UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO

ANDRÉ PRISCO VARGAS

Tratamento de Conflitos e Detecção de Deltas em Atualização através de Visões XML

Dissertação apresentada como requisito parcial
para a obtenção do grau de
Mestre em Ciência da Computação

Prof. Dr. Carlos Alberto Heuser
Orientador

Profa. Dra. Vanessa P. Braganholo
Co-orientadora

Porto Alegre, maio de 2007

CIP – CATALOGAÇÃO NA PUBLICAÇÃO

Vargas, André Prisco

Tratamento de Conflitos e Detecção de Deltas em Atualização através de Visões XML / André Prisco Vargas. – Porto Alegre: PPGC da UFRGS, 2007.

75 f.: il.

Dissertação (mestrado) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação, Porto Alegre, BR–RS, 2007. Orientador: Carlos Alberto Heuser; Co-orientadora: Vanessa P. Braganholo.

I. Heuser, Carlos Alberto. II. Braganholo, Vanessa P. III. Título.

UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
Reitor: Prof. José Carlos Ferraz Hennemann
Vice-Reitor: Prof. Pedro Cezar Dutra Fonseca
Pró-Reitora de Pós-Graduação: Prof[a]. Valquíria Linck Bassani
Diretor do Instituto de Informática: Prof. Flávio Rech Wagner
Coordenadora do PPGC: Prof. Luciana Porcher Nedel
Bibliotecária-Chefe do Instituto de Informática: Beatriz Regina Bastos Haro

AGRADECIMENTOS

São muitas as pessoas que preciso agradecer, sozinho seria impossível concluir este trabalho. Primeiramente a Deus, doador da vida e de toda Graça. A Ele a glória.

Aos meus pais, Valdir e Luiza, por terem me guiado e me ajudado de todas as formas que um filho pode ser ajudado e guiado. Desde a comemoração por receber a aceitação no PPGC ao dia de hoje não me faltou apoio e compreensão. Obrigado pelo amor de vocês.

À minha amada Patrícia, que com amor e zelo me ajudou em todos os momentos, me acompanhando sempre. Obrigado por ter entendido as semanas fora de casa, as dificuldades, a falta de tempo. A ti todo meu amor.

Ao meu orientador Carlos Heuser, pelo excelente trabalho e orientação. Seu conhecimento na Ciência da Computação e percepção para rumos de pesquisa são motivos de admiração. Obrigado pela oportunidade e pela paciência na escolha dos temas e nas revisões dos trabalhos. Obrigado por ser sempre acessível no trato com todos. À minha co-orientadora Vanessa Braganholo, que foi de fato incansável no seu trabalho. Obrigado pela ajuda na escolha do tema do trabalho e por ter acreditado nele, pela ajuda na escrita dos artigos. Que possas continuar a ser sempre assim, dedicada e bem sucedida em tudo que fazes.

Ao Grupo de Bancos de Dados da UFRGS pelas idéias, ajuda nos trabalhos, apresentações, artigos. Sempre proporcionando bom convívio. A todos do laboratório 215. Como é bom trabalhar num ambiente sério e leve.

Ao Marcelo, Rodrigo e Carlos, meus colegas de apartamento, por terem me ajudado sempre nos períodos em que estava em Porto Alegre. Obrigado por proporcionar um clima agradável, organizado e sobretudo divertido. Obrigado à Melissa e à Cristina, pelas jantas, pizzas e por serem grandes amigas. Ao ministério Alfa e Ômega, sobretudo ao Márcio, à Rosi e à Letícia, que me acolheram como um irmão e que me ensinaram o que é escolher a melhor parte desta vida.

Agradecimentos à FURG e ao CPD, por terem me concedido liberação parcial durante um ano de trabalho. Agradecimentos ao CNPq por ter me financiado desde minha graduação até o início deste trabalho.

Finalmente agradeço a todos que de uma forma ou outra contribuiram para o término desta etapa em minha vida. Muito obrigado.

SUMÁRIO

LISTA DE ABREVIATURAS E SIGLAS 6

LISTA DE FIGURAS 7

LISTA DE TABELAS 9

RESUMO 10

ABSTRACT 12

1 INTRODUÇÃO 13
1.1 Interface entre XML e Base de Dados Relacional 13
1.2 Detecção de Diferenças entre documentos XML 14
1.3 Tratamento de Conflitos 15
1.4 Cenários de Aplicação 15
1.4.1 Atualização de Bases de Dados Através de Formulários XML 15
1.4.2 Aplicações Móveis 16
1.5 Objetivo e Contribuições 17

2 TRABALHOS RELACIONADOS 19
2.1 PATAXÓ 19
2.1.1 UXQuery 19
2.1.2 Representação Interna da Estrutura da Visão XML 20
2.1.3 Atualizações no PATAXÓ 22
2.1.4 Linguagem de Atualização do PATAXÓ 23
2.2 Detecção de Diferenças 24
2.2.1 Estrutura dos Documentos Analisados 26
2.2.2 Modelo das Operações 26
2.2.3 Correspondência entre os Nodos 27
2.2.4 Linguagem de Representação do *Delta* 28
2.2.5 Visão Geral do X-Diff 32
2.2.6 Quadro Comparativo entre Algoritmos de Detecção de *Delta* 38
2.3 Controle de Concorrência, Conciliação e Tratamento de Conflito 39
2.3.1 Métodos de Controle de Concorrência 39
2.3.2 Tratamento de Conflito e Conciliação 40
2.4 Considerações finais 41

3 TRANSAÇÕES DESCONECTADAS 43
3.1 Detector de *Deltas* 45
3.1.1 X-DIFF 46
3.2 Gerenciador de Atualizações 46
3.3 Gerenciador de Transações 51

4 TRATAMENTO DE CONFLITOS 52
4.1 Tratamento de conflitos 53
4.2 Regras de Detecção de Conflitos - Modo Relaxado 53
4.2.1 Regra 1: Nodos Folha pertencentes ao mesmo elemento-estrela 54
4.2.2 Regra 2: *Sub-árvores estrela* dependentes 54
4.2.3 Regra 3: Nodo-estrela e seus nodos folha descendentes. 55
4.2.4 Regra 4: Nodos-estrela dependentes 56
4.2.5 Aplicação das Regras para Detecção dos Conflitos 57
4.3 Regras de Detecção de Conflitos - Modo de Chaves 58
4.3.1 Regra 5: Modificação concorrente de mesmo nodo folha 58
4.3.2 Regra 6: Nodos Chaves pertencentes ao mesmo elemento-estrela 58
4.3.3 Aplicação das Regras para Detecção dos Conflitos - Modo de Chaves . . . 59
4.4 Notificação do Usuário 59

5 IMPLEMENTAÇÃO DA ARQUITETURA 62
5.1 Locadora On-Line 63
5.2 Considerações Finais 65

6 CONCLUSÃO 68
6.1 Publicações 69
6.2 Implementação 70
6.3 Trabalhos Futuros 70

REFERÊNCIAS 71

LISTA DE ABREVIATURAS E SIGLAS

ACID	Atomicidade Consistência Isolação Durabilidade
ANSI	American National Standards Institute
BCNF	Boyce Codd Normal Form
CVS	Concurrent Version System
DTD	Document Type Definition
MVCC	Multiversion Concurrency Control
PATAXÓ	Permitindo ATualizações Através de visões Xml em bancos de dados relaciOnais
PDA	Personal Digital Assistant
SGBD	Sistema de Gerenciamento de Bancos de Dados
UFRGS	Universidade Federal do Rio Grande do Sul
W3C	World Wide Web Consortium
XID	Xyleme ID
XML	eXtensible Markup Language

LISTA DE FIGURAS

Figura 1.1: Cenário de Aplicação entre Empresas. 14
Figura 1.2: Esquema da base de dados de pedidos da empresa Fornecedora. . . . 16
Figura 1.3: Visão XML originalmente entregue à empresa Cliente. 16
Figura 1.4: Atualizações feitas sobre a base de dados. 17
Figura 1.5: Visão XML modificada pela empresa Cliente. 17

Figura 2.1: Definição em UXQuery da visão XML original. 20
Figura 2.2: *Query tree*. 21
Figura 2.3: Exemplo de *Query Tree* redundante. 22
Figura 2.4: Exemplo de visão XML redundante. 22
Figura 2.5: Visão XML que não preenche os requisitos para ser bem-aninhada (*well-nested*). 23
Figura 2.6: DTD referente às visões XML das Figuras 1.3 e 1.5 23
Figura 2.7: Exemplo de atualização usando linguagem de atualização do PATAXÓ. 23
Figura 2.8: Grafo de indução representando a comparação entre a visão original e a visão atualizada. 28
Figura 2.9: Grafo de indução filtrado, representando a comparação entre a visão original e a visão atualizada 28
Figura 2.10: Exemplo de duas versões de um documento XML. 30
Figura 2.11: Exemplo de *Delta* no MHDIFF. 31
Figura 2.12: Resultado do *delta* entre as versões a e b do documento 2.10, gerado pelo Algoritmo XyDiff 32
Figura 2.13: Resultado do *delta* gerado pelo DELTAXML a partir da visão original e da visão da Figura 2.14. 33
Figura 2.14: Exemplo de visão atualizada com um novo item foi inserido no início da lista. 34
Figura 2.15: Resultado do *delta* gerado pelo deltaxml a partir da visão original e da visão da Figura 2.14. 35
Figura 2.16: Resultado do *delta* gerado pelo X-Diff a partir da visão original e da visão da Figura 2.14. 35
Figura 2.17: Resultado do *delta* entre as visões original e atualizada, gerado pelo Algoritmo X-Diff. 37
Figura 2.18: Exemplo de casamento de nodos no X-Diff. 38

Figura 3.1: Arquitetura para a solução proposta. 44
Figura 3.2: Visões XML e *deltas* envolvidos na transação. 45
Figura 3.3: View' - Visão XML que representa o estado atual da base de dados. . 45

Figura 3.4: Resultado do *delta* entre as visões original e visão', gerado pelo Algoritmo X-Diff 46
Figura 3.5: Operações de atualização da visão original → visão atualizada; e da visão original → visão' 47
Figura 3.6: Exemplo de Operações de atualização entregue ao PATAXÓ. 49
Figura 3.7: Visão XML com modificações de valores de chave. 49
Figura 3.8: Árvore com um novo pedido, inserida na visão apresentada na Figura 3.7 49
Figura 3.9: Atualização da chave primária "BLUEPEN". 50

Figura 4.1: Exemplo de Tupla na base de dados relacional e na visão XML. Os elementos em negrito na visão XML correspondem a tupla hachurada na Tabela 54
Figura 4.2: Exemplo de Tupla na base de dados relacional e na visão XML. Os elementos hachurados representam parte de mais de uma tupla na relação 55
Figura 4.3: Visão retornada pelo cliente, na qual o elemento BLUEPEN é excluído. 56
Figura 4.4: Resultado do algoritmo de *merge*. 61

Figura 5.1: Módulos responsáveis pela detecção de conflitos, contidos no módulo Controle de Transação. 62
Figura 5.2: Exemplo de repositório temporário. 63
Figura 5.3: Base de Dados de Exemplo. 64
Figura 5.4: Definição em UXQuery da visão XML de reservas de filmes. 64
Figura 5.5: Visão XML original da consulta de reservas de filmes. 65
Figura 5.6: Visão XML com as reservas de filmes, atualizada pelo cliente. 65
Figura 5.7: Operações detectadas pelo sistema, e notificação sobre conflitos. . . 65
Figura 5.8: Definição em UXQuery da visão XML de catálogo de filmes de suspense. 66
Figura 5.9: Visão XML original da consulta de catálogo de filmes. 66
Figura 5.10: Visão XML de catálogo de filmes, atualizada pelo usuário. 66
Figura 5.11: Atualizações feitas sobre a base de dados da locadora. 66
Figura 5.12: Operações detectadas pelo sistema, e notificação sobre conflitos. . . 67

LISTA DE TABELAS

Tabela 2.1: Tipos abstratos de nodos da *query tree* 21
Tabela 2.2: Exemplos de assinatura de nodos no X-Diff 36
Tabela 2.3: Quadro comparativo entre algumas técnicas de detecção de diferenças 42

Tabela 4.1: Pré-requisitos para execução no modo “relaxado”. Na horizontal há operações em *visão’* e na vertical na visão atualizada. 57
Tabela 4.2: Pré-requisitos para execução no modo de chaves. Na horizontal há operações em *visão’* e na vertical na visão atualizada. 59

RESUMO

A linguagem XML tem se tornado um padrão no intercâmbio de informações na Web. No entanto, a maioria das organizações continua a armazenar seus dados em bancos de dados relacionais. Diante deste ambiente, surge a necessidade de se construir aplicações que permitam às empresas o intercâmbio de informações via XML, mas sem que estas empresas tenham que migrar suas bases relacionais. Neste trabalho, é apresentada uma técnica para importar e exportar documentos XML, focada em cenários entre empresas onde visões XML são extraídas de uma base de dados relacional e enviadas via Web (ou qualquer outro meio) para outra aplicação que as edita e as retorna. Através da edição da visão XML, a própria base de dados relacional é modificada, atualizando assim os dados da empresa. A base de dados relacional deve ser atualizada com as novas informações da visão XML. Neste tipo de transação tem-se as seguintes considerações:

1. A visão XML pode ser atualizada por qualquer aplicação. Editores de texto, banco de dados XML e aplicações específicas estão entre as aplicações que podem atualizá-la.

2. A aplicação que recebe a visão XML fica de posse dela por um período não determinado, podendo inclusive não retornar a visão. Durante este período a aplicação pode ficar desconectada da base de dados geradora da visão XML.

3. Enquanto a visão XML está sendo editada, outras aplicações podem acessar e atualizar a base de dados.

4. Não existe conhecimento semântico específico sobre os dados contidos na visão XML.

Portanto, para este tipo de aplicação, não é realista fazer um controle de concorrência baseado em bloqueios das tuplas contidas na visão. Da mesma forma, não é possível esperar que o usuário expresse, através de uma linguagem ou formato padrão, as alterações efetuadas na visão XML. Ocorre neste ambiente dois problemas: (i) identificar as modificações feitas na visão e (ii) identificar e resolver conflitos que possam ser causados por modificações na base de dados durante a transação. O objetivo deste trabalho é desenvolver uma técnica para exportação e importação de visões XML que minimize estes dois problemas. Neste trabalho é proposta uma arquitetura que utiliza algoritmos de detecção de diferenças em documento XML e uma extensão do sistema de atualização de visões XML PATAXÓ, um sistema já existente de importação e exportação de documentos XML em bases relacionais. Também é apresentado, para o módulo de gerenciamento de transações da arquitetura, uma proposta de detecção e tratamento de conflitos baseada em regras geradas apenas sobre a estrutura da visão XML.

Palavras-chave: Atualização utilizando visões, Detecção de Diferenças, Visões XML, Tratamento de Conflitos.

Conflict Resolution and Difference Detection in Updates through XML views

ABSTRACT

XML has become the standard format for exchanging information on the Web. However, many organizations continue storing their data in relational databases. In this context, it becomes necessary to build applications that allow companies to exchange information via XML without having to share their relational databases.

This thesis introduces a technique for exporting and importing XML documents from relational databases in a scenario of business to business (B2B) applications. In the considered scenario, a XML view is extracted from a relational database and then sent via the Web (or any other means) to another separate application where the information is edited and then sent back after a certain period of time. Changes introduced on the XML view must be mapped into updates on the relational database, thus implementing business transactions etc.

These types of transaction have the following considerations.

1) Any application may be used to edit the XML view. There is no need for a specific application to update it.

2) The application that receives the XML view retains its view during an indeterminate period of time having the option of not returning the view. During this period of time, the application is disconnected from the relational DBMS.

3) During the period of time in which the XML view is being updated, other applications may access and update the database.

