inserted>
   <Inserted
   pos="0:0:0:3"><maritalSt>single</maritalSt>
   </Inserted>
</delta>
```

Fig. 3. Documento XML original (a) e modificado (b), com alterações detectadas (c) (d)

Por questões de desempenho, este artigo propõe gerar uma nova representação contendo todas as modificações do documento, e armazenar esta versão consolidada no *super peer*. Desta forma, consultas que solicitam o histórico de um documento podem ser respondidas, acessando-se um único arquivo localizado no *super peer*, eliminando a necessidade de um processamento distribuído por toda a rede. Nem todos os documentos possuem uma versão consolidada armazenada no *super peer*, mas principalmente aqueles documentos que são mais frequentemente acessados pelo sistema. O arquivo físico que contém todo o histórico de um documento XML é referenciado neste artigo como arquivo *H-Doc*. O mecanismo de agrupamento de versões de documentos em um único arquivo físico é descrito na seção a seguir.

4.1 Agrupamento de Documentos nos *Super Peers*

A partir de *n* arquivos físicos contendo a evolução do documento, gera-se uma representação única, contendo todas as modificações. Esta nova representação pode ser armazenada como um novo arquivo (*H-Doc*). Desta forma, para cada documento com *n* versões, onde *n>1*, uma nova representação é criada e armazenada no *super peer*. Rótulos temporais são responsáveis por validar e invalidar dados em versões específicas, baseados nos tempos iniciais (*TS*) e tempos finais (*TE*). *TS* e *TE* representam o período de tempo no qual uma informação é válida no mundo real.

Assume-se que cada elemento possui dois atributos temporais, *TS* e *TE*. *TS* é inicializado com o tempo de modificação do arquivo no qual o elemento foi adicionado/modificado. *TE* é inicializado com *now* e permanece com este valor enquanto for válido; um elemento é considerado válido até sua próxima modificação/remoção. *TE* é atualizado quando um elemento é modificado ou removido; neste caso, *TE* é encerrado quando uma nova instância de elemento é criada. A Figura 4 mostra a representação em árvore gerada para armazenar o histórico dos documentos apresentados na Figura 3.

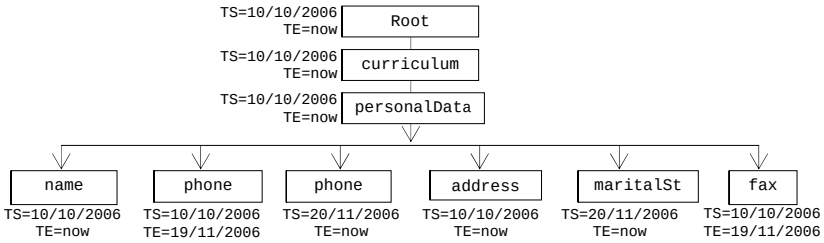

Fig. 4. Representação em árvore

A tarefa de gerar uma versão consolidada contendo todo o histórico do documento é similar ao processo de *merging* de arquivos, bastante utilizado em sistemas de versões concorrentes (CVS). No entanto, na proposta deste trabalho, não existem conflitos a serem resolvidos, uma vez que todo o histórico do documento é armazenado, com os respectivos rótulos temporais.

A representação consolidada do documento mostrado na Figura 4 pode ser armazenada fisicamente em um novo arquivo XML, mostrado na Figura 5.

```
<?xml version="1.0" encoding="UTF-8"?>
<ROOT TS="10/10/2006" TE="now">
 <curriculum TS="10/10/2006" TE="now">
  <personalData TS="10/10/2006" TE="now"><name TS="10/10/2006" TE="now">…</name>
  <phone TS="10/10/2006" TE="19/11/2006">…</phone><phone TS="20/11/2006" TE="now">…</phone>
  <address TS="10/10/2006" TE="now">…</address><marital TS="20/11/2006" TE="now">…</marital>
  <fax TS="10/10/2006" TE="19/11/2006">…</fax>
  </personalData>
 </curriculum>
</ROOT>
```

Fig. 5. Versão consolidada – arquivo *H-Doc*

Representações consolidadas são armazenadas no *super peer* e podem ser utilizadas para um processamento mais rápido de certas consultas que solicitam histórico de documentos. Dessa maneira, não há a necessidade de acessar todas as versões de um documento espalhadas pelos *peers.* Obviamente, os metadados do ambiente *DetVX* devem prever a representação destas versões consolidadas, de forma que o processador de consultas possa tirar vantagem dos arquivos *H-Doc*, quando estes existirem. O detalhamento dos metadados não é descrito neste artigo.

