Suporte a transações longas em atualizações através de visões XML

André Prisco Vargas[1]
Orientadores: Carlos Alberto Heuser[1] **e Vanessa P. Braganholo (COPPE/UFRJ)**

[1]{apvargas, heuser}@inf.ufrgs.br, vanessa@cos.ufrj.br
Nível: Mestrado
Programa de Pós Graduação em Computação – Instituto de Informática
Universidade Federal do Rio Grande do Sul
Ingresso: Março de 2004 – Previsão de Conclusão: Março de 2006

Resumo. *Visões XML de bancos de dados relacionais têm sido largamente utilizadas por aplicações Web. Neste artigo, tratamos o cenário onde visões XML são extraídas de uma base relacional e entregues a uma aplicação via Web, que a edita e a envia de volta. Neste contexto, o uso de bloqueios para controle de concorrência é inviável, já que o usuário pode ficar de posse da visão por um longo período, antes de retorná-la. Para isso devem ser resolvidos alguns problemas, como a identificação de quais alterações foram feitas pelo usuário e se o estado da base de dados foi alterado durante a transação. Serão abordados estes dois problemas, usando como base o sistema de atualização de visões XML PATAXÓ.*

1. Introdução

O uso de XML para intercâmbio de dados tem se tornado um padrão em aplicações empresariais, principalmente na Web. Porém, em sua maioria, o armazenamento dos dados ainda é feito em banco de dados relacionais. Dessa forma muitas aplicações que trabalham com dados em XML precisam que as informações armazenadas nas bases de dados sejam recuperadas através de visões no formato XML. Entre os trabalhos com o objetivo de gerar visões XML a partir de bases de dados relacionais estão o XPeranto [5] e o Silk-Route [4]. No entanto estes trabalhos resolvem apenas parte do problema, já que em muitos casos somente a geração das visões XML não é suficiente. Nas aplicações empresariais também se deseja poder atualizar as visões XML, e que essas alterações sejam refletidas na base de dados relacional.

Um exemplo para tal arquitetura seria uma empresa fornecedora F e uma empresa cliente C, que compra os produtos de F. A empresa F fornece um documento XML para ser preenchido com os produtos as serem adquiridos (suponha que a empresa C já conhece o catálogo dos produtos de F). O documento pode ser enviado a um dispositivo móvel, como um *Palm*. Um funcionário da empresa C pode então selecionar os produtos a serem adquiridos, editando o próprio documento XML. O documento atualizado é então enviado à empresa F, que atualiza sua base de dados relacional, registrando os novos pedidos de produtos. Podem haver outros casos mais complexos. Suponha o caso da empresa C desejar alterar um pedido já enviado. A empresa envia uma visão XML com os pedidos feitos anteriormente e a empresa cliente altera o pedido editando tal visão. Este exemplo será usado como base neste artigo.

```
Produto (prodId,descrição,preço)                //Excluindo as canetas
  primary key (prodId)                          DELETE FROM Produto
Pedido (numPedido, data)                        WHERE prodId = 1
  primary key (numPedido)
LinhaPedido (numPedido,prodId,quantidade)       //Aumentando o preço das borrachas
  primary key (numPedido,prodId)                UPDATE Produto
  foreign key (numPedido) references Pedido     SET preço = 0.10
  foreign key (prodId) references Produto       WHERE prodId = 2
```

Figura 1. (a) Base relacional da empresa F. (b) Modificações

O sistema PATAXÓ [1] é uma abordagem para atualização de bases de dados relacionais através de visões XML. Nesta abordagem, o usuário gera visões através de consultas UXQuery. O sistema então gera a visão XML a partir da base de dados relacional e retorna a visão XML resultante ao usuário. O usuário pode fazer atualizações sobre a visão através de uma linguagem de atualização específica do sistema [1]. As requisições de atualização são enviados e mapeadas para atualizações na base relacional. Porém tal proposta não é totalmente aplicável ao cenário descrito acima, tendo dois problemas em aberto: (i) No cenário acima descrito, as atualizações são feitas diretamente no documento XML e não através de alguma linguagem. As atualizações devem ser descobertas através da diferença (*delta*) entre a visão enviada e a retornada ao usuário; (ii) Na proposta original do PATAXÓ, o gerenciamento de transações fica por conta do próprio SGBD, o que na prática significa fazer um bloqueio nas tuplas envolvidas em uma visão (modelo de transações ACID). Porém, nesse contexto, o uso de bloqueios não é viável, já que o usuário pode ficar de posse das visões por um tempo muito grande. Durante o tempo em que o usuário fica de posse da visão, a base de dados pode ser alterada, o que pode gerar conflitos com as visões de posso de algum cliente.