4) There is no specific semantic knowledge regarding the data contained within the XML view.

Therefore, with this type of application, it is not realistic to have a pessimistic concurrency control mecahnism based on data locking. In the same way, it is not realistic to expect a user to express the updates contained within the XML view through a specific language. Thus there are two main problems to be solved. Firstly, the identification of which modifications were made within the view, and secondly, identifying and solving conflicts that may arise due to updates in the database during the transaction. The objective of this thesis is to develop a technique for exporting and importing XML views that addresses these two problems. The thesis describes an approach to detect XML differences, as well as a extension of Pataxó, an already existing XML import/export system for relational databases. Additionally, the thesis describes the transaction management module that implements the proposed approach for detecting and handling conflicts due to updates on the XML view.

Keywords: Updates through views, Delta Detection, Conflict Resolution.

1 INTRODUÇÃO

Desde sua criação em 1998, a linguagem XML (W3C, 2004) está tornando-se cada vez mais um padrão no intercâmbio de informações entre aplicações empresariais. Dessa forma, é bastante comum o cenário onde a empresa armazena seus dados em um banco de dados relacional e os exporta na forma de documentos XML. Assim, a empresa pode fazer uso de uma tecnologia já amadurecida, eficiente, utilizando sistemas legados e comunicando-se através de um padrão aberto e flexível.

Por exemplo, suponha que uma empresa Cliente deseja comprar produtos de uma empresa Fornecedora (Figura 1.1). A empresa Fornecedora disponibiliza seus produtos através de um formulário de pedidos em XML. Um funcionário da empresa Cliente recebe via Web em um dispositivo móvel, como um PDA (*Personal Digital Assistant*), o formulário de pedidos, que inicialmente é uma visão XML vazia, contendo apenas os elementos estruturais. O funcionário preenche o formulário e, em seguida, faz a devolução deste à empresa Fornecedora. O caso pode ter complicações se, após ter enviado o pedido, a empresa Cliente decidir alterá-lo. Nesse caso, a visão é reenviada e a empresa Cliente pode fazer suas atualizações. A empresa Fornecedora recebe o novo pedido e precisa:

- Descobrir quais foram as atualizações feitas pela empresa Cliente através da visão alterada.

- Alterar o pedido na base de dados.

Este exemplo e os cenários que serão vistos adiante, possuem alguns requisitos que devem ser ressaltados. O primeiro deles é que não deve ser necessária nenhuma aplicação específica na máquina da empresa Cliente. Desse modo, não há conhecimento de como a visão será alterada, podendo ser através da edição do documento XML diretamente ou através de uma aplicação avançada. Outro requisito importante é que a empresa Cliente pode ficar desconectada da empresa Fornecedora entre o período de recebimento e envio da visão. Este período em que a visão é analisada e atualizada localmente pelo cliente é indeterminado e pode durar horas. Assim, fica inviável uma abordagem baseada em bloqueios dos registros(ULLMAN, 2002).

1.1 Interface entre XML e Base de Dados Relacional

Para o tipo de aplicação proposto, um dos problemas a ser resolvido é a utilização de uma interface entre as visões XML e a base de dados relacional.

Na literatura, existem diversos trabalhos com o objetivo de extrair documentos XML a partir de bases de dados relacionais. Entre eles estão o XPeranto (SUBRAMANIAN, 2000) e o SilkRoute (FERNÁNDEZ, 2000). Estes trabalhos, porém, tratam apenas parte

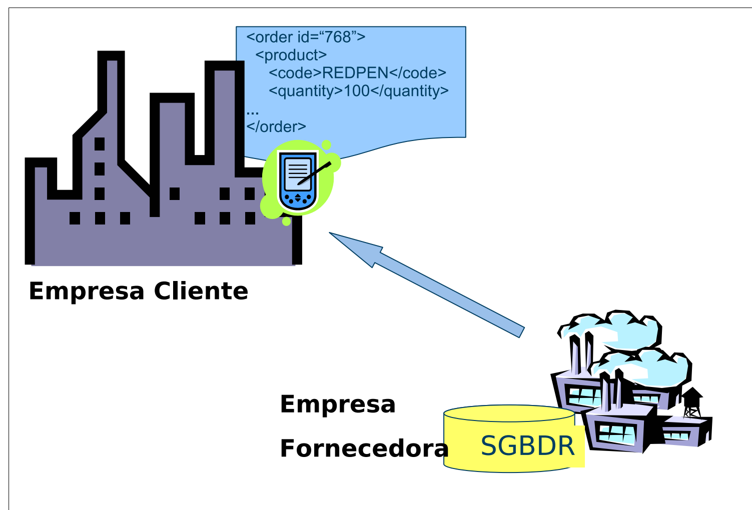


Figura 1.1: Cenário de Aplicação entre Empresas.

do problema, apresentando técnicas para extrair visões XML, mas não abordando técnicas para atualizar tais visões (CAREY, 2000).

O sistema PATAXÓ (BRAGANHOLO, 2004) é uma abordagem usada tanto para gerar visões XML quanto para atualizar bases de dados relacionais através de visões XML. As visões são geradas através da linguagem UXQuery (BRAGANHOLO, 2004), uma extensão do XQuery (BOAG, 2005), e são atualizadas através de uma linguagem de atualização própria.

No entanto, o sistema PATAXÓ possui algumas características que podem limitar sua utilização nos cenários propostos na seção 1.4:

- Neste sistema, as atualizações são feitas através de uma linguagem própria, de forma que estas não podem ser feitas diretamente no documento XML. Neste caso, o usuário ou aplicação cliente deve conhecer a linguagem para efetuar as operações.

- As transações utilizadas são as do SGBD. Como a maioria dos SGBDs utiliza transações ACID, baseadas em bloqueios, sua utilização torna-se impraticável nos cenários apresentados, onde o usuário faz suas alterações estando desconectado do sistema e onde tais alterações demoram um período indeterminado.

Diante destas limitações, torna-se necessária a utilização de um mecanismo estendido para atualização das visões XML.

1.2 Detecção de Diferenças entre documentos XML

Neste trabalho o interesse não está em um documento XML específico, mas na detecção de quais alterações foram feitas sobre ele. Na literatura existem diversas técnicas de detecção de diferenças entre documentos (*delta*). Softwares de controle de versão como CVS (CEDERQVIST, 2003) e *Subversion* (COLLINS-SUSSMAN, 2007), ou de controle de arquivos como GNU/DIFF (EGGERT, 2007) são bastante utilizados no controle de arquivos texto. Porém, tais aplicações lidam com textos planos, não levando em conta a estrutura dos arquivos.

Os sistemas apresentados neste trabalho (veja seção de trabalhos relacionados) foram construídos para detectar diferenças em documentos XML, estruturados em árvore. Estas técnicas são utilizadas em sistemas de controle de versões, detecção de atualizações em páginas Web, indexação de bibliotecas digitais, etc.

Cada técnica possui características que influenciam na qualidade dos resultados e no desempenho. Serão apresentados técnicas como MHDIFF (CHAWATHE, 1997), X-Diff (WANG, 2003), XyDiff (COBENA, 2002).

1.3 Tratamento de Conflitos

O controle de concorrência em bancos de dados é um tema já bastante conhecido e abordado na literatura. Transações ACID, com utilização de bloqueios de tuplas, são utilizadas na maioria dos SGBDs comerciais. Para transações muito grandes, formadas por centenas de operações, outras técnicas de bloqueios também são tratadas na literatura (ULLMAN, 2002).

Porém, neste trabalho, as transações são desconectadas, ou seja, o usuário faz as atualizações na visão XML que está na sua máquina, desconectada da base de dados central. Tal ambiente apresenta-se diferente do ambiente tradicional por tornar inviável o uso de bloqueios, visto que seria impraticável a base de dados ficar inacessível durante todo o período em que o usuário está de posse de sua visão XML.

As técnicas usadas em contextos nos quais as transações são formadas por centenas de operações (conhecidas na literatura por "transações longas") também não se aplicam. Nelas, o sistema já conhece desde o início da transação quais são as operações envolvidas. Além disso, a maior demanda de tempo decorre do grande número de operações envolvidas. Por outro lado, as transações desconectadas não possui técnicas para grande número de operações, uma vez que este número pode estar limitado à capacidade dos dispositivos móveis. Neste caso o tempo da transação é grande devido ao período em que o usuário está alterando sua visão XML. Neste período o sistema ainda desconhece quais são as operações envolvidas na transação.

Portanto, neste contexto outras técnicas para controle de concorrência devem ser consideradas. Por exemplo, o SGBD PostgreSQL utiliza uma técnica baseada em múltiplas versões de uma tupla, de forma que uma escrita não bloqueia uma leitura e uma leitura não bloqueia uma escrita (CAREY, 1986; POSTGRESQL, 2007).

Em algumas aplicações, são adotadas técnicas em que as transações não são bloqueadas, podendo ocorrer atualizações simultâneas na mesma tupla. Nestes casos, alterações concorrentes geram os **conflitos**, que devem ser tratados. Existem algumas estratégias para tratamento de conflitos, que usam principalmente o cancelamento das operações, operações de compensação e identificação de conflito para ser tratado pelo usuário. Em (PHATAK, 1999; KLIEB, 1996; SATYANARAYANAN, 1990) são apresentadas técnicas de tratamento de conflito. No decorrer deste trabalho, são apresentadas algumas técnicas para transações desconectadas, além da abordagem desenvolvida.

1.4 Cenários de Aplicação

Para melhor apresentar a proposta deste trabalho, serão descritos alguns cenários de aplicação, também utilizados no decorrer do mesmo.

1.4.1 Atualização de Bases de Dados Através de Formulários XML

Uma companhia deseja receber informações via Web. Para isto, ela deseja gerar formulários XML, por exemplo formulários XForms (DUBINKO, 2003), para que sua base de dados seja automaticamente preenchida pelos clientes. O formulário (um documento XML) é gerado e enviado ao cliente, que o preenche e o retorna à companhia (o processo

```
Customer (custId, name, address),
   primary key (custId)
Product (prodId, description, curPrice),
   primary key (prodId)
Order (numOrder, date, custId, status),
   primary key (numOrder),
   foreign key (custId) references Customer
LineOrder (numOrder, prodId, quantity, price),
   primary key (numOrder, prodId),
   foreign key (prodId) references Product,
   foreign key (numOrder) references Order
```

Figura 1.2: Esquema da base de dados de pedidos da empresa Fornecedora.

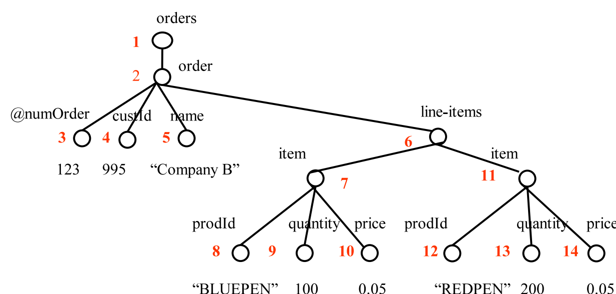


Figura 1.3: Visão XML originalmente entregue à empresa Cliente.

é feito, de forma transparente, pelo navegador Web).

Neste contexto, a aplicação deve:

- Gerar o documento XML a partir da base de dados relacional;
- Receber o documento XML alterado (formulário preenchido) e detectar quais foram as alterações feitas pelo cliente;
- Atualizar a base de dados através das alterações detectadas.

Neste cenário não há conflitos, uma vez que apenas inserções serão feitas e as informações inseridas por um cliente não são compartilhadas por outro cliente. Além disso, pode-se presumir que as informações relacionadas, como chaves estrangeiras, não serão alteradas por outras aplicações.

1.4.2 Aplicações Móveis

Este cenário é formado pelas empresas Cliente e Fornecedora, como apresentadas no início deste capítulo. A empresa Fornecedora possui uma base de dados relacional de pedidos, produtos e clientes. O esquema desta base de dados é apresentado na Figura 1.2. A empresa Cliente requisitou a empresa Fornecedora o retorno de um pedido já efetuado a fim de alterá-lo. O resultado da requisição pode ser um documento XML como apresentado na Figura 1.3 (os números ao lado dos nodos são usados para referenciá-los ao longo do texto).

Enquanto a empresa Cliente analisa seu pedido e está decidindo quais mudanças deseja efetuar, a base de dados relacional da empresa Fornecedora está sendo alterada, com as operações apresentadas na Figura 1.4. Estas atualizações podem ter sido feitas através de outra visão XML ou diretamente na base de dados. As atualizações sobre *LineOrder* afetam a visão XML que está sendo analisada pela empresa Cliente, uma vez que o preço da *blue pen*, um dos produtos que a empresa Cliente solicitou, está sendo modificado.

```
//Aumentar o preço da "blue pen"
UPDATE Product
SET curPrice = 0.10
WHERE prodId = "BLUEPEN";

UPDATE "LineOrder"
SET price = 0.10
WHERE prodId = "BLUEPEN"
AND numorder = (SELECT numorder from "Order" where status = "open");
```

Figura 1.4: Atualizações feitas sobre a base de dados.

Após um certo período, a empresa Cliente continua decidindo o que deseja mudar em seu pedido. Após cinco horas e, sem saber que o preço do produto *blue pen* foi dobrado, ela decide aumentar a quantidade de *blue pens* para 200 e de *red pens* para 300 e inserir um novo produto (100 *notebooks* (NTBK)) no pedido. Ao fazer as modificações, a empresa Cliente envia a visão XML como apresentado na Figura 1.5 (as alterações estão em negrito).

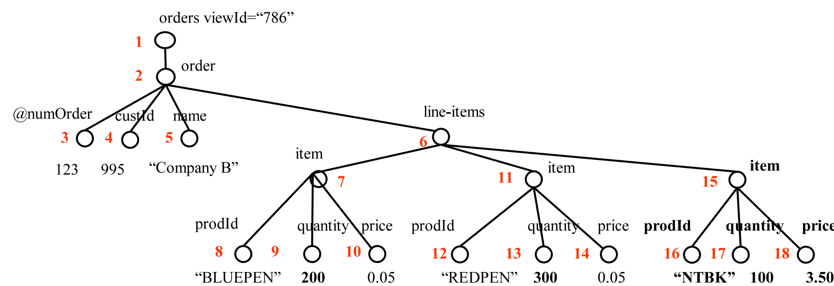


Figura 1.5: Visão XML modificada pela empresa Cliente. As alterações estão em negrito.

Quando a empresa Fornecedora receber a visão atualizada, ela deve:

- Gerar o documento XML via PATAXÓ;
- Receber o documento XML alterado (formulário preenchido) e detectar quais foram as alterações feitas pelo cliente;
- Detectar como as atualizações feitas na base de dados, como as da Figura 1.4, podem afetar as atualizações da empresa Cliente e quais são exatamente esses conflitos;
- Decidir como tratar os conflitos;
- Atualizar a base de dados via PATAXÓ.

1.5 Objetivo e Contribuições

O objetivo deste trabalho é propor um mecanismo de exportação/importação de visões, resolvendo os problemas listados anteriormente. Neste trabalho o PATAXÓ é usado para gerar os documentos XML a partir da base de dados e realizar as atualizações do usuário na base de dados relacional. Entre as contribuições deste trabalho estão:

- Uma técnica para descoberta de operações na base de dados relacional, baseada nas modificações feitas no documento XML. Esta técnica é adaptada às visões geradas pelo PATAXÓ e a sua linguagem de atualização.

- Uma abordagem para verificar o estado da base de dados durante a transação. Esta é feita comparando o estado da base de dados no início da transação e no momento em que a visão retorna ao sistema.
- Uma técnica para tratamento de conflitos baseada na estrutura da visão XML.
- Um algoritmo para geração de uma visão de retorno ao usuário, enfatizando as atualizações efetuadas e as que provocaram conflitos.