4.2 Consulta à Versões de Documentos

Sistemas que fornecem suporte a documentos versionados devem disponibilizar mecanismos para recuperação destas informações. No ambiente *DetVX*, antes de rotear uma consulta a um determinado *peer*, o sistema verifica se a consulta pode ser respondida localmente. Em caso afirmativo, a consulta é processada e os resultados são retornados ao usuário; caso contrário, a consulta é roteada para o *peer* adequado. Para verificar se uma consulta pode ser respondida localmente (pelo *peer* solicitante), o sistema verifica os metadados de seu *super peer*. Por exemplo, considere uma consulta que solicita a primeira versão de um arquivo *F*; através dos metadados, verifica-se em qual *peer* esta informação está armazenada (analisando-se o número da versão solicitada).

Com o agrupamento de versões em um único arquivo físico (*H-Doc*), determinadas consultas podem ser diretamente respondidas pelo *super peer*, como: obter o endereço do autor antes de *10/10/2006*; obter a última versão do endereço dos autores; obter o histórico de endereço dos autores, etc. Caso o *peer* solicitante não tenha condições de responder a consulta, a solicitação é enviada ao *super peer*, o qual se responsabiliza por respondê-la (caso possua o histórico daquele documento armazenado localmente) ou roteá-la para o *peer* que possui a informação solicitada. Desta forma, evita-se que a rede fique sobrecarregada com requisições que somente determinado *peer* (ou *super peer*) está apto a responder. Nesta seção será considerado apenas o processo de consulta em arquivos *H-Doc*. Questões prévias de análise de metadados e roteamento de consultas não são abordadas.

Considerando um elemento qualquer *e* de um documento *D*, os filtros temporais aplicados podem ser baseados em uma data *x* ou em um intervalo de tempo *x* e *y*. Algumas cláusulas utilizadas para a execução de consultas temporais são mostradas a seguir:

- *select_Before (e, x)*: esta cláusula retorna o conteúdo dos elementos *e* que são válidos no documento *H-Doc* antes da data *x*. Em outras palavras, procura-se por elementos cujo tempo inicial de validade (*TS*) seja menor à data *x* especificada;
- *select_After (e, x)*: esta cláusula retorna o conteúdo dos elementos *e* que são válidos no documento *H-Doc* após a data *x*. Em outras palavras, procura-se por elementos cujo tempo final de validade (*TE*) seja maior à data *x* especificada;
- *select_Between (e, x, y)*: esta cláusula retorna o conteúdo dos elementos *e* que são válidos no documento *H-Doc* entre o perído *x* e *y*, onde $x<y$. Em outras palavras, procura-se por elementos cujo tempo inicial de validade (*TS*) seja menor ou igual a *y* e cujo tempo final de validade (*TE*) seja maior ou igual a *x*;
- *select_Now (e)*: esta cláusula retorna o conteúdo dos elementos *e* que são válidos no documento *H-Doc* no momento atual. Em outras palavras, procura-se por elementos cujo tempo final de validade (*TE*) seja igual a *now*;
- *select_Before (D, x)*: esta cláusula retorna o conteúdo do documento *D* que é válido antes da data *x*. Em outras palavras, procura-se por documentos cujo tempo inicial de validade (*TS*) do elemento raiz seja menor à data *x* especificada;
- *select_After (D, x)*: esta cláusula retorna o conteúdo do documento *D* que é válido após a data *x*. Em outras palavras, procura-se por documentos cujo tempo final de validade (*TE*) do elemento raiz seja maior à data *x* especificada;
- *select_Between (D, x, y)*: esta cláusula retorna o conteúdo do documento *D* que é válido entre o perído *x* e *y*. Em outras palavras, procura-se por documentos cujo tempo inicial de validade (*TS*) do elemento raiz seja menor ou igual a *y* e cujo tempo final de validade (*TE*) do elemento raiz seja maior ou igual a *x*;

- *select_Now (E)*: esta cláusula retorna o conteúdo do documento *D* que é válido no no momento atual. Em outras palavras, procura-se por documentos cujo tempo final de validade (*TE*) do elemento raiz seja igual a *now*.