O objetivo deste artigo é abordar estes dois problemas em aberto, criando um mecanismo de controle de transação apropriado para este contexto, implementado como um módulo do sistema PATAXÓ.

A figura 1(a) é um exemplo de base de dados relacional, que poderia ser usado pela empresa F. Suponha que a empresa C enviou alguns pedidos de compra para F, e agora deseja alterar um deles. A visão XML gerada (figura 2) é então enviada à empresa C. Enquanto a empresa C está de posse da visão, um outro usuário faz as modificações (não necessariamente através de visões XML) apresentadas na figura 1 (b), excluindo as canetas e aumentando o preço das borrachas. Após um período o usuário da empresa C retorna a visão alterada, aumentando a quantidade dos pedidos de borracha e caneta e inserindo o pedido de cadernos, conforme a figura 3.

Neste contexto, quando o sistema recebe a visão XML alterada, deve tratar os seguintes problemas: (i) Detectar as diferenças entre a visão original e a visão alterada; (ii) Detectar que alterações na base de dados pode entrar em conflito com as alterações do usuário; (iii) Decidir como tratar com os conflitos e enviar as alterações ao PATAXÓ.

Cada um desses passos será descrito nas seções posteriores. O restante do texto está dividido como segue. A seção 2 apresenta trabalhos relacionados e a seção 3 descreve a abordagem proposta para a solução. Finalmente, a seção 4 apresenta a conclusão.

2. Detecção de *Deltas*

Um *delta* consiste em um conjunto de operações que expressa a diferença entre dois dados. A detecção de *delta* em dados estruturados possui alguns problemas importantes,

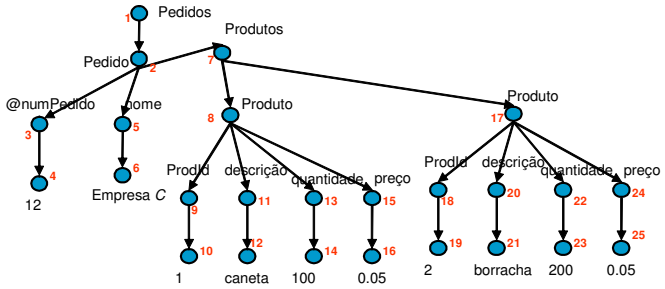

Figura 2. Visão XML original

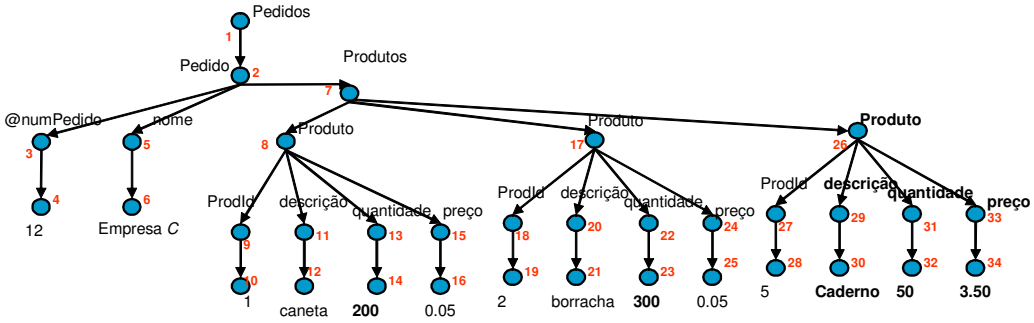

Figura 3. Visão XML alterada pela empresa C e retornada a empresa F

como a correspondência entre os nodos e a descoberta do *delta* que melhor expressa as alterações feitas de fato pelo usuário ou que é mais adequado para uma aplicação. O MHDIFF [2] é um dos primeiros trabalhos a tratar da detecção de diferenças em dados estruturados, sendo base para outras propostas. O Xydiff [3] é um algoritmo para detecção de mudanças especificamente projetado para lidar com documentos XML. Já o X-Diff [7] trata de documentos XML não-ordenados, sendo mais adequados ao caso de documentos XML gerados a partir de bases relacionais.