O restante deste trabalho é organizado da seguinte forma. O Capítulo 2 apresenta o sistema PATAXÓ, que é utilizado como base neste trabalho. Neste mesmo capítulo também é feita uma revisão bibliográfica de trabalhos relacionados nas áreas de tratamento de diferenças em documentos XML e tratamentos de conflitos. O Capítulo 3 apresenta a arquitetura do sistema proposto e as técnicas desenvolvidas para utilização das diferenças entre documentos XML. O Capítulo 4 apresenta a técnica desenvolvida para o tratamento dos conflitos na base de dados, envolvendo a identificação, o tratamento propriamente dito e a forma de relatar à aplicação cliente. O Capítulo 5 apresenta a implementação de um protótipo e a utilização de um exemplo. Finalmente, no Capítulo 6, é apresentada a conclusão.

2 TRABALHOS RELACIONADOS

A atualização de bases de dados relacionais através de importação e exportação de visões XML necessita de técnicas para geração das visões, detecção das alterações efetuadas sobre a visão e atualização da base de dados relacional. Técnicas para controle de concorrência no acesso aos dados também são necessárias. O objetivo deste capítulo é apresentar alguns trabalhos relacionados a tais técnicas. Primeiramente será apresentado o sistema PATAXÓ, a forma como é gerada a visão XML e a forma como as alterações são mapeadas para o modelo relacional. Na Seção 2.2 é apresentada uma série de algoritmos para detectar alterações em dados estruturados em árvore (sendo ou não XML), focalizando as características mais importantes para este trabalho. Por fim, a Seção 2.3 apresenta trabalhos na área de tratamento de conflitos.

2.1 PATAXÓ

O sistema de atualização PATAXÓ (BRAGANHOLO, 2006) é capaz de construir visões XML a partir de bases de dados relacionais e mapear as atualizações sobre a visão XML para operações sobre esta base de dados. A proposta consiste em mapear uma visão XML para uma (ou várias) visão relacional correspondente, e mapear as alterações sobre a visão XML para alterações sobre as visões relacionais. As operações sobre a visão relacional são, então, mapeadas para as operações sobre a base de dados utilizando técnicas já conhecidas de atualização de visões relacionais (BRAGANHOLO, 2004).

De uma forma geral, o processo de geração e atualização das visões é o seguinte: a definição da visão XML é expressa através da linguagem UXQuery (BRAGANHOLO, 2003), uma versão adaptada da linguagem XQuery (BOAG, 2005), e internamente mapeada para uma *query tree* (BRAGANHOLO, 2004), que é uma estrutura auxiliar para identificar o mapeamento entre a base de dados de origem e a visão XML. A *query tree* é usada para mapear a visão XML para uma ou mais visões relacionais. A atualização é feita mapeando-se as atualizações sobre o documento XML para atualizações sobre visões relacionais. Através destas, a base de dados relacional é atualizada, utilizando o trabalho de Dayal e Bernstein (DAYAL, 1982).

Nas próximas seções serão apresentados maiores detalhes sobre este sistema.

2.1.1 UXQuery

A UXQuery é uma linguagem desenvolvida para o sistema PATAXÓ para definição das visões XML. Semelhante à XQuery (BOAG, 2005), esta linguagem possui adaptações próprias para a geração de visões atualizáveis originadas de bases relacionais. Uma especificação mais abrangente pode ser vista em (BRAGANHOLO, 2004).

```
<orders>
{for $c in table('Customer'),
     $o in table('Order')
 where $c/custId = $o/custId
 return
   <order numOrder='{$o/numOrder/text()}'>
            <custId>{$c/custId/text()}</custId>
            <name>{$c/name/text()}</name>
            <line-items>
            {for $l in table('LineOrder')
             where $o/numOrder = $l/numOrder}
             return
                <line-item>
                   <prodId>{$l/prodId/text()}</prodId>
                   <quantity>{$l/quantity/text()}</quantity>
                   <price>{$l/price/text()}</price>
                </line-item>
       }
     </line-items>
   </order>
}
</orders>
```

Figura 2.1: Definição em UXQuery da visão XML original, apresentada na Figura 1.3.

A Figura 2.1 apresenta a definição em UXQuery da visão XML utilizada no cenário exemplo (Figura 1.3). A única diferença desta consulta para uma consulta em XQuery é a utilização da operação *table*. A operação *table* liga uma variável a uma tabela relacional. Por exemplo, a operação *for $l in table('LineOrder') where $o/numOrder = $l/numOrder* consulta a tabela *LineOrder* e recolhe as tuplas cujo *numOrder* esteja presente na tabela *Order* (que aparece anteriormente na consulta). Em outras palavras, a operação consulta cada linha de pedido que pertença ao número de pedido em questão.

2.1.2 Representação Interna da Estrutura da Visão XML

A definição da visão XML é representada internamente no PATAXÓ por uma estrutura de dados denominada *Query Trees* (BRAGANHOLO, 2004). As *Query Trees* representam a forma como os dados originados da base de dados relacional serão inseridos na estrutura da visão XML. A *query tree* apresentada na Figura 2.2 foi gerada a partir da expressão UXQuery apresentada na Figura 2.1.

Cada nodo da *query tree* possui um tipo específico, conforme apresentado na Tabela 2.1 (BRAGANHOLO, 2004). Os nodos τ_N e τ_T (no exemplo, *order* e *item*) são chamados de **nodos-estrela**. Um nodo-estrela gera uma coleção de elementos, que possuem os dados de uma tupla da base de dados. A fonte dos dados está expressa no próprio conteúdo do nodo-estrela e pode ser uma tabela ou a junção de várias tabelas (na *query tree* mencionada acima, o nodo *order* é a junção das tabelas *Customer* e *Order*). Cada elemento da coleção gerada pelo nodo-estrela é chamado de **sub-árvore estrela** e a raiz de cada sub-árvore estrela da coleção (por exemplo, um elemento *item*), é chamada de **elemento-estrela**.

2.1.2.1 Propriedades das Query Trees

O PATAXÓ utiliza a proposta de Dayal e Bernstein (DAYAL, 1982) para atualizar o banco de dados relacional a partir das visões XML, portanto a construção de suas visões deve possuir certas características a fim de torná-las atualizáveis a partir desta proposta. Um estudo mais aprofundado sobre atualizabilidade das visões geradas pelas *Query Trees* pode ser encontrado em (BRAGANHOLO, 2004).

De uma forma resumida, a lista abaixo apresenta as propriedades que devem ser re-

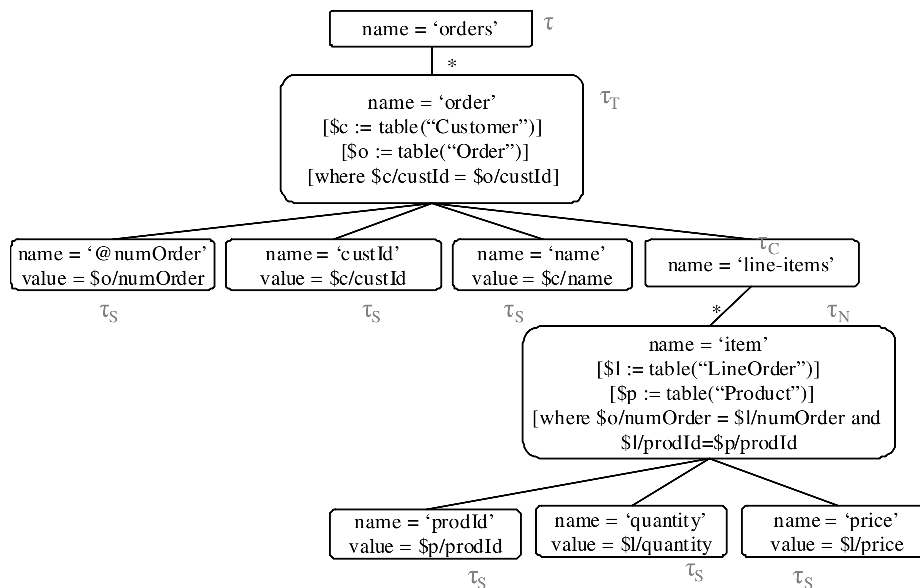


Figura 2.2: *Query tree* referente ao exemplo do cenário da seção 1.4.2.

Tabela 2.1: Tipos abstratos de nodos da *query tree* (BRAGANHOLO, 2004)

Tipo	Nodo
τ	nodo-raiz da *query tree*
τ_S	Nodo folha da árvore
τ_C	Nodos que não são folha e que não são do tipo nodo-estrela
τ_N	Nodos-estrela, que possuem como filhos apenas nodos folha ou τ_c
τ_T	Nodos-estrela, que não sejam do tipo τ_n

forçadas:

1. O esquema do banco de dados relacional utilizado deve estar em BCNF (*Boyce Codd Normal Form*)(ULLMAN, 2002)

2. Os valores dos nodos folha devem ser preenchidos unicamente com valores da base de dados relacional. Os nodos não podem ser preenchidos com constantes nem com valores gerados a partir de aplicações de funções.

3. Os dados da visão relacional devem estar unicamente em nodos folha do XML.

4. Não deve haver redundância de valores da base de dados na visão XML, ou seja, um valor de um atributo da base de dados somente deve aparecer em um único nodo folha da *Query tree*. As Figuras 2.3 e 2.4 apresentam uma *query tree* e a visão gerada por ela. A visão é redundante, visto que a junção de *LineOrder* com *Product* faz com que a descrição do produto *blue pen* apareça mais de uma vez na visão XML. A atualização de dados redundantes poderia gerar inconsistências. No exemplo, se apenas um *blue pen* for atualizado, a visão XML terá dois valores diferentes para *Product.description*.

5. A *query tree* deve ser bem-aninhada (*well-nested*), ou seja, se uma relação $R1$ está ligada a uma relação $R2$ por uma chave estrangeira, então o elemento que contém os nodos folha vindos de $R2$ deve ser ancestral do elemento que contém os nodos

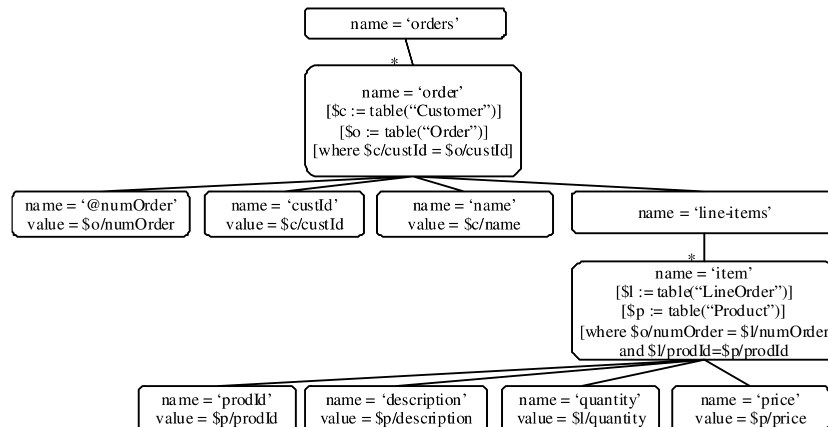


Figura 2.3: Exemplo de *Query Tree* redundante. O item $\$p/description$ não deveria se repetir entre os elementos *order*.

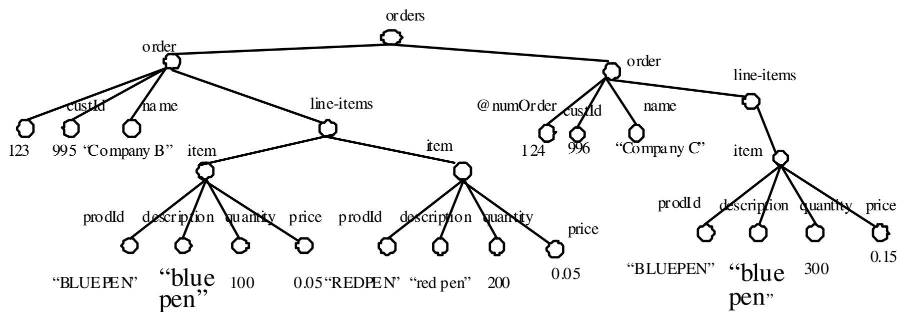


Figura 2.4: Exemplo de visão XML redundante, originado da *query tree* da Figura 2.3

folha de $R1$. A Figura 2.5 apresenta um exemplo de visão não-atualizável. Na base de dados relacional, *LineOrder* está relacionada por uma chave estrangeira a *Order*. Logo, os elementos originados da relação *Order* devem ser ancestrais dos elementos originados da relação *LineOrder*. Porém, não é isso o que acontece neste exemplo, o que torna a visão não-atualizável. O atributo *numOrder*, que vem da relação *Order*, não está em um nodo-ancestral ao nodo *item*. O nodo *item* é um nodo que contém os atributos *price* e *quantity*, que vêm da tabela *LineOrder*. Visões XML estruturadas de forma não bem-aninhada não são atualizáveis porque também podem gerar redundâncias e potenciais inconsistências. A visão apresentada na Figura 2.5 (não-atualizável) pode ser comparada com a visão XML original (atualizável) apresentada na Figura 1.3.

2.1.3 Atualizações no PATAXÓ

As atualizações das visões XML no PATAXÓ seguem a proposta de Dayal e Bernstein (1982) e, portanto, devem seguir suas restrições que abrangem não só a construção da visão como o conjunto de atualizações existentes. Esta seção apresenta algumas restrições que serão pertinentes no decorrer deste trabalho. Um estudo mais completo sobre a atualizabilidade das visões XML pode ser visto em (BRAGANHOLO, 2004).

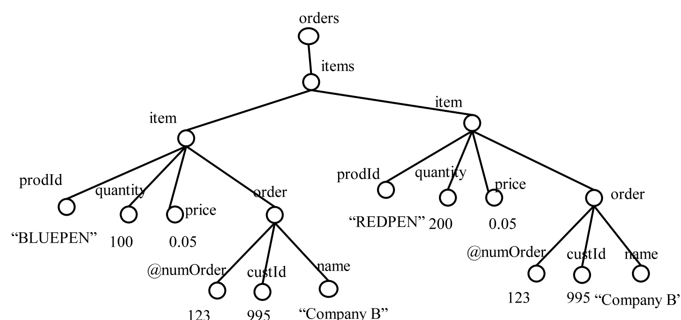


Figura 2.5: Visão XML que não preenche os requisitos para ser bem-aninhada (*well-nested*).

```
<!ELEMENT orders (order)*>
<!ATTLIST orders viewId CDATA #FIXED "786">
<!ELEMENT order (custId, name, line-items)>
<!ATTLIST order numOrder CDATA #REQUIRED>
<!ELEMENT line-items (item*)>
<!ELEMENT item (prodId, quantity, price)>
```

Figura 2.6: DTD referente às visões XML das Figuras 1.3 e 1.5

```
t = modify
Δ = "200"
ref = orders/order/item[@numOrder="123" and
      custId="995"]/item[prodId="BLUEPEN"]/quantity
```

Figura 2.7: Exemplo de atualização usando linguagem de atualização do PATAXÓ.

Alteração do Esquema Uma vez definida a visão XML, a estrutura desta é definida por uma DTD gerada pelo PATAXÓ. A Figura 2.6 apresenta a DTD referente às Figuras 2.1 e 2.2. Quando a visão XML é enviada à aplicação cliente, a DTD também é enviada. O PATAXÓ não considera alterações estruturais na base de dados, portanto todas as atualizações sobre a visão XML devem estar de acordo com as definições da DTD.

Modificações Visto que não devem haver alterações na DTD, modificações só podem ocorrer nos nodos folha, e somente em seus valores (textos ou valores de atributos). Alterações em nomes de atributos ou no nome dos elementos não são permitidas.