  Alguns exemplos de consultas temporais implementadas são mostrados na seção 5.2.

5. Implementação

A implementação da ferramenta *XVersion* disponibiliza um ambiente que agrupa em um único arquivo de saída os *n* arquivos XML de entrada, através do processamento de diferenças. O agrupamento destes arquivos baseia-se no uso de algoritmos *diff* e rótulos temporais que especificam a validade de cada elemento dentro do documento de saída gerado. Com a geração da representação única do histórico dos documentos, o usuário pode submeter consultas temporais, utilizando *XQuery* [17].

5.1 Agrupamento dos Documentos

Dentre as funcionalidades da *XVersion*, a primeira a ser utilizada pelo usuário é o comparador de arquivos XML. Para esta comparação, foi utilizada a implementação em Java do algoritmo *XyDiff*. A partir de *n* documentos XML fornecidos como entrada, executa-se o *XyDiff* para cada par de arquivos a_i e a_{i+1} (onde *i* é o índice do documento XML dentro da lista de documentos fornecidos), obtendo assim *n-1* documentos representando as diferenças entre cada par de arquivos fornecido. A Figura 6 mostra os arquivos delta gerado para três versões de documentos XML fornecidas, utilizando *XyDiff*. Considere que as três versões possuem as seguintes datas de modificação, respectivamente: *09/09/2006*, *05/11/2006* e *11/11/2006*.

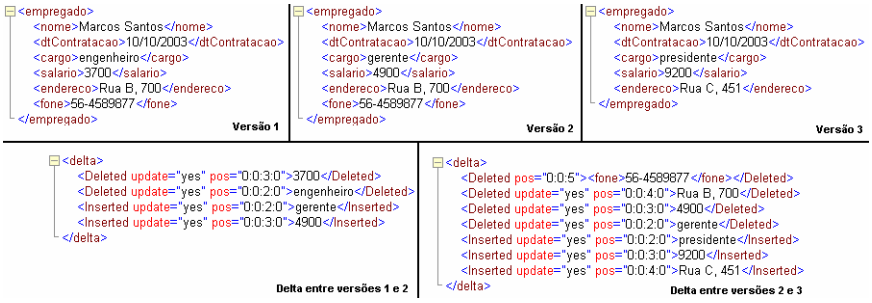

Fig. 6. Arquivos delta gerados para três versões de documentos XML

A seguir, é criada a representação do arquivo consolidado para a representação do histórico das versões. Para isto, é criado um arquivo XML representando a união de cada par de versões fornecidas. Esta união é feita a partir dos dois arquivos fornecidos e do arquivo delta contendo as diferenças entre estes. O último passo é juntar todas as uniões geradas em um único arquivo. A Figura 7 mostra o documento *H-Doc* resultante, contendo o histórico das versões dos arquivos mostrados na Figura 6.