3. Suporte a Transações Longas

Nesta seção é descrita a abordagem proposta para o controle de concorrência para as visões atualizáveis. O termo ”transação longa”aplicado neste trabalho difere de seu uso mais comum na bibliografia. Neste trabalho as transações não são necessariamente formadas por muitas operações, fazendo com que fiquem executando durante um grande período de tempo. No cenário aplicado neste trabalho, a questão está no tempo em que o sistema demora para saber quais são as operações necessárias para executar as transações, que é o tempo em que o usuário está de posse da visão. O controle de concorrência é formado pelos módulos de Detecção de *Delta*, Gerenciamento de Transações e Gerenciamento de Atualizações, conforme a figura 4. Primeiramente o usuário envia ao Gerenciador de Transações a consulta de definição da visão XML (em UXQuery). O PATAXÓ recebe tal requisição e retorna a visão XML (como no exemplo da figura 2) e sua DTD para o usuário. A definição UXQuery, a DTD e a visão XML são armazenadas num repositório temporário.

O usuário, após ter ficado de posse da visão, retorna a visão alterada ao sistema, como no exemplo da figura 3. Esta é entregue ao Detector de *Deltas* que vai compará-la com a visão originalmente entregue ao usuário. O resultado é enviado ao Gerenciador

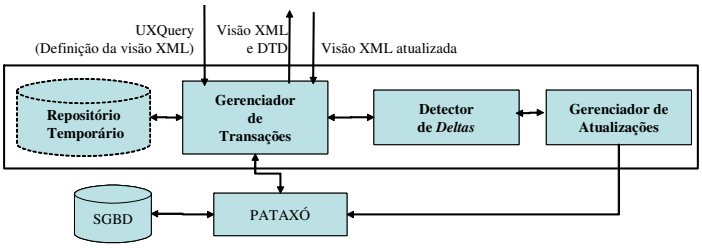

Figura 4. Arquitetura da solução proposta

de Transações, que detecta e analisa possíveis conflitos nas atualizações. As atualizações que não resultam em conflitos são enviadas ao PATAXÓ, que atualiza a base de dados.

O tratamento de conflitos ainda é uma questão em aberto neste trabalho. Uma das alternativas é enviar ao usuário o conflito gerado afim de que ele refaça sua transação. Este caso porém pode gerar uma situação de postergação indefinida, uma vez que enquanto o usuário refaz sua transação, novos conflitos são gerados por outras atualizações. Outra alternativa sendo estudada é executar as operações não conflitantes da transação. Porém tais alternativas e seus efeitos não foram completamente estudadas a fim de adotá-las.

3.1. Detector de *Deltas*

O Detector de *Deltas* tem como base o X-Diff [7], principalmente por considerar árvores não-ordenadas e por detectar operações suportadas pelo PATAXÓ (inserção e exclusão de subárvores, modificação de nodos texto). Este porém deve ser adaptado ao contexto de atualização das bases de dados. Quando se trata de documentos XML comuns, não há necessariamente restrições quanto às alterações no documento. No caso das visões XML, as atualizações não podem alterar o esquema da visão XML, já que isso não é permitido pelo PATAXÓ. Caso uma alteração torne a visão inválida em relação à DTD, a transação é cancelada e o usuário é avisado.