Inserções e Exclusões Inserções e exclusões só podem ser feitas em elementos-estrela. Nodos folha ou de outro tipo não podem ser inseridos ou excluídos, uma vez que geraria mudanças estruturais tanto no esquema XML quanto no esquema relacional.

2.1.4 Linguagem de Atualização do PATAXÓ

As atualizações no PATAXÓ são expressas através de uma linguagem simplificada, conforme exemplificado na Figura 2.7. Ela é formada por uma tripla $< t, \Delta, ref >$, onde:

1. t é o tipo de atualização a ser efetuada. Existem três tipos de atualização, *modify*

para modificação de conteúdo, *insert* para inserção de nodo e *delete* para exclusão de nodo.

2. Δ é o nodo conteúdo a ser colocado na visão XML. Pode ser um texto simples ou uma sub-árvore XML.

3. ref é uma expressão XPath que aponta para o nodo a ser atualizado. O XPath deve possuir um caminho válido e suas operações de qualificação de caminho devem ser sempre de igualdade, por exemplo, um caminho do tipo */orders/order[@numOrder > 100]* não será aceito pelo PATAXÓ.

Após serem escritas as atualizações nesta linguagem, o PATAXÓ gera um mapeamento de cada alteração para atualizações SQL, para serem efetuadas na base de dados relacional.

2.2 Detecção de Diferenças

Em muitos casos o interesse em um conjunto de dados não está somente na sua última versão, mas também nas versões anteriores e nas diferenças entre elas. As aplicações vão desde a descoberta de diferença entre arquivos texto simples à descoberta de diferenças em bases de dados versionadas.

Neste trabalho, as operações sobre a visão XML não são feitas através de nenhuma linguagem ou aplicação específica, mas baseadas apenas na edição direta do documento XML. Logo, o interesse não está em um documento XML específico, mas nas alterações que foram feitas sobre este documento, pois é a partir delas que serão geradas as operações de atualização na base de dados. Da mesma forma, para a detecção dos conflitos (como será visto mais adiante) não é suficiente conhecer somente o estado atual da base de dados, mas se ocorreram e quais foram as alterações no estado da base de dados desde que a visão foi entregue ao usuário.

Para que se detecte as alterações geradas, a abordagem adotada é formar versões de um documento XML. A primeira versão representa a visão originalmente entregue para o usuário e a segunda versão a alterada por ele. Dessa forma o problema resume-se em descobrir a diferença entre duas versões de um documento XML, que, conforme é visto ao longo desta seção, é um problema já abordado na literatura.

Seja D_x um estado de uma estrutura de dados e op uma operação de atualização definida sobre esta estrutura. Escreve-se $op(D_x)$ para denotar o resultado da aplicação de op sobre D_x. Portanto, um *delta* $\Delta = op_1, op_2...op_n$ é um conjunto de operações de atualização (também conhecido por *script*) sobre uma estrutura de dados tal que $\Delta(D_x) = D_y$.

Existem diversas aplicações que lidam com detecção de diferenças entre dados, como por exemplo, sistemas de controle de versão de arquivos fonte, como CVS (CEDERQVIST, 2003), *Subversion* (COLLINS-SUSSMAN, 2007) e o software Diff (EGGERT, 2007) do Linux, que também é popular. Estas aplicações, no entanto, lidam com documentos não-estruturados, levando em conta apenas as informações e não a forma como estas estão estruturadas.

A detecção do *delta* entre dados estruturados em árvores tem cada vez mais aplicações, uma vez que dados em HTML e XML, que podem ser modelados como árvores, têm se tornado um padrão na Web e na integração de sistemas. A detecção do *delta* possui diversas aplicações, entre elas:

```
detectDelta(XML_inical, XML_final)

Analisa Nodos de XML_inicial
Analisa Nodos de XML_final
para cada n_i em XML_inicial faça
   Achar correspondente em XML_final
fim para
para cada n_f em XML_final faça
   Achar correspondente em XML_inicial
fim para
Montar Grafo de Correspondência
se achou mais de um correspondente então
   Usar melhor alternativa (seguindo heurística específica de cada técnica)
fim se
se Se algum n_i não tem correspondente em XML_final então
   Marcar n_i como excluído
fim se
se Se algum n_f não tem correspondente em XML_inicial então
   Marcar n_f como inserido
fim se
para cada n_i e n_f que possui correspondência faça
   Descobrir operação op tal que op(n_i) = n_f
fim para
Ordenar Operações (Esta etapa é opcional e dependente da técnica específica)
Retorna lista de Operações
```

Algoritmo 1: Algoritmo geral usado nas técnicas de detecção de *deltas*

1. Bases de dados versionadas: Uma base de dados histórica, por exemplo, pode armazenar dados XML ou HTML. O sistema pode capturar a última versão e gerar os *deltas* entre as versões. Tendo uma versão é possível se fazer consultas a outras através do *delta*, mesmo que estas não estejam armazenadas na base.

2. Indexação: O projeto Xyleme (XYLEME, 2001) aborda a utilização da detecção de mudanças para indexar por palavras grandes volumes de documentos XML.

3. Monitoramento de Mudanças: Podem ser feitos sistemas que ficam visitando sites da Web, esperando uma determinada alteração. Por exemplo, o sistema pode alertar quando um determinado produto entra em promoção no site.

4. Sincronização de bases de dados: O uso de detecção de mudanças pode ser usado em bancos de dados que permitam atualização de visões XML. Através do *delta* pode ser gerado o conjunto de atualizações na base de dados (BRAGANHOLO, 2004).

A detecção de *delta* em dados estruturados possui alguns aspectos importantes, que serão vistos no decorrer desta seção.

De uma maneira geral as técnicas de detecção de *delta* atuam conforme o Algoritmo 1. Obviamente, cada sistema possui técnicas específicas de otimização, qualidade de resultados e outras características próprias.

2.2.1 Estrutura dos Documentos Analisados

As propostas de sistemas para detecção de diferenças entre dados estruturados, em sua maioria, utilizam como modelo de dados documentos XML, como pode-se ver nos algoritmos XyDiff (COBENA, 2002; MARIAN, 2001), X-Diff (WANG, 2003), Xandy (LEONARDI, 2006) e outros. No entanto, nem todas as abordagens trabalham necessariamente com XML. O MHDIFF (CHAWATHE, 1997; GARCIA-MOLINA, 1997), uma das primeiras propostas na área, não trabalha com documentos XML especificamente, mas com uma estrutura de árvores própria. Porém, seu estudo torna-se importante porque suas técnicas são abrangentes e servem como base para muitos trabalhos posteriores na área.

Nas técnicas em que os documentos são estruturados em árvores não-ordenadas, como X-Diff e Xandy, não é levada em conta qualquer relação de ordem entre os nodos-irmãos. Essas técnicas são principalmente utilizadas quando o documento representa ou foi gerado a partir de bases de dados relacionais.

Outra característica importante na avaliação é a forma pela qual o sistema de detecção de *delta* armazena os documentos XML enquanto está lidando com eles. A maioria dos sistemas armazena os documentos em memória. Porém, recentemente algumas propostas como Xandy (LEONARDI, 2006), Oxone (LEONARDI, 2006) e Helios (LEONARDI, 2005), começaram a lidar com documentos XML armazenados em SGBDs. Tal técnica torna o sistema escalável, uma vez que os documentos podem ser maiores que a memória disponível. Uma desvantagem é que as operações de comparação utilizam operações do SGBD, o que podem acarretar uma diminuição de performance.

2.2.2 Modelo das Operações

Os sistemas de detecção de *delta* buscam utilizar um modelo de operações semanticamente correto, ou seja, com operações que representem de maneira fiel as modificações efetuadas pelo usuário. Desse modo, as operações são expressas tanto para agirem sobre as informações quanto sobre a estrutura da árvore. A seguir será utilizado como base o modelo de operações do MHDIFF (CHAWATHE, 1997; GARCIA-MOLINA, 1997), por abranger os outros modelos propostos.

Inserção: Criação de um novo nodo n na árvore. A inserção é definida por $INS(n, v, p, C)$, sendo v o rótulo do nodo n a ser inserido, p o nodo que será pai de n e C o conjunto de filhos de p que agora serão filhos de n.

Exclusão: É a operação inversa da inserção. É definida por $DEL(n)$, sendo n o nodo a ser excluído. Os filhos de n passam a ser filhos do pai de n. A raiz não pode ser excluída.

Atualização: Operação que troca o rótulo de um nodo. É definida por $UPD(n, v)$, colocando o rótulo v no nodo n.

Realocação: Consiste em mover uma sub-árvore para uma outra posição, de modo que a raiz dessa passa a ser filha de um outro nodo. Ela pode ser definida por $MOV(n, p)$. Sendo n o nodo-raiz da sub-árvore e p o nodo que será pai de n.

Cópia: Cria uma cópia de uma sub-árvore para uma outra posição. $CPY(n, p)$ cria uma cópia da sub-árvore de raiz n e coloca o nodo-raiz como filho de p.

Cola: É a operação inversa à cópia. Junta duas sub-árvores isomórficas[1] em uma só. $GLU(n1, n2)$ junta duas sub-árvores com raízes $n1$ e $n2$. A sub-árvore $n1$ desaparece.

As operações de cópia e cola não são usadas na maioria das aplicações, sendo substituídas por uma seqüência de inserções e exclusões. A operação de realocação (*move*) é mais aplicada em técnicas onde árvores ordenadas são utilizadas, como no XyDiff. Abordagens como X-Diff (WANG, 2003) e Xandy (LEONARDI, 2006) não utilizam operações de realocação.

2.2.3 Correspondência entre os Nodos

Antes de descobrir as diferenças, é necessário descobrir que nodo em uma versão do documento corresponde a um nodo na outra versão do documento. A primeira etapa para detecção do *delta*, denominada correspondência entre os nodos, consiste em realizar esta tarefa.

Algumas técnicas de correspondência são baseadas na identificação de nodo, ou seja, cada elemento XML possui uma identificação única, que não pode ser alterada. Apesar deste tipo de estrutura facilitar muito a correspondência entre os nodos, seu campo de aplicação é muito limitado, uma vez que nem sempre se tem controle sobre os documentos XML (por exemplo, numa aplicação que recolhe documentos na Web). Além disso, as aplicações que alteram o XML devem gerenciar a identificação dos nodos, o que excluiria a maioria das aplicações que lidam com XML. Dessa forma, serão analisados apenas trabalhos que não utilizam identificação única de nodos para correspondência.

A seguir, com o objetivo de exemplificar a técnica de correspondência de nodos, é apresentada uma descrição resumida do algoritmo de correspondência usado no MHDIFF.

Correspondência dos Nodos no MHDIFF

O MHDIFF utiliza uma técnica baseada em **grafos de indução** (figura 2.8) e modelos de custos. A primeira tarefa consiste em enumerar os nodos da árvore original e da árvore alterada e ligar estes nodos num grafo (o grafo de indução). O grafo de indução é um grafo bipartido[2] em que os nodos da árvore original ligam-se com os nodos da árvore alterada, representando assim sua correspondência. As figuras 2.8 e 2.9, utilizadas como exemplo, possuem a mesma forma de enumeração. Os nodos na parte de cima da figura (1 a 14) representam a árvore original e seguem a mesma numeração apresentada na Figura 1.3 (visão original). Os nodos na parte de baixo da figura representam a árvore alterada e seguem a numeração da Figura 1.5 (visão atualizada). A fim de que não haja dúvida na identificação dos nodos, uma das técnicas do MHDIFF é acrescentar um valor arbitrário a cada número de identificação da árvore modificada. Neste caso será utilizado um valor igual a 14, logo os números apresentados são de 15 a 32.

A princípio todos os nodos estão interligados, conforme apresentado na Figura 2.8. Os nodos especiais $\oplus$ e $\ominus$ auxiliam na detecção de inserções e exclusões.

A segunda tarefa é filtrar o grafo, eliminando arestas indesejadas. Para isso é utilizado um modelo de custo das operações do *Delta* (CHAWATHE, 1996, 1997). Para exemplificar de forma simples, considere o arco $a1 = [10, 32]$. As operações que envolvem

[1]Duas árvores são isomórficas se estas possuem os mesmos nodos numa mesma estrutura (com os mesmos pais), mesmo que a ordem dos nodos seja diferente.

[2]Grafo bipartido é um grafo que possui dois conjuntos de nodos, de modo que cada nodo só está conectado a nodos do outro conjunto.

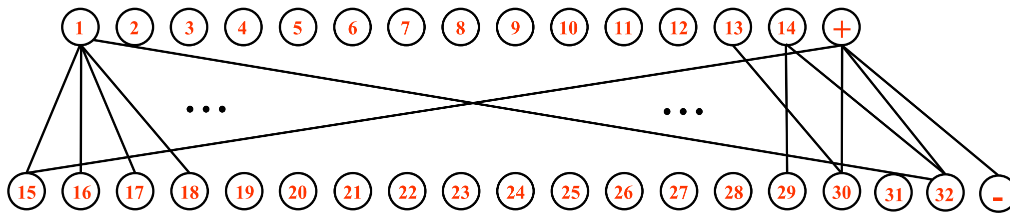


Figura 2.8: Grafo de indução representando a comparação entre a visão original e a visão atualizada.

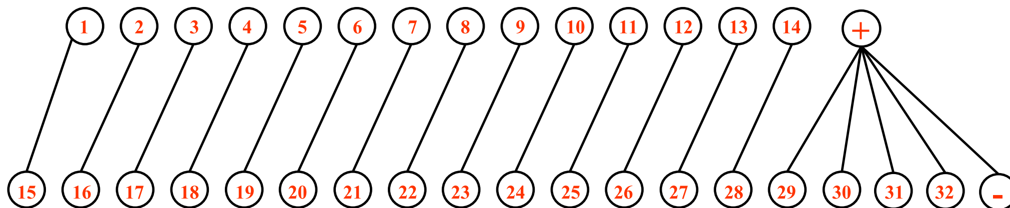


Figura 2.9: Grafo de indução filtrado, representando a comparação entre a visão original e a visão atualizada

a transformação do nodo 10 no nodo 32 (veja Figuras 1.3 e 1.5) envolvem operações de atualização e de realocação. Suponha que o custo para estas operações seja de 5 unidades. Outra alternativa para estes nodos seriam os arcos $a2 = [10, \ominus]$ e $a3 = [32, \oplus]$, cada uma com um custo de 1 unidade. Dessa forma tem-se que:

$[custo([10, 28]) > custo([10, \ominus]) + custo([32, \oplus])]$

Logo, o arco $a1$ deve ser filtrado, já que existe pelo menos uma combinação de arcos com um custo menor para estes dois nodos.

Na implementação da filtragem dos nodos, não seria viável comparar todas as combinações possíveis, para todo o tipo de arco. Por causa disso, o MHDIFF utiliza heurísticas para diminuir a complexidade computacional do problema (CHAWATHE, 1997; GARCIA-MOLINA, 1997). A Figura 2.9 apresenta o grafo de indução filtrado.

2.2.4 Linguagem de Representação do *Delta*

Um importante aspecto nos algoritmos de detecção é a forma pela qual o *delta* é representado. Cada algoritmo segue sua forma particular, voltado para o tipo de aplicação da saída. Algumas aplicações geram saídas voltadas para uma análise direta pelo usuário, enquanto outras são projetadas para serem analisadas por outras aplicações.

Nesta seção os algoritmos são classificados com base na forma como identificam os nodos alterados e como apresentam as modificações.

2.2.4.1 Identificação dos nodos

Antes de qualquer manipulação dos nodos, é necessário que estes sejam identificados. Neste trabalho são apresentadas duas técnicas de identificação de nodos.