Embora a geração do documento *H-Doc* seja simples de se entender, o desenvolvimento do algoritmo que trata da união de mais de dois arquivos foi complexo. Um nodo pode

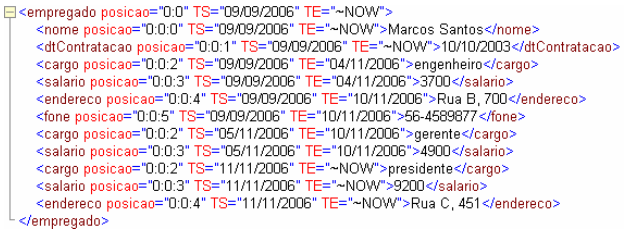

Fig. 7. Históricos das versões - documento *H-Doc* resultante

sofrer várias alterações ao longo das versões, o que implica em aparecer várias vezes no documento consolidado gerado. Neste caso, torna-se necessário identificar qual das versões deste nodo é a válida. Adotou-se neste trabalho uma abordagem baseada na equivalência de nodos por posição, na qual metadados são inseridos nos arquivos XML antes de executar qualquer operação sobre eles. Estes metadados indicam a posição do elemento dentro do arquivo XML (atributo *posicao*) e a partir deles é possível descobrir a equivalência de nodos entre versões. A estrutura de geração dos valores do atributo *posicao* segue a identificação de nodos utilizada pelo algoritmo *XyDiff*.

5.2 Consulta ao Histórico de Documentos

A última funcionalidade implementada na *XVersion* é a possibilidade de se executar consultas sobre o arquivo *H-Doc*. Desta forma, uma única consulta é expressa sobre um único arquivo físico. Através de filtros temporais, informações históricas podem ser retornadas sem a necessidade de processar a consulta em toda a rede. O trabalho apresentado em [16] mostra uma proposta similar, e usa funções definidas pelo usuário para expressar consultas temporais em *XQuery* à versões de documentos XML.

Exemplos de consultas temporais incluem: obter informações válidas *antes* de uma certa data/período de tempo, obter informações válidas *após* uma certa data/período de tempo, obter informações válidas *entre* um certo período de tempo, obter informações válidas no momento atual, obter todo o histórico de uma certa informação, entre outras. Algumas consultas executadas na ferramenta são mostradas na Figura 8.

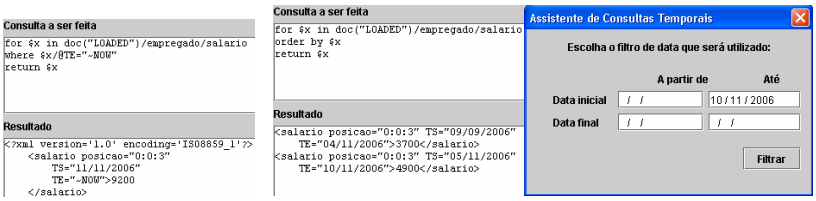

Fig. 8. Exemplos de consultas temporais

A primeira consulta retorna o conteúdo do elemento *salário* no momento atual. Para responder a esta consulta, a ferramenta busca todos os elementos cujo tempo de validade final seja *now*. A segunda consulta retorna os valores de salário válidos antes do dia *10/11/2006*. Para responder a esta consulta, a ferramenta busca todos os elementos *salário* cujo tempo inicial de validade seja inferior à data especificada. Para a execução das consultas foi utilizada a biblioteca *Qizx/Open*[5], distribuída sob a licença GNU.

[5] http://www.xfra.net/qizxopen/

6. Conclusões

Este artigo apresentou um mecanismo de agrupamento e consulta à versões de documentos XML, inserido na arquitetura do *DetVX*, um ambiente para detecção de réplicas e versões XML em um contexto P2P. Gerenciamento de versões tem sido bastante estudado, mas a característica distribuída dos ambientes P2P torna o problema mais complexo. Este artigo apresentou uma abordagem simples de representação de versões em um único arquivo físico, gerado pelo processamento de diferenças entre versões. Consultas podem ser expressas através da restrição de valores nos rótulos temporais. Desta forma, elimina-se a necessidade de processar consultas distribuídas na rede P2P, uma vez que a representação consolidada com o histórico das versões é armazenada no *super peer*. Com a abordagem proposta, espera-se melhorar o desempenho das consultas em documentos frequentemente acessados. No entanto, este artigo não apresentou resultados comparativos de desempenho, os quais serão realizados futuramente, à medida que o ambiente no qual a ferramenta *XVersion* está inserido seja finalizado.