3.2. Gerenciador de Transações

Como foi visto anteriormente, a base de dados pode ser alterada enquanto uma visão XML está sendo alterada. No exemplo, o usuário aumentou a quantidade do pedido de borrachas, cujo preço aumentou, e de canetas, produto que não existe mais na base de dados. O primeiro passo é detectar esse tipo de conflito. Para isso, quando a visão é retornada, o sistema gera uma outra visão (seguindo a mesma definição UXQuery que gerou as outras), expressando o estado atual do banco. Estas três visões devem ser comparadas pelo Detector de *Deltas*. O Detector de *Delta* gera dois scripts com as diferenças entre elas e os envia para o Gerenciador de Atualizações. A figura 5 apresenta os *deltas* do exemplo. $Update(a, x)$ coloca o valor x no nodo a, $Insert(x, a)$ insere a subárvore x como filha do nodo a e $Delete(x)$ exclui a subárvore x. O Delta1 representa a diferença entre a visão original e a alterada pelo usuário. O Delta2 representa a diferença entre a visão entregue ao usuário e a que representa o estado atual da base de dados. Os números do exemplo correspondem aos números dos nodos que aparecem nas figuras 2 e 3.

3.3. Gerenciador de Atualizações

O Detector de *deltas* retorna as alterações num formato próprio, diferente do usado no PATAXÓ. Este módulo recebe o *delta* e mapeia tais modificações para operações da linguagem de atualização do PATAXÓ. Por restrição de espaço, não serão apresentadas as

```
Delta1(Original -> Alterada) = Update(14, 200), Update(23, 300), Insert(t1, 7))
t1 = <item>
        <prodId>2</prodId>
        <description>caderno</description>
        <quantity>50</quatity>
        <price>3.50</price>
     </item>
Delta2(Original -> Atual) = Delete(8), Update(24, 0.10)
```

Figura 5. *Deltas* usados no exemplo apresentado

funções de mapeamento nem a sintaxe da linguagem, mas estas podem ser vistas, respectivamente, em [6] e [1].

O Gerenciador de Atualizações, além de converter as atualizações para a sintaxe do PATAXÓ, também detecta se houve conflitos nas atualizações (se alguma porção da visão foi alterada no banco de dados durante a transação). Para isso, ela utiliza os deltas gerados e faz um *merge* da visão atualizada com a visão gerada a partir do estado atual do banco de dados. O algoritmo de merge que definimos [6] marca os conflitos na árvore e envia a visão *marcada* de volta para o usuário, para que ele resolva os conflitos. As atualizações são traduzidas apenas se não houver nenhum conflito.

4. Conclusões

Este artigo apresenta uma proposta para suportar transações longas em atualizações através de visões XML. Esta abordagem utiliza o PATAXÓ para gerar as visões XML e mapear as atualizações para a base relacional. Com o mecanismo proposto, o usuário não precisa estar conectado ao sistema para fazer as atualizações, o que é uma vantagem para aplicações móveis. Além disso, as atualizações ficam independentes do uso de uma linguagem de atualização, fazendo com que se tenha maior flexibilidade para atualizar as visões XML.

Neste trabalho, a pesquisa para a detecção e tratamento de *delta*, bem como o mapeamento deste para o PATAXÓ são tarefas já concluídas. Dentre as tarefas futuras está um estudo maior adaptações necessárias para utilização das técnicas de *delta* para o PATAXÓ e das formas de tratar os conflitos gerados pelo acesso concorrente.

Referências

[1] V. Braganholo, S. Davidson, and C. Heuser. From xml view updates to relational view updates: old solutions to a new problem. In *VLDB*, Toronto, Canada, 2004.

[2] Sudarshan S. Chawathe and Hector Garcia-Molina. Meaningful change detection in structure data. In *SIGMOD*, pages 26–37, Tuscon, Arizona, may 1997.

[3] Gregory Cobena, Serge Abiteboul, and Amelie Marian. Change-centric management of versions in an xml warehouse. In *VLDB*, 2001.

[4] M. Fernández, Y. Kadiyska, D. Suciu, A. Morishima, and W. Tan. Silkroute: A framework for publishing relational data in xml. In *ACM TODS 27*, 2002.

[5] J. Shanmugasundaram, J. Kiernan, E. Shekita, C. Fan, and J. Funderburk. Querying XML views of relational data. In *VLDB*, Roma, Itália, 2001.

[6] André Prisco Vargas. Suporte a transações longas em atualização através de visões xml, 2006. Dissertação de Mestrado, Universidade Federal do Rio Grande do Sul (em preparação).

[7] Yuan Wang, David J. DeWitt, and Jin-Yi Cai. X-diff: An effective change detection algorithm for XML documents. In *ICDE*, pages 519–530, Bangalore, India, March 2003.