Identificação de Nodos por XPath

XPath (W3C, 2004; CLARK, 1999) é o padrão atual para recuperação de nodos em um documento XML. A utilização do XPath pode trazer vantagens para a análise dos dados, uma vez que diversas linguagens de programação e outros softwares possuem interfaces e bibliotecas para manipulação do XPath. Por ser a base de localização de nodos na linguagem de consulta XQuery, a identificação por XPath facilita ferramentas que geram

a saída baseadas nesta linguagem, além de facilitar a análise por bases de dados XML (FERNANDEZ, 2004).

A utilização do XPath possui algumas desvantagens, conforme apresentado em (COBENA, 2003).

- Uma mesma expressão XPath nem sempre pode representar o mesmo nodo em diferentes versões de um documento.
- Um caminho XPath pode ser ambíguo, ou seja, pode não apontar para um único nodo no documento.
- Mais de um caminho XPath pode apontar para um mesmo nodo no documento.

Para identificar os nodos dos documentos, o X-Diff gera uma estrutura de dados chamada XHash (WANG, 2003). Esta estrutura é basicamente formada pelo caminho do nodo escrito em uma notação semelhante ao XPath (veja Seção 2.2.5). Em seguida, é calculado com base neste caminho um número hash que é usado como assinatura do nodo.

Como será visto posteriormente, o modelo de operações adotado pelo X-Diff e sua estrutura de árvore não-ordenada, permite que o XHash seja adotado sem os problemas acima listados. Além disso, a identificação por caminho pode facilitar a expressão final do *delta* para leitura feita diretamente pelo usuário.

Identificação de Nodos por XID

A identificação dos nodos dos documentos XML é gerada no XyDiff utilizando-se uma estrutura de dados denominada XID (Xyleme IDs). Os XIDs são identificações persistentes que são dadas a todos os nodos do documento.

De forma simplificada, o primeiro passo para a construção do XID é, a partir da primeira versão de um documento, associar um número para cada nodo encontrado. Por exemplo, no documento apresentado na Figura 2.10(a), os XIDs estão representados pelos números ao lado dos nodos. Estes foram gerados usando-se a ordem pós-fixada à esquerda para percorrer a árvore. Quando o XyDiff recebe uma nova versão do documento, este faz um mapeamento dos nodos com a versão anterior. Nodos já mapeados continuam com a mesma identificação adotada na versão anterior. Nodos não mapeados recebem novos XIDs.

A Figura 2.10(b) representa uma nova versão do documento. Somente os nodos não mapeados na versão 2.10(a) recebem um novo XID.

Nesta técnica, pode-se observar que um XID nunca é ambíguo e pode ser referenciado de forma única para todas as versões de um documento. Dessa forma, um nodo é referenciado por apenas um XID e um XID aponta para apenas um nodo. O projeto Xyleme, do qual saíram os XIDs, tem por um dos objetivos a indexação de documentos XML. Com a identificação persistente, esta tarefa se torna mais eficaz. (XYLEME, 2001; MARIAN, 2001).

2.2.4.2 Modelos de Representação das Operações

A forma como as operações são representadas é um importante aspecto a ser considerado na utilização de uma ferramenta de detecção de diferenças. Por exemplo, numa aplicação que armazena diversas versões de um documento (CHIEN, 2001; MARIAN,

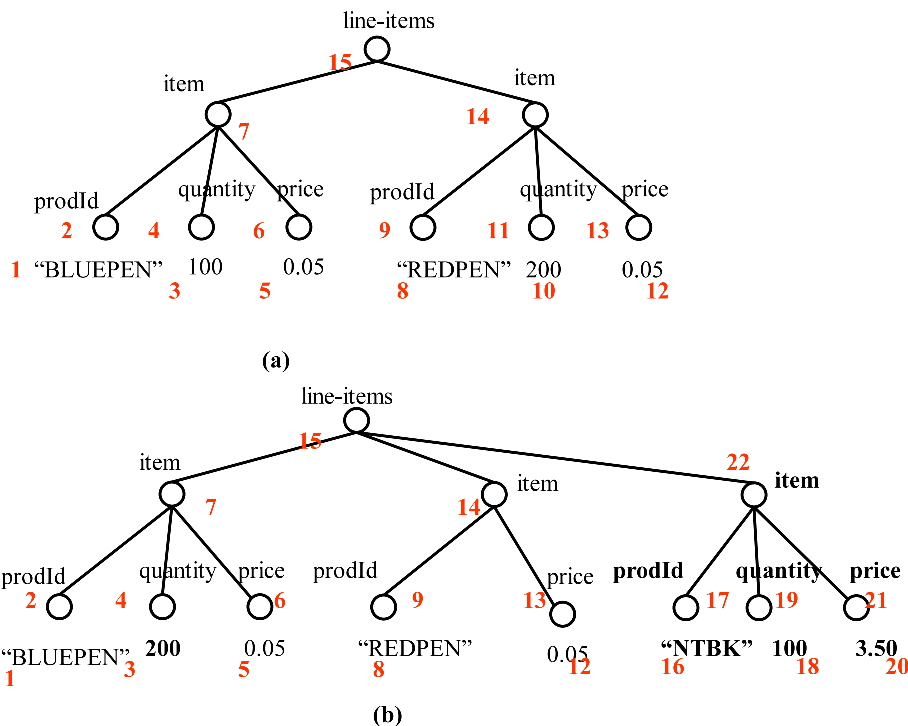


Figura 2.10: Exemplo de duas versões de um documento XML. O número associado ao nodo é seu XID. Note que na árvore (b) os nodos 10 e 11 foram excluídos e os nodos de 16 a 22 foram inseridos.

2001), o algoritmo de diferenças pode estar sendo aplicado para reconstruir versões não-armazenadas do documento. Numa aplicação de indexação de documentos, talvez o objetivo seja identificar um nodo ao longo das versões. Outras aplicações podem usar a detecção para monitorar sites na Web, variações de preços, entre outros.

Em cada um dos casos a abordagem para se representar as diferenças pode ser mais ou menos adequada. Neste trabalho serão apresentadas algumas formas de representação de *deltas*.

Representação do *Delta* por Lista de Operações

Uma abordagem para representar as diferenças é entregar como saída uma lista de operações. Há diversas vantagens em se gerar uma saída nesta forma. Para o caso de uma base de dados versionada, por exemplo, um dos objetivos não é armazenar todas as versões dos documentos, mas apenas algumas versões e o *delta* entre elas, de forma que seja possível construir uma versão não-armazenada a partir de uma versão e suas diferenças. Uma das vantagens desta técnica é a economia de espaço de armazenamento das diferenças e a facilidade para construir uma versão através de uma versão anterior e seu *delta*. A saída expressa por uma lista de operações torna também mais fácil o mapeamento para um outro formato, como comandos SQL ou XQuery.

No MHDIFF (CHAWATHE, 1997), o *delta* é representado como uma lista de operações **ordenadas**. As operações são ordenadas a fim de prevenir operações que, apesar de corretas, não tenham relevância na geração da versão final do documento (por exemplo, mover um nodo para uma sub-árvore que será excluída em uma operação posterior). As regras de ordenação das operações podem ser vistas em (CHAWATHE, 1996, 1997).

1. upd(3,200)
2. del(10)
3. del(11)
4. ins(22,"item",15,{ })
5. ins(17,"prodId",22,{ })
6. ins(19,"quantity",22,{ })
7. ins(21,"price",22,{ })
8. ins(16,"NTBK",17,{ })
9. ins(18,"100",17,{ })
10. ins(20,"3.50",17,{ })

Figura 2.11: Exemplo de *Delta* no MHDIFF.

O *delta* produzido pela diferença entre a versão original (Figura 2.10 (a)) e a versão atualizada (Figura 2.10 (b)) poderia ser expresso no MHDIFF com a expressão na Figura 2.11. Estas visões XML, presentes nas Figuras 2.10 (a) e (b), serão usadas como exemplo para aproveitar o identificador de nodo (números ao lado dos nodos) que se aproxima do utilizado no modelo original do MHDIFF. Note que as operações 2 e 5 a 10 são necessárias, pois o modelo não considera exclusão nem inclusão de sub-árvores como operações primárias (CHAWATHE, 1997).

O XyDIFF também utiliza a abordagem de lista de operações, gerando sua saída como um documento XML (XyDelta) contendo as operações. Assim como no MHDIFF, o XyDelta pode ser revertido, ou seja, tendo-se um *delta* Δ que transforma o documento D_1 para o documento D_2, pode-se encontrar um *delta* Δ^{-1} que gera o documento D_1 a partir do documento D_2. Outras características sobre o XyDelta podem ser vistas em (COBENA, 2003).

As operações no XyDelta também são ordenadas segundo a função *COMPARE* apresentada em (COBENA, 2003). No XyDiff, o XyDelta representa o conjunto de operações, ou seja, não possui qualquer ordenação. O XyScript é uma lista dessas operações com uma ordenação válida. O principal objetivo é manter o posicionamento relativo dos nodos quando estes forem objeto de exclusão ou inserção. De uma forma geral, são adotadas as seguintes regras (COBENA, 2003):

1. A operação de exclusão deve ser anterior às outras;

2. Operações de exclusão de nodos-irmãos devem ser ordenadas na ordem reversa em relação à posição dos nodos a serem excluídos;

3. Operações de inserção de nodos-irmãos devem ser ordenadas na mesma ordem dos nodos a serem inseridos;

O documento apresentado na Figura 2.12 apresenta o *delta* gerado pelo XyDiff, representando a diferença entre a versão original (Figura 2.10 (a)) e a versão atualizada (Figura 2.10 (a)).

Representação do *Delta* por Operações Anexadas ao Documento

```
<Delta>
<update XID="3">
  <old-value>
    100
  </old-value>
  <new-value>
    200
  </new-value>
</update>
<delete XID="11" XID-map="(10-11)" parentXID="14" pos="1">
  <quantity>
    200
  </quantity>
</delete>
<insert XID="22" XID-map="(16-22)" parentXID="15" pos="1">
  <item>
      <prodId>NTBK</prodId>
      <quantity>100</quantity>
      <price>3.50</price>
  </item>
</insert>
</Delta>
```

Figura 2.12: Resultado do *delta* entre as versões a e b do documento 2.10, gerado pelo Algoritmo XyDiff

A segunda abordagem utilizada para representar as operações é apresentar uma versão do documento (final ou inicial) e anexar a este documento elementos XML que expressem as operações realizadas sobre ele ou que gerem a outra versão.

Esta abordagem geralmente é adotada quando a saída será analisada diretamente pelo usuário, uma vez que a saída se apresenta de forma mais amigável, facilitando a percepção por parte do usuário das diferenças entre as versões. Por exemplo, neste tipo de formato de saída o usuário não precisa buscar índices como o XID, no XyDiff, uma vez que os nodos seguem a mesma estrutura do documento original. Note que pelo próprio formato de apresentação não existe, nesse caso, a ordenação das operações.

Nesta abordagem existem algumas desvantagens quando se quer fazer análises mais complexas. A primeira é com relação ao tamanho da representação do *delta*, uma vez que cada um carrega o documento anexado. Além disso, as operações dessa forma tornam difíceis a inversão de um *delta* (gerar a versão original a partir da alterada) ou a sua agregação (compor um *delta* a partir de duas ou mais saídas). Operações que analisam para uma consulta diversos *deltas* também podem se tornar mais difíceis.

Neste trabalho serão apresentados dois modelos que adotam operações anexadas ao documento: o X-Diff e o DeltaXML.

A representação do *delta* gerado pelo DeltaXML (FONTAINE, 2001) é feita em um documento XML com a estrutura semelhante ao do documento original. As operações são representadas como atributos nos nodos do documento, usando o *namespace deltaxml*. Para representar alterações nos nodos-texto, são usados os elementos *<deltaxm:oldtext>* e *<deltaxm:newtext>*. A Figura 2.13 apresenta o *delta* gerado através da versão original 2.10 (a) e a versão atualizada 2.10 (b).

A seguir, será analisado o algoritmo de detecção de *delta* X-Diff. Esta técnica possui características que a tornam mais adequada a este trabalho, portanto, é necessário que seja feita uma análise mais aprofundada.

2.2.5 Visão Geral do X-Diff

O X-Diff (WANG, 2003; DEWITT, 2003) é um algoritmo de detecção de diferenças cuja proposta é utilizar modelos de árvores não-ordenadas. O X-Diff possui as seguintes

```
<line-items xmlns:deltaxml="http://www.deltaxml.com/ns/well-formed-delta-v1" deltaxml:delta="WFmodify">
      <item deltaxml:delta="WFmodify">
            <prodId deltaxml:delta="unchanged">BLUEPEN</prodId>
            <quantity deltaxml:delta="WFmodify">
                  <deltaxml:PCDATAmodify>
                        <deltaxml:PCDATAold>100</deltaxml:PCDATAold>
                        <deltaxml:PCDATAnew>200</deltaxml:PCDATAnew>
                  </deltaxml:PCDATAmodify>
            </quantity>
            <price deltaxml:delta="unchanged">0.05</price>
      </item>
      <item deltaxml:delta="WFmodify">
            <prodId deltaxml:delta="unchanged">REDPEN</prodId>
            <quantity deltaxml:delta="delete">200</quantity>
            <price deltaxml:delta="unchanged">0.05</price>
      </item>
      <item deltaxml:delta="add">
            <prodId>NTBK</prodId>
            <quantity>100</quantity>
            <price>3.50</price>
      </item>
</line-items>
```

Figura 2.13: Resultado do *delta* gerado pelo DELTAXML a partir da versão da Figura 2.10 (a) e a versão atualizada da Figura 2.10 (b).

características, que o tornam adequado a esta proposta:

- Modelo de árvores não-ordenadas: O X-Diff não leva em conta a mudança de ordem entre elementos-irmãos. Como nesta abordagem o documento XML é gerado a partir de uma base de dados relacional, a ordem de construção das coleções pode variar, sem porém haver alguma diferença semanticamente relevante. Além disso, o próprio PATAXÓ também não leva em conta a ordem dos elementos XML. Dessa forma, ignorar essas diferenças é desejável.

- Modelo de operações simplificado: O X-Diff não possui no seu modelo operações como realocação (*move*), cópia ou cola. Apesar de ser uma desvantagem em algumas aplicações, por diminuir a semântica das operações [3], sua restrição fica de acordo com o modelo de operações PATAXÓ, que utiliza apenas inserções, atualizações e exclusões para manipulação dos documentos XML. Além disso, um número menor de operações permite algoritmos menos complexos para a geração do *Delta*.

- Busca pelo Melhor *Delta*: Apesar de existir diversas combinações de operações que podem gerar um *delta* válido, ou seja, que consegue gerar a partir do documento original o documento final, muitas destas combinações não são semanticamente corretas, ou seja, não expressam quais foram as modificações feitas pelo usuário.

  No caso de uma aplicação de detecção de diferenças em que o objetivo seja apenas gerar uma versão do documento a partir de outra, os problemas causados por não se escolher o melhor *delta* são pequenos, podendo ser apenas a criação de operações mais custosas computacionalmente. Mas na aplicação proposta, o interesse principal está nas operações do *delta* e na sua conversão para operações SQL. Dessa maneira, a escolha de um *delta*, válido no contexto do documento XML, poderá

[3]Como um exemplo, um cliente poderia mover um elemento $item$ de um */order[@numOrder="123"/line-items* para */order[@numOrder="124"/line-items* querendo expressar que este item deve pertencer ao segundo pedido.