Atualmente o foco do ambiente em desenvolvimento é trabalhar no problema de *ranking* de páginas Web em máquinas de busca. Um dos critérios utilizados em *ranking* de páginas é o número de *links* que apontam para uma determinada página *p*. No entanto, uma nova versão de uma página *p* pode ter um pequeno número de apontadores, principalmente porque as páginas que apontam para *p* ainda não estão cientes da nova versão disponível. Neste contexto, detecção de réplicas e versões pode ser útil para melhorar o *ranking* de novas versões de páginas mesmo que o número de apontadores ainda seja pequeno.

O ambiente proposto neste artigo baseia-se na arquitetura *super peer*, mas faz uso de uma entidade central para o gerenciamento de alguns metadados. No entanto, o ambiente pode ser utilizado em redes distribuídas (utilizando DHT – *distributed hash tables*), com algumas adaptações. Como trabalho futuro, pretende-se ampliar o conjunto de funções temporais fornecido pela *XVersion* e incorporar a ferramenta ao ambiente *DetVX* em desenvolvimento.

Referências

1. Aberer, K. and Hauswirth, M.. An Overview on Peer-to-Peer Information Systems. Workshop on Distributed Data and Structures (WDAS), Paris, France, 2002.
2. Chawathe, S. e Widom, J.. Representing and Querying Changes in Semistructured Data. Proc. of 14th Intl. Conference on Data Engineering, 4-13, 1998.
3. Chien, S.; Tsotras, V.J. e Zaniolo, C.. Version Management of XML Documents. Lecture Notes in Computer Science, 2001.
4. Chien, S.Y.; Tsotras, V.J. e Zaniolo, C.. XML Document Versioning. SIGMOD Rec., ACM Press, 30, 46-53, 2001.
5. Cobena, G., Abiteboul, S. e Marian, A.. Detecting Changes in XML Documents. , Proc. of 18th Intl. Conference on Data Engineering, 41-52, 2002.
6. Cobena, G.; Abdessalem, T. e Hinnach, Y.. A Comparative Study for XML Change Detection. Verso report number 221, INRIA, 2002.
7. Conradi, R. e Westfechtel, B.. Version Models for Software Configuration Management. ACM Comput. Surv., 30(2):232–282, 1998.
8. Grandi, F., Mandreolo, F., Tiberio, P. E Bergonzini. P.. A Temporal Data Model and Management System for Normative Texts in XML Format. Proc. of the 5th ACM Intl. Workshop on Web Information and Data Management, (WIDM), ACM Press, 29-36, 2003.

9. Jensen, C. S., Dyreson, C. E. et al.. The Consensus Glossary of Temporal Database Concepts. In: Temporal Databases - Research and Practice. Springer-Verlag, 1998.
10. Katz, R. e Chang, E.. Managing Change in a Computer-Aided Design Database. Proc. of VLDB Conference, 1987.
11. Ozsoyoglu, G. e Snodgrass, R.. Temporal and Real-Time Databases: A Survey. IEEE Transactions on Knowledge and Data Engineering, 7, 513-532, 1995.
12. Ronnau, S.; Scheffczyk, J. e Borghoff, U.M.. Towards XML Version Control of Office Documents. DocEng '05: Proc. of the 2005 ACM symposium on Document engineering, ACM Press, 10-19, 2005
13. Tansel, A., Clifford, J., Gadia, S., Jajodia, S., Segev, A. e Snodgrass, R. T.. Temporal Databases: Theory, Design and Implementation. Redwood City: Benjamin/Cummings Publishing Company, Inc., 1993
14. Vagena, Z.; Moro, M. e Tsotras, V.. Supporting Branched Versions on XML Documents. Proc. of 14th Intl. Workshop on Research Issues on Data Engineering: Web Services for e-Commerce and e-Government Applications, 137-144, 2004.
15. Westfechtel, B., Munch, B. P., e Conradi, R. A Layered Architecture for Uniform Version Management. IEEE Trans. Software Eng., 27(12):1111–1133, 2001.