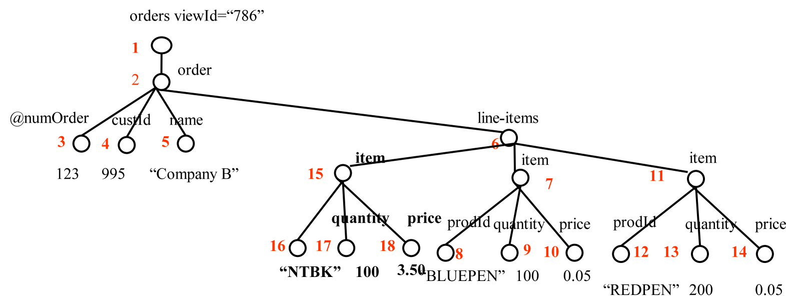


Figura 2.14: Exemplo de visão atualizada com um novo item foi inserido no início da lista.

ser impossível de ser mapeado para atualizações SQL ou gerar efeitos colaterais indesejados no contexto da base de dados relacional. Por exemplo, a partir da visão XML da Figura 1.3, o usuário insere um novo item, conforme apresentado na Figura 2.14. Note que o usuário fez tal inserção no início da coleção. A Figura 2.15 apresenta o *delta* gerado pelo DeltaXML[4] e a Figura 2.16 o *delta* gerado pelo X-Diff. O X-Diff detectou corretamente a inserção enquanto o DeltaXML trocou o atributo *prodId* (chave primária na base relacional) de todos os produtos e inseriu o último elemento da coleção. Por buscar *deltas* semanticamente mais relevantes o X-Diff se mostra mais adequado.

Operações do X-Diff

Existem três operações básicas que o X-Diff reconhece:

1. $Insert(x(nome, valor), y)$: insere o nodo x, com o nome e valor como *nome* e *valor* respectivamente, como filho do nodo y;
2. $Delete(x)$: exclui o nodo x;
3. $Update\ (x, valor)$: coloca *valor* como novo conteúdo do nodo x.

As operações de inserção e exclusão podem atuar sobre sub-árvores e a operação de atualização só pode atuar alterando o conteúdo de nodos folha.

As operações de cópia, cola e realocação não são utilizados no X-Diff. Como o X-Diff não leva em conta a ordem dos elementos, realocações de elementos dentro de um mesmo conjunto não são detectadas como alterações. Os demais casos são detectados como inserções e exclusões de nodos ou sub-árvores.

Correspondência dos Nodos no X-Diff

O X-Diff utiliza as árvores DOM (HEGARET, 2006) para representar os documentos XML. Os tipos de nodos a serem utilizados são:

- *Elements* - elementos XML. Por exemplo, */items*;

[4]Versões mais recentes do DeltaXML parecem contornar este problema.

```
<orders viewId="786">
  <order numOrder="123">
    <custId> ... </custId>
    <name> ... </name>
    <line-items>
      <item>
        <prodId> BLUEPENNTBK </prodId>
        <quantity> ... </quantity>
        <price> 0.053.50 </price>
      </item>
      <item>
        <prodId> REDPENBLUEPEN </prodId>
        <quantity> ... </quantity>
        <price> ... </price>
      </item>
      <item>
        <prodId> REDPEN </prodId>
        <quantity> 300 </quantity>
        <price> 0.05 </price>
      </item>
    </line-items>
  </order>
</orders>
```

Figura 2.15: Resultado do *delta* gerado pelo deltaxml a partir da visão original e da visão da Figura 2.14 (DELTAXML, 2007).

```
<orders viewId="786">
      <order numOrder="123">
            <custId>995</custId>
            <name>Company B</name>
            <line-items>
                  <item>
                        <prodId>BLUEPEN</prodId>
                        <quantity>200<?UPDATE FROM "100"?></quantity>
                        <price>0.05</price>
                  </item>
                  <item>
                        <prodId>REDPEN</prodId>
                        <quantity>300<?UPDATE FROM "200"?></quantity>
                        <price>0.05</price>
                  </item>
                        <item><?INSERT item?>
                        <prodId>NTBK</prodId>
                        <quantity>100</quantity>
                        <price>3.50</price>
                  </item>
            </line-items>
      </order>
</orders>
```

Figura 2.16: Resultado do *delta* gerado pelo X-Diff a partir da visão original e da visão da Figura 2.14.

- *Text* - textos e os valores dos elementos;

Tabela 2.2: Exemplos de assinatura de nodos no X-Diff

Visão Original	Assinatura X-Diff
Nodo de valor “995”	$/orders/order/custId/\$TEXT\$$
Nodo 4 (custId)	$/orders/order/custId/\$ELEMENT\$$
Nodo 3 (@numOrder)	$/orders/order/numOrder/\$ATTRIBUTE\$$

- *Atribute* - Atributos dos elementos.

O primeiro passo consiste em gerar a assinatura dos nodos, que logo em seguida é usada para gerar correspondência entre estes. A correspondência dos nodos segue os passos abaixo:

1. Leitura (*Parse*) dos documentos e geração das assinaturas: O X-Diff faz a leitura dos documentos, localizando cada nodo e gerando uma identificação única para cada um, chamada de **assinatura do nodo** (KHAN, 2002). A assinatura é formada pelo nome do nodo, concatenado com o nome de todos os ancestrais deste nodo desde a raiz do documento. A Tabela 2.2 apresenta alguns exemplos de assinaturas de nodos. O documento usado como base é a visão original (Figura 1.3).

   Por exemplo, as assinaturas do nodo número 4 (custId) da visão original (Figura 1.3), do nodo de valor “995” e o nodo 3 são apresentados na Figura 2.2.

   Para otimizar o processo de correspondência dos nodos, o X-Diff gera o XHash (WANG, 2003) de cada uma das assinaturas.

2. Correspondência dos Nodos:

   A correspondência dos nodos é feita com base nas seguintes afirmativas: a) dois nodos casam se e somente se seus nodos tiverem a mesma assinatura; b) se dois nodos casam, então seu pais também casam (não existe operação de realocação de nodos no X-Diff, como dito anteriormente); c) o casamento de nodos é sempre um para um, ou seja, um nodo em um documento não pode corresponder a dois nodos no outro (no X-Diff não existe operações cópia ou cola de sub-árvores, como dito anteriormente); d) uma vez que o tipo do nodo é uma informação presente na assinatura, dois nodos só casam se forem do mesmo tipo.

   Com base nas afirmativas acima a correspondência ocorre somente se a assinatura dos nodos for a mesma, os nodos pais já estiverem casados e existir correspondência entre os nodos-raiz.

   A correspondência é feita a partir da raiz do documento e indo para os nodos filhos recursivamente. Para cada nodo filho de A, busca-se um nodo de mesma assinatura em A’ (A e A’ são dois nodos que já casaram). Se for encontrado, então os nodos casam; caso não seja encontrada correspondência, o algoritmo buscará a operação de menor custo. O algoritmo para a correspondência pode ser expresso como apresentado em (DEWITT, 2003).

3. Geração do *Edit Script* de Custo-Mínimo:

   Nesta fase o X-Diff gera o *delta* entre os dois documentos. Uma análise mais aprofundada sobre os custos de operação e a geração do *delta* de menor custo pode ser vista em (WANG, 2003; DEWITT, 2003). Pode-se dizer que a geração do *delta* segue três regras:

```
<orders viewId="786">
      <order numOrder="123">
            <custId>995</custId>
            <name>Company B</name>
            <line-items>
                  <item>
                        <prodId>BLUEPEN</prodId>
                        <quantity>200<?UPDATE FROM "100"?></quantity>
                        <price>0.05</price>
                  </item>
                  <item>
                        <prodId>REDPEN</prodId>
                        <quantity>300<?UPDATE FROM "200"?></quantity>
                        <price>0.05</price>
                  </item>
                  <item><?INSERT item?>
                        <prodId>NTBK</prodId>
                        <quantity>100</quantity>
                        <price>3.50</price>
                  </item>
            </line-items>
      </order>
</orders>
```

Figura 2.17: Resultado do *delta* entre as visões original e atualizada, gerado pelo Algoritmo X-Diff.

(i) se a assinatura dos nodos for igual, então não há operação;

(ii) se a assinatura for diferente (não houver casamento), então adicionar operações de *Delete* e *Insert*;

(iii) se a assinatura dos nodos folha for diferente, mas a diferença está apenas nos valores dos nodos, então adicionar operação de *Update*.

O algoritmo para a geração do *Edit Script* pode ser expresso como apresentado em (DEWITT, 2003).

Representação do *Delta* no X-Diff

O *delta* no X-Diff não é representado na forma de uma série de comandos seqüenciais, como no MHDIFF ou no XyDiff. Ele representa as operações no próprio documento, inserindo elementos que representam operações de modificação. O documento base para a representação é o documento atualizado.

A Figura 2.17 apresenta o *delta* gerado entre a visão original (Figura 1.3 e a visão atualizada (Figura 1.5).

Limitações do X-Diff

O X-diff possui algumas limitações na detecção de diferenças, dentre as quais são apresentadas a seguir as mais importantes.

- Alteração no nome do elemento: O X-Diff não permite a alteração do nome de um elemento, seja ele um nodo ou um atributo. Caso um elemento tenha seu nome trocado será detectado com uma exclusão seguida de uma inclusão, mesmo se a sua sub-árvore tiver se mantido idêntica à original. Nesse caso, pode-se dizer que não foi escolhido o menor *delta*, uma vez que foi feito uma série de exclusões e inclusões que não existiram do ponto de vista semântico.

Um exemplo ocorre quando um documento XML possui nodos opcionais. Por exemplo, considere o trecho de um documento, apresentado abaixo:

```
<pessoa>
    ....
   <sexo>
     <masculino/>
   </sexo>
    ....
</pessoa>
```

Para esse tipo representação, a modificação do sexo para *<feminino/>* não será detectada como um *update* mas como uma exclusão seguida de inserção.

- Unificação de Elementos: a Figura 2.18 apresenta um exemplo de elementos de mesmo nome que são unidos. Os nodos 3 e 4 ($/u/w$) possuem a mesma assinatura e podem casar com o nodo 3' ($/u/w$). Como o casamento é feito a partir da raiz e o casamento deve ser 1 para 1, um deles é escolhido, no caso (3,3'). Dessa forma, o casamento (8,6') não poderá ser efetivado, uma vez que seus nodos pai (4 e 3') não foram casados. Em alguns casos, esta detecção pode trazer resultados que não representam as alterações feitas pelo usuário sobre o documento XML (DEWITT, 2003).

- Escalabilidade: Apesar de ter uma ótima performance em documentos pequenos, o X-Diff tem uma queda de performance quando detecta diferenças em documentos maiores. Além disso, existe uma queda no tempo de detecção quando o número de nodos alterados passa de 10% do documento, conforme estudos apresentados em (LEONARDI, 2006; DEWITT, 2003). Como o X-Diff armazena todo o documento na memória, este está limitado ao tamanho da memória da máquina. Esta pode ser uma limitação importante no caso de documentos muito grandes.

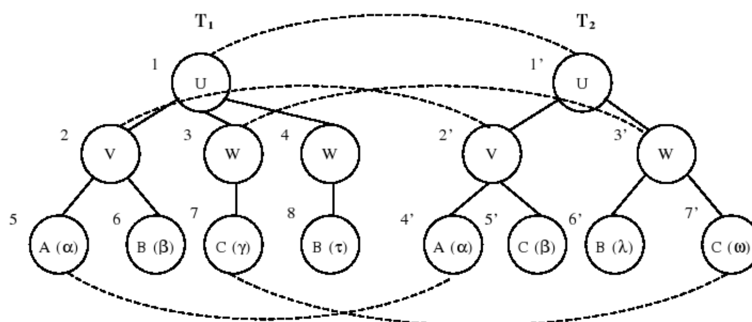


Figura 2.18: Exemplo de casamento de nodos no X-Diff (DEWITT, 2003)

2.2.6 Quadro Comparativo entre Algoritmos de Detecção de *Delta*

A tabela 2.3 apresenta um quadro comparativo entre diversas técnicas de detecção de delta disponíveis na literatura. Apesar de existirem muitas técnicas, apenas algumas são apresentadas, por possuírem características importantes e por se diferenciarem mais entre si. Para tal quadro foram consultados, além dos trabalhos contidos nas referências dentro da tabela, alguns trabalhos voltados para a comparação entre técnicas (COBENA, 2002; PETERS, 2005; GREGORY COBENATALEL ABDESSALEM, 2002).

Nem todos os aspectos foram estudados em todas as técnicas, mas os aspectos mais relevantes nesse trabalho e as técnicas que poderiam ser aplicadas neste contexto.

2.3 Controle de Concorrência, Conciliação e Tratamento de Conflito

O controle de concorrência em bancos de dados é um tema já bastante conhecido e tratado na literatura. Transações ACID, com utilização de bloqueios de tuplas, são utilizadas na maioria dos SGBDs comerciais. O objetivo do controle de concorrência é manter a consistência dos dados e a ordenação correta das operações. Se uma transação for executada isoladamente na base de dados, ela fará o banco partir de um estado consistênte para outro estado consistente. O controle de concorrência tem a função de, mesmo com várias transações ocorrendo no SGBD, garantir que cada uma produza o mesmo resultado que obteria caso fossem executadas uma após a outra isoladamente (ELMASRI, 2004; ULLMAN, 2002).

Os SGBDs podem adotar níveis de isolamento para suas transações. Níveis mais baixos de isolamento aumentam a possibilidade de acesso concorrente e a eficiência geral do sistema, porém, aumenta o risco de que transações acessem a base de dados em um estado intermediário inconsistente (POSTGRESQL, 2007). O padrão ANSI SQL define níveis de isolamento para controle de concorrência e a possibilidade de ocorrência de *phenomenas* em cada nível. *Phenomena* são efeitos indesejáveis ao usuário de um banco de dados. Os níveis de isolamento e os *phenomenas* relacionados podem ser vistos em (BERENSON, 1995).

2.3.1 Métodos de Controle de Concorrência

O principal método para o controle de concorrência é o bloqueio de tupla. Nesse método o SGBD garante que apenas uma transação pode acessar determinada tupla de cada vez (ULLMAN, 2002; ÖZSU, 1999).

Para transações muito grandes, formadas por centenas de operações, outras técnicas de bloqueios são tratadas na literatura. Conhecidas por transações longas, existem diversas técnicas e questões próprias desta área (ULLMAN, 2002). A maior dificuldade está em gerar um bloqueio durante um longo período, impedindo o acesso concorrente. A técnica de sagas (GARCIA-MOLINA, 1987) produz um particionamento da transação, como se cada uma fosse uma transação curta (saga). Cada saga possui uma função de compensação, que é usada caso o banco precise abortar a saga.

Em bases de dados distribuídas, outras técnicas de bloqueios são utilizadas. Transações com consolidação em duas fases (*two-phase commit*) e bloqueios distribuídos são utilizados em abordagens na área (ULLMAN, 2002).

Controle de Concorrência Baseado em Múltiplas Versões (MVCC)

SGBDs comerciais, como PostgreSQL, utilizam uma técnica baseada em múltiplas versões de uma tupla, de forma que uma escrita não bloqueia uma leitura e uma leitura não bloqueia uma escrita. Cada transação enxerga um instantâneo (*snapshot*) dos dados (uma versão da base de dados), sem levar em consideração o estado corrente dos dados subjacentes. Este modelo protege a transação contra enxergar dados inconsistentes, o que poderia ser causado por atualizações feitas por transações simultâneas na mesma tupla, fornecendo um isolamento da transação para cada sessão do banco de dados (CAREY, 1986; POSTGRESQL, 2007).

Transações Desconectadas

Nos últimos anos tem surgido um interesse cada vez maior nos dispositivos móveis, uma vez que aparelhos celulares e PDAs estão ganhando cada vez mais popularidade. Neste tipo de ambiente, devido às limitações de armazenamento, processamento e conexão, parte da base de dados é replicada para o dispositivo permitindo que o usuário faça atualizações localmente. Os dados podem ser distribuídos para vários dispositivos e receber atualizações concorrentes. Este tipo de base de dados é conhecido como bases de dados desconectadas. Suas transações podem ser chamadas de **transações desconectadas** (KLIEB, 1996).

Nesta dissertação as transações são desconectadas, ou seja, o usuário faz as atualizações na visão XML que está na sua máquina, desconectada da base de dados central. Tal ambiente torna inviável o uso de bloqueios tradicionais. Este ambiente também difere do aplicado às transações longas. Nelas, o sistema já conhece desde o início da transação quais são as operações envolvidas. A maior demanda de tempo decorre do grande número de operações envolvidas. Por outro lado, nas transações desconectadas a maior demanda de tempo é devido ao período em que o usuário está alterando sua visão XML. Da mesma forma as transações desconectadas não possuem técnicas voltadas para transações com grande número de operações, uma vez que este número pode estar limitado à capacidade dos dispositivos móveis.

Portanto, neste contexto outras técnicas para controle de concorrência devem ser consideradas.

2.3.2 Tratamento de Conflito e Conciliação

Enquanto as técnicas de controle de concorrência citadas no início desta seção são pessimistas, no contexto das transações desconectadas o modelo de controle de concorrência é otimista (SATYANARAYANAN, 1990; ÖZSU, 1999), ou seja, não existe uma prevenção para o acesso concorrente aos dados. O principal problema neste caso é manter a serialização das operações dos diversos usuários (PHATAK, 1999).

Neste modelo de transação o **conflito** ocorre quando mais de um cliente atualiza a mesma informação. Dependendo do contexto, essa informação pode ser uma tupla, uma estrutura de nodo em um documento XML ou uma parte de um objeto real. A situação de conflito está relacionada à semântica da aplicação que utiliza a base de dados. O tratamento do conflito também é tratado na literatura como **conciliação**. O objetivo da conciliação é efetuar o maior número de atualizações de todos os clientes e manter a consistência da base de dados (PHATAK, 1999; TERRY, 1995).

As estratégias abordadas na literatura para tratamento de conflitos são vinculadas ao tipo de aplicação. Neste trabalho serão apresentadas duas estratégias, baseadas respectivamente em operações de compensação e regras de dependência.

Tratamento baseado em Operações de Compensação

Em (PHATAK, 1999) é apresentada uma proposta para um sistema onde tuplas relacionais são distribuídas para os dispositivos móveis, ocorrendo as transações desconectadas. Nesta abordagem, são definidos dois níveis de granularidade para o tratamento dos conflitos.

O primeiro nível é a **granularidade voltada aos dados**. Nesse nível, cada atributo de uma tupla deve ser considerado um dado isolado para o tratamento do conflito. A estratégia passa por duas fases: geração da função de compensação, na qual o sistema

altera os dados para torná-los consistentes, e retorno ao usuário, que acontece caso ocorra alguma falha.

A grande dificuldade desta abordagem é a dependência da semântica da base de dados, ou seja, o sistema gerenciador de conflitos deverá conhecer o conteúdo da base de dados e armazenar estratégias para cada conflito que possa ocorrer. Tal estratégia restringe muito sua aplicação.

O segundo nível de granularidade é a granularidade por transação, cuja diferença principal é que toda a transação deve gerar conflito caso ocorram atualizações, ou seja, todos os dados dentro da transação são dependentes entre si.

Tratamento baseado em análise de dependência

Em (KLIEB, 1996) é apresentada uma proposta para tratamento baseada em regras de conflito. Neste caso, os conflitos são denominados *Clashes* e seguem regras de dependência. Por exemplo, duas alterações no mesmo dado geram *clashes* que devem ser informados ao usuário. Duas exclusões no mesmo dado ou inclusões e exclusões dentro de um conjunto, não geram *clashes*.

O uso das regras de dependência, porém, são voltados para bases de dados em arquivos e orientados a regitros. As regras de dependência são apresentadas em (KLIEB, 1996).

Em (SATYANARAYANAN, 1990) é apresentada uma proposta para tratamento de conflitos no sistema de arquivos distribuído. As cópias dos arquivos são distribuídas em várias máquinas e podem ser alteradas paralelamente. Os arquivos são conciliados através de técnicas baseadas em regras de dependência.

A utilização de regras de dependência são principalmente utilizadas para identificar os conflitos, mas não influencia nas operações dos usuários, como faz o modelo de operações de compensação. Por ser menos dependente da semântica da base de dados, este modelo tende a ser mais abrangente.

2.4 Considerações finais

Neste capítulo foram apresentados três aspectos que formam a base para este trabalho: o PATAXÓ; técnicas de detecção de *deltas* e técnicas de detecção e tratamento de conflitos. O PATAXÓ é utilizado nesta dissertação para a geração das visões XML e para mapear as atualizações para as visões relacionais. Dentre as técnicas para detecção de *deltas*, foi adotado o X-Diff, pelos motivos mencionados anteriormente. Este algoritmo atua no gerenciamento das transações, na descoberta das operações e auxilia na detecção dos conflitos. Por fim, as técnicas de detecção e tratamento de conflito são usadas como base para o desenvolvimento da abordagem utilizada neste trabalho.

A forma como tais técnicas interagem será explicada no capítulo seguinte, que apresenta a arquitetura proposta.

Tabela 2.3: Quadro comparativo entre algumas técnicas de detecção de diferenças

	MHDIFF	XyDiff	X-Diff	DeltaXML	Xandy	DiffXML	OXONE	HELIOS
Referência	(CHAWATHE, 1997)	(MARIAN, 2001)	(WANG, 2003)	(FONTAINE, 2001)	(LEONARDI, 2006)	(MOUAT, 2002)	(LEONARDI, 2006)	(LEONARDI, 2005)
Complexidade do Algoritmo	quadrático	linear	quadrático	linear		linear		
Uso da Memória	?	linear	quadrático	linear	uso de SGBD	linear	uso de SGBD	uso de SGBD
Modelo dos Dados	árvore genérica	árvore XML	árvore XML	árvore XML	árvore XML	árvore XML	árvores XML	árvore XML
Operações suportadas	Básicas,realocação cópia, cola	Básicas realocação	básicas	básicas	básicas	Básicas realocação		realocação
Método de Correspondência	Método de menor custo em grafo de Indução	Top-Down e Botton-Up Uso de assinaturas de nodo	Top-Down entre assinaturas de nodo	Comparação D-Band	Baseados nos algoritmos			
Edit Script de Custo Mínimo	SIM	Não	SIM	Não [a]				
Modelo da Saída	Script com instruções ordenadas	Script com as instruções em um XML específico	Instruções no Documento XML	Instruções no Documento XML				
Observações	Usado como base para algoritmos modernos			Produto Comercial				

[a] Testes nas últimas versões do software indicam o uso de operações de Custo Mínimo.

3 TRANSAÇÕES DESCONECTADAS

A utilização de bases de dados que não estão permanentemente conectadas, mas que devem em determinado período ser sincronizadas é conhecida na literatura como bases de dados desconectadas (KLIEB, 1996; PHATAK, 1999). Sua utilização é aplicada às bases em dispositivos móveis ou em contextos onde a comunicação permanente não é possível ou possui um custo elevado.

Neste trabalho, as transações ocorrem em bases de dados desconectadas, ou seja, o usuário altera uma parte da base de dados (a visão XML) armazenada em seu dispositivo, desconectado da base de dados central e de outras visões XML que estão sendo atualizadas por outros usuários.

O objetivo deste capítulo é apresentar a abordagem deste trabalho para lidar com transações desconectadas, utilizando como base o exemplo da Seção 1.4.2. A arquitetura proposta é apresentada na Figura 3.1. Os principais módulos do sistema são o *Gerenciador de Transações*, *Detector de Deltas* e o *Gerenciador de Atualizações*:

O *Gerenciador de Transações* é responsável por controlar as transações do sistema que estão abertas. Este recebe a consulta de definição da visão, a repassa para o PATAXÓ, recebe do PATAXÓ a visão XML resultante e sua DTD e as entrega para o cliente. Antes de enviar os dados para o cliente, este módulo executa as seguintes tarefas:

1. adiciona um `viewId` à raiz da visão XML (no exemplo da Figura 1.3, este valor aparece como 786)[1];

2. adiciona este mesmo atributo, com o mesmo valor, à raiz da consulta de definição da visão;

3. adiciona este mesmo atributo, com o mesmo valor como constante, à DTD;

4. armazena a visão XML, a DTD e a consulta de definição da visão num **Repositório Temporário**, uma vez que estes serão usados mais tarde quando a visão atualizada pelo usuário retornar ao sistema.

Quando a visão atualizada é retornada pelo cliente ao sistema, o Gerenciador de Transações verifica o `viewId`[2] e o utiliza para identificar a qual transação esta visão pertence. Ao identificar a transação, o Gerenciador de Transações extrai, do Repositório Temporário, a visão originalmente entregue ao cliente, a consulta de definição da visão e a DTD correspondentes. Em seguida é verificada a validade da visão entregue pelo cliente, em

[1] O valor que está sendo atribuído ao atributo `viewId` é um número seqüencial controlado pelo Gerenciador de Transações.

[2] Um requisito do sistema é que `viewId` não pode ser modificado durante a transação.

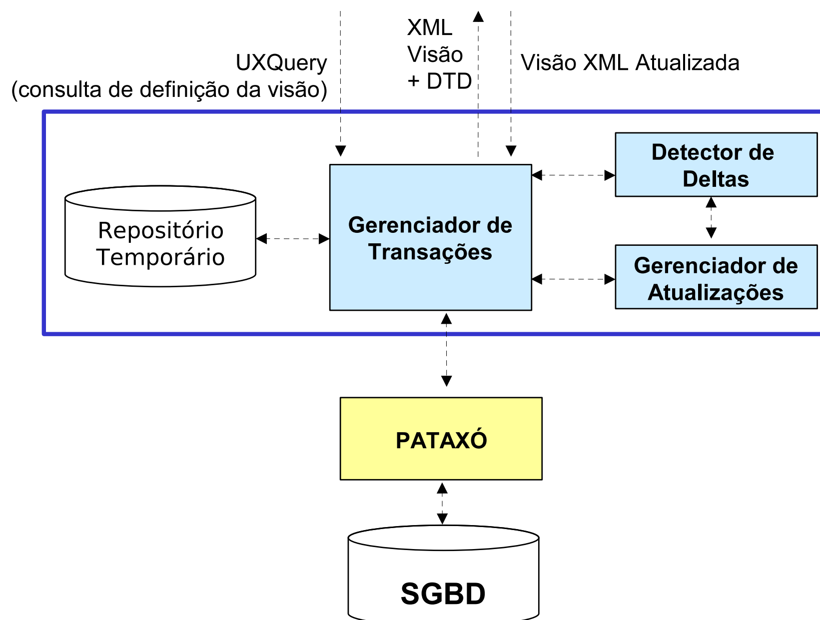


Figura 3.1: Arquitetura para a solução proposta.

relação à DTD da transação. Caso a visão não seja válida, a transação é abortada e o cliente é notificado. No caso da visão ser válida, o Gerenciador de Transações envia a consulta de definição da visão XML ao PATAXÓ.

Ao receber a visão atualizada, o PATAXÓ gera uma nova visão XML refletindo o estado atual da base de dados. Se esta nova visão XML for igual à visão original, então durante a transação não ocorreu nenhuma alteração na base de dados que afetasse a visão XML. Nesse caso, todas as atualizações feitas através da visão XML atualizada deverão ser encaminhadas para a base de dados relacional. Neste estágio, o Gerenciador de Transações está lidando, para a mesma transação, com três versões da visão XML:

- A visão XML *original* (O): é a visão XML que foi inicialmente entregue ao cliente no início da transação. No exemplo proposto, a visão original é apresentada na Figura 1.3.

- A visão XML *atualizada* (A): é a visão XML que foi alterada pelo cliente e enviada novamente ao sistema. A visão XML atualizada é apresentada na Figura 1.5.

- A *visão'*: é uma nova visão XML que é o resultado da mesma consulta que gerou a visão original, porém executada quando o cliente retornou a visão atualizada. A *visão'* é usada para capturar possíveis conflitos causados por atualizações externas em tuplas que estão na visão original. A Figura 3.3 apresenta um exemplo de visão'. Esta visão expressa as modificações apresentadas na Figura 1.4.

O módulo *Detector de Deltas* é responsável por detectar as diferenças entre as visões XML. Numa transação, duas comparações são efetuadas, conforme representado na Figura 3.2. A primeira é feita entre a visão *original* com a visão *atualizada* para detectar quais foram as atualizações feitas pelo cliente nesta visão. O *delta* resultante desta operação será usado como base para, enfim, atualizar a base de dados relacional. A segunda comparação é feita entre a visão *original* e a *visão'* a fim de detectar se ocorreram e quais foram as modificações no estado da base de dados durante a transação. Detalhes sobre estas comparações podem ser vistos na Seção 3.1.

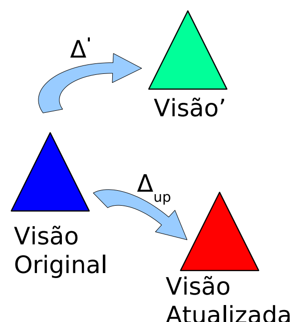


Figura 3.2: Visões XML e *deltas* envolvidos na transação. Δ_{up} representa as alterações do usuário e Δ' representa as alterações na base de dados

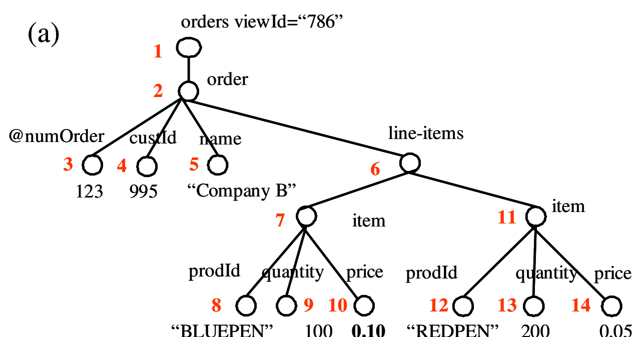


Figura 3.3: View' - Visão XML que representa o estado atual da base de dados.

Os *deltas* detectados pelo *Detector de Deltas* são enviados ao *Gerenciador de Atualizações*, que faz o mapeamento do *delta* para um formato compatível com o PATAXÓ.

O Gerenciador de Atualizações recebe os *deltas* e gera a partir destes as operações do PATAXÓ. O Gerenciador de Transações então recebe tais operações e tem a função de detectar e tratar os conflitos que possam existir, conforme apresentado no Capítulo 4. Após a detecção e tratamento, as operações são enviadas ao PATAXÓ para serem efetuadas na base de dados relacional.

3.1 Detector de *Deltas*

O *Detector de Deltas* é responsável por detectar as mudanças feitas na visão XML e por detectar as alterações na base de dados. Para tal, o módulo utiliza um algoritmo já existente para detectar *deltas* entre as visões.

O X-Diff (WANG, 2003) é o algoritmo adotado para este trabalho, principalmente por detectar as operações suportadas pelo PATAXÓ e por utilizar um modelo de árvores não-ordenadas. O MH-DIFF (CHAWATHE, 1997) não suporta inserções e exclusões em sub-árvores, mas apenas em nodos, o que não é adequado neste contexto. O XyDiff (COBENA, 2002) e o XMLTreeDiff (CURBERA, 1999) utilizam árvores ordenadas, o que também não é apropriado. Além disso, (WANG, 2003) mostra que em muitos casos o XyDiff gera resultados incorretos. Maiores detalhes podem ser visto na Seção 2.2.