16. Wang, F.; Zaniolo, C.; Zhou, X.; Moon, H, J.. Version Management and Historical Queries in Digital Libraries. Proc. of the 12th Intl. Symposium on Temporal Representation and Reasoning, 207-209, 2005.
17. XQuery. XQuery 1.0: An XML Query Language. W3C Candidate Recommendation 8 June 2006. Disponível em http://www.w3.org/TR/xquery/
18. Wang, Y., DeWitt, D. J., Cai, J.. X-Diff: An Effective Change Detection Algorithm for XML Documents. Intl. Conference on Data Engineering, 519-530, 2003.
19. Schollmeier, R.. A Definition of Peer-to-Peer Networking for the Classification of Peer-to-Peer Architetures and Applications. Proc. of the 1st Intl. Conf. on Peer-to-Peer Computing, 27-29, Linköping, Sweden. IEEE Computer Society 2001, ISBN 0-7695-1503-7.
20. Su, H., Kramer, D., Chen, L., Claypool, K. T., Rundensteinrer, E. A.. XEM: Managing the Evolution of XML Documents. Proc. of 11th Intl. Workshop on Research Issues in Data Engineering, Heidelberg, 2001.
21. Gao, D. and Snodgrass, R.T.. Temporal Slicing in the Evaluation of XML Queries. Proc. of Very Large Database Systems, 2004.
22. Wang, F. and Zaniolo, C.. Representing and Querying the Evolution of Databases and their Schemas in XML. In Workshop on Web Engineering, SEKE, San Francisco, USA, 2003.
23. Chawathe, S.S.; Abiteboul, S. and Widom, J.. Managing Historical Semistructured Data. Theory and practice of object systems, 2000.
24. Grandi, F. e Mandreoli, F.. The Valid Web: an XML/XSL Infrastructure for Temporal Management of Web Documents. Proc. of Advances in Information Systems, (ADVIS 2000), 2000.
25. Gergatsoulis, M. e Stavrakas, Y.. Representing Changes in XML Documents using Dimensions. First XML Database Symposium, (XSym), Berlin, Germany, 208-222, 2003.
26. Amagasa, T.; Yoshikawa, M. e Uemura, S.. A Data Model for Temporal XML Documents. Proceedings of the 11th Intl. Conf. on Database and Expert Systems Applications, (DEXA), London, England , 2000.
27. Amagasa, T.; Yoshikawat, M. e Uemura, S.. Realizing Temporal XML Repositories using Temporal Relational Databases. Proc. of the 3rd Intl. Symposium on Cooperative Database Systems for Advanced Applications, (CODAS), 60-64, 2001.
28. Dyreson, C.. Observing Transaction-Time Semantics with /sub TT/XPath. Proc. of the 2nd Intl. Conf. on Web Information Systems Engineering , 1, 193-202, vol.1, 2001.
29. Chien, S.; Tsotras, V.; Zaniolo, C. and Zhang, D.. Storing and Querying Multiversion XML Documents using Durable Node Numbers. Proc. of the 2nd Intl. Conf. on Web Information Systems Engineering, 1, 232-241, vol.1, 2001.
30. Vagena, Z. e Tsotras, V.. Path-Expression Queries over Multiversion XML Documents. Proc. of Intl. Workshop on the Web and Databases (WebDB), 49-54, 2003.
31. Grandi, F., Mandreoli, F., Tiberio, P.. Temporal Modeling and Management of Normative Documents in XML Format. Data & Knowledge Engineering, v. 54, n. 3, p. 327-354, September, 2005.
32. Wang, F. e Zaniolo, C.. Temporal Queries in XML Document Archives and Web Warehouses. Proc. of the 10th Intl. Symposium on Temporal Representation and Reasoning and Fourth International Conference on Temporal Logic, 47-55, 2003.
33. Wang, F.; Zaniolo, C.; Zhou, X. e Moon, H.J.. Managing Multiversion Documents and Historical Databases: a Unified Solution Based on XML. Proc. of the 8th Intl. Workshop on the Web, (WebDB), Baltimore, Maryland, USA, 151-153, 2005.