Antes de seguir adiante, serão apresentadas algumas características do X-Diff que serão úteis neste contexto.

```
<orders viewId="786">
      <order numOrder="123">
            <custId>995</custId>
            <name>Company B</name>
            <line-items>
                  <item>
                        <prodId>BLUEPEN</prodId>
                        <quantity>100</quantity>
                        <price>0.10<?UPDATE FROM "0.05"?></price>
                  </item>
                  <item>
                        <prodId>REDPEN</prodId>
                        <quantity>200</quantity>
                        <price>0.05</price>
                  </item>
            </line-items>
      </order>
</orders>
```

Figura 3.4: Resultado do *delta* entre as visões original e visão', gerado pelo Algoritmo X-Diff

3.1.1 X-DIFF

O X-Diff (WANG, 2003) suporta inserções, exclusões e modificações. No Capítulo 2 estas operações foram apresentadas superficialmente. Nesta seção serão apresentados mais alguns detalhes relativos às operações. Especificamente, o X-Diff pode operar sobre sub-árvores e sobre nodos folha para inclusão e exclusão. Operações de modificação, somente são aceitas em nodos de texto, ou seja, nomes de elementos e nomes de atributos não podem ser modificados. Conteúdos de texto e valores de atributos podem ser modificados. As operações permitidas são expressas abaixo.

Inserção em nodo folha A operação *Insert(x(nome, valor), y)* insere um nodo folha x com o nome *nome* e valor *valor*. O nodo x é inserido como filho do nodo y.

Exclusão de nodo folha A operação *Delete(x)* exclui o nodo folha x.

Modificação de nodo texto Uma modificação de um nodo texto é expressa como *Update(x, novo-valor)*. O nodo x tem como novo valor *novo-valor*.

Inserção de sub-árvore A operação *Insert(T_x, y)* insere uma sub-árvore T_x (tendo o nodo x como raiz) como filho do nodo y.

Exclusão de sub-árvore A operação *Delete(T_x)* exclui a sub-árvore T_x tendo o nodo x como raiz). Quando não há dúvida em relação a qual nodo x deve ser acessado, esta operação pode ser expressa como *Delete(x)*.

Como foi discutido no Capítulo 2, uma importante característica do X-Diff é que este usa a relação de pai-filho para calcular o custo-mínimo para a correspondência dos nodos entre duas árvores T_1 e T_2. Esta relação é capturada através do uso da *assinatura* do nodo. A assinatura de um nodo x é expressa por *Name(x_1)/.../Name(x_n)/Name(x)/Type(x)*, onde (x_1/.../x_n/x) é o caminho da raiz até x, e *Type(x)* é o tipo do nodo x. No caso de x ser um elemento não-atômico, então sua assinatura não inclui *Type(x)* (WANG, 2003). Duas sub-árvores T_1 e T_2 correspondem se suas assinaturas forem a mesma.

3.2 Gerenciador de Atualizações

A função do Gerenciador de Atualizações é receber o *script* com as atualizações geradas pelo X-Diff e retornar um conjunto de operações escritas na linguagem de atua-

$E_1(O \rightarrow U)$ = *Update(9, 200), Update(13, 300), Insert(t_1, 6))*

```
t1 = <item>
       <prodId>NTBK</prodId>
       <quantity>100</quatity>
       <price>3.50</price>
     </item>
```

$E_2(O \rightarrow$ *visão')* = *Update(10, 0.10)*

Figura 3.5: Operações de atualização da visão original → visão atualizada; e da visão original → visão'

lizações do PATAXÓ. Para que se efetue este mapeamento, algumas questões devem ser tratadas. A principal delas está em lidar com a forma como o *delta* é expresso pelo X-Diff e como este *delta* deve ser entregue ao PATAXÓ. As atualizações no PATAXÓ devem especificar o caminho do nodo ou sub-árvore a ser atualizado (referenciado por *ref* nas atualizações, conforme apresentado na seção 2.1.3). Porém, as expressões de caminho no formato de saída do X-Diff não são explícitas, uma vez que seu *delta* é formado por marcações no próprio documento XML e não por uma lista de operações.

A Figura 2.17 apresenta o *delta* gerado pelo X-Diff entre a visão original e a visão atualizada e a Figura 3.4 representa o *delta* gerado pelo X-Diff entre a visão original e a visão'. A Figura 3.5 apresenta as operações equivalentes para estes *deltas*, expressos de uma forma resumida, utilizando o número dos nodos para identificação.

Para gerar o caminho *ref* de atualização, o Gerenciador de Atualizações se vale das chaves primárias das tabelas da base de dados como filtros nas expressões de caminho. Note que é necessário que as chaves apareçam nas visões para que estas sejam atualizáveis (BRAGANHOLO, 2003, 2004). Especificamente, para uma operação sobre o nodo x, utiliza-se o caminho p do nodo x até a raiz, buscando todas as chaves que são descendentes dos nodos em p.

No exemplo proposto, as chaves são *custId, numOrder* e *prodId*[3]. As regras de transformação de uma operação X-Diff numa operação de atualização do PATAXÓ são apresentadas a seguir. A função *generateRef* utiliza as chaves primárias para construir os filtros, como mencionado anteriormente. A forma geral de uma operação de atualização do PATAXÓ é <$tipo$, Δ, *ref*>.

Inicialmente, a função *generateRef*(x) obtém o nodo pai de x, x_n, e em seguida seu nodo pai x_{n-1} e assim sucessivamente até ser alcançado o nodo-raiz. Os elementos obtidos formam um caminho $p = x_1/.../x_{n-1}/x_n/x$. Dessa forma, para cada nodo y em p, o sistema busca nodos filho que são chaves primárias na base de dados relacional. Usa-se este conjunto de nodos para especificar um filtro que usa o nome do nodo e seu valor na visão XML.

A função **GeraOperacoes**, apresentada no algoritmo 2, recebe o *delta* do X-Diff e retorna a lista de operações no formato de atualizações do PATAXÓ. Basicamente a função percorre o *delta*, que é uma árvore XML, buscando as instruções de atualização. Para cada instrução é feito o mapeamento correspondente, conforme mostrado abaixo. O caminho XPath da atualização é fornecido pela função *generateRef*:

- *Insert(x(nome, valor), y)* é mapeado para <*delta*,*insert*, x, *generateRef(y)*>;

[3]Note que mesmo que o nome da chave na visão XML seja diferente do nome na base de dados relacional, o sistema é capaz de fazer a correspondência, utilizando a consulta de definição da visão.

- *Delete(x)* é mapeado para <*delete*, {}, *generateRef(x)*>;
- *Update(x, novo-valor)* é mapeado para <*modify*, {*novo-valor*}, *generateRef(x)*>;
- *Insert(T_x, y)* é mapeado para <*insert*, T_x, *generateRef(y)*>;
- *Delete(T_x)* é mapeado para <*delete*, {}, *generateRef(x)*>.

GeraOperacoes($Delta$)

Busca na árvore $Delta$ por instruções do X-Diff (As instruções são nodos delimitados por "<?" e "?>")
para cada instrução i em $Delta$ **faça**
  n = nodo que contém a instrução
  se i igual a "UPDATE" **então**
    Δ = nodo texto de n
    $caminho$ = generateref(n)
    $tipo$ = "modify"
    $operao$ = "<$type$,Δ,ref>"
    Adiciona $operacao$ à lista de operações
  fim se
  se i igual a "INSERT" **então**
    Δ = sub-árvore contida no nodo n
    $caminho$ = generateref(n)
    $tipo$ = "insert"
    $operao$ = "<$type$,Δ,ref>"
    Adiciona $operacao$ à lista de operações
  fim se
  se i igual a "DELETE" **então**
    Δ = conjunto vazio
    $caminho$ = generateref(n)
    $tipo$ = "delete"
    $operao$ = "<$type$,Δ,ref>"
    Adiciona $operacao$ à lista de operações
  fim se
fim para
Retorna lista de Operações

Algoritmo 2: Algoritmo para geração das operações de atualização do PATAXÓ a partir do X-Diff.

Como exemplo, as operações de E_1 da Figura 3.5 são mapeadas para as operações apresentadas na Figura 3.6

Atualização de Chaves e Ordenação de Operações

Apesar de não ser uma operação comum na prática, o usuário deste sistema ou da base de dados relacional pode realizar alterações nas chaves primárias diretamente na base de dados ou através da visão XML. Para que tais alterações não incorram em inconsistências nas atualizações estas operações precisam receber tratamento diferenciado. O problema pode ser dividido em dois casos para melhor apresentação.

- *Update(10, 200)* ≡ *<modify*, {200}, *orders/order[@numOrder= "123"and custId= "995"]/line-item/item[prodId= "BLUEPEN"]/quantity>*;
- *Update(13, 300)* ≡ *<modify*, {300}, *orders/order[@numOrder= "123"and custId= "995"]/line-item/item[prodId= "REDPEN"]/quantity>*;
- *Insert(t_1, 6)* ≡ *<insert*, t_1, *orders/order[@numOrder= "123"and custId= "995"] /line-item>*, com t_1 conforme apresentado na Figura 3.5.

Figura 3.6: Exemplo de Operações de atualização entregue ao PATAXÓ.

O primeiro caso a ser analisado envolve a alteração de chaves dentro da visão atualizada. Por exemplo, suponha que o usuário tenha feito atualizações na visão original, gerando uma visão atualizada conforme apresentada na Figura 3.7, em relação a visão original apresentada na Figura 1.3. Nesta transação, o usuário alterou algumas informações de $Order$ e trocou o valor de sua chave primária, $numOrder$, de 123 para 321. Além disso, o usuário acrescentou um novo $Order$, com $numOrder$ igual a 123, mesmo valor da antiga chave.

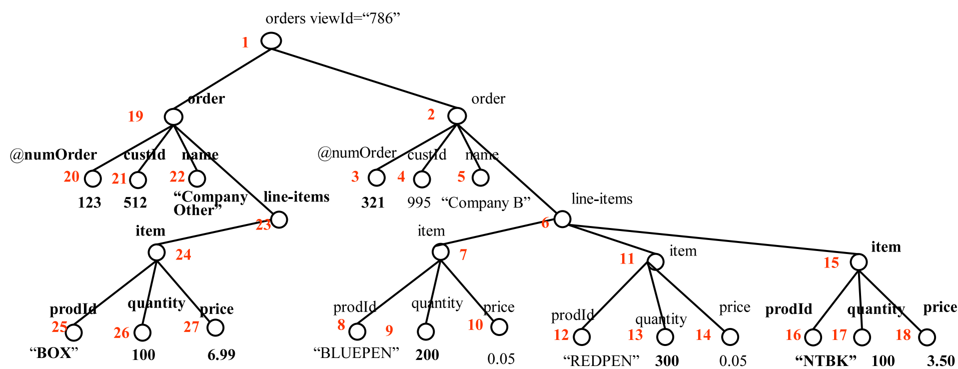


Figura 3.7: Visão XML com modificações de valores de chave. As alterações(em negrito) são relativas à visão original.

```
t2 = <order numOrder="123">
      <custId>512</custId>
      <name>Company B</name>
      <line-items>
            <item>
            <prodId>BOX</prodId>
            <quantity>100</quantity>
            <price>6.99</price>
            </item>
      </line-items>
     </order>
```

Figura 3.8: Árvore com um novo pedido, inserida na visão apresentada na Figura 3.7

Para que as alterações sejam feitas corretamente, o valor das chaves deve estar correto, portanto não se pode alterar elementos baseados em valores de chaves que ainda não fo-

ram criados ou em valores de chaves que já estão desatualizados. Portanto, o Gerenciador de Transação deve ordenar as operações visando o uso correto das chaves. A ordenação segue as seguintes regras:

1. As operações de modificação devem ser as primeiras a serem executadas;

2. Dentre as operações de modificação, as que modificam chaves devem ser as últimas a serem executadas;

3. Dentre as operações de modificação de chaves, devem ser executadas primeiro as que estiverem em maior nível de profundidade na árvore XML;

4. As operações de inserção e exclusão não necessitam ser ordenadas entre si.

Dessa forma, as operações de modificação podem utilizar as chaves com os valores originais para depois as chaves serem ordenadas. Além disso, as inserções podem ser feitas corretamente, mesmo que possuam valores de chaves originalmente utilizados em outros elementos.

O segundo caso ocorre quando uma chave primária foi alterada em visão', porém seu antigo valor está sendo usado como base para as atualizações do usuário na visão atualizada. Este caso atualmente não ocorre, uma vez que o tratamento de conflitos impede este tipo de combinação de operações, conforme será apresentado no Capítulo 4. Porém, como a arquitetura é flexível e pode prever outros tipos de tratamento de conflitos com outros tipos de regras, tal problema deve ser considerado e tratado.

Por exemplo, suponha que a atualização apresentada na Figura 3.9 tenha sido executada na base de dados enquanto o usuário estava atualizando sua visão em seu dispositivo móvel. Neste caso, as operações são escritas utilizando as chaves originais, presentes na visão entregue ao usuário. Porém, as atualizações no PATAXÓ devem ser feitas baseadas nas chaves atuais, ou seja, as que estão presentes em visão'. Além disso, qualquer regra de tratamento de conflito deve utilizar os mesmos valores de chaves a fim de comparar corretamente as tuplas.

```
//Modificando a chave primaria do produto "blue pen"
UPDATE "Product"
SET prodId = "BLPN"
WHERE prodId = "BLUEPEN"
```

Figura 3.9: Atualização da chave primária "BLUEPEN".

Para tratar tal situação, a utilização de chaves segue em duas etapas. Na primeira etapa as comparações de chaves são feitas com base nas chaves da visão original. Esta visão é usada na arquitetura como base na detecção do *delta* tanto para visão' quanto para a visão atualizada. Dessa forma, a utilização das chaves contidas na visão original permite a uniformização das chaves (veja Figura 3.2).

Na segunda etapa, é feito um mapeamento das chaves modificadas em visão'. Cada uma das alterações recebe, se necessário, o novo valor de chave, baseado no valor contido em visão'. Com este mapeamento, as atualizações podem ser efetuadas pelo PATAXÓ.

Cabe ressaltar que as etapas de ordenação e mapeamento de chaves são efetuadas após a detecção de conflitos e somente ocorrem com as operações que poderão ser executadas.

Supondo que a transação ocorreu com as operações da visão atualizada da Figura 3.7 e com a alteração no banco de dados presente na Figura 3.9, e que nenhuma operação entrou em conflito, as operações serão apresentadas ao PATAXÓ conforme abaixo:

1. *Update(9, 200)* $\equiv$ <*modify*, {200}, *orders/order[@numOrder= "123"and custId= "995"]/line-item/item[prodId= "BLUEPEN"]/quantity*>;

2. *Update(13, 300)* $\equiv$ <*modify*, {300}, *orders/order[@numOrder= "123"and custId= "995"]/line-item/item[prodId= "REDPEN"]/quantity*>;

3. *Update(3, 321)* $\equiv$ <*modify*, {321}, *orders/order[@numOrder= "123"and custId= "995"]/@numOrder*>;

4. *Insert*(t_1, *6)* $\equiv$ <*insert*, t_1, *orders/order[@numOrder= "321"and custId= "995"] /line-item*>, com t_1 conforme apresentado na Figura 3.5;

5. *Insert*(t_2, *1)* $\equiv$ <*insert*, t_2, *orders/order[@numOrder= "123"and custId= "995"] /line-item*>, com t_2 conforme apresentado na Figura 3.8.

3.3 Gerenciador de Transações

O Gerenciador de Transações tem por objetivo fornecer a comunicação entre este sistema e o PATAXÓ. Através deste módulo o sistema requisita e recebe do PATAXÓ as visões XML, suas DTDs, as consultas UXQuery, relatos de erros, entre outros. Esta também é a porta de comunicação do sistema com o usuário, enviando e recebendo as visões XML e entregando as notificações de erro ou sucesso das operações.

A principal função do Gerenciador de Transações, no entanto, é determinar quais operações podem ou não ser executadas, gerenciando o tratamento de conflitos. As funcionalidades do Gerenciador de Transações será melhor abordada no Capítulo a seguir.

4 TRATAMENTO DE CONFLITOS

Devido às restrições impostas para este tipo de aplicação com transações desconectadas e de tempo indefinido, o modelo proposto para as transações é otimista e não bloqueante, ou seja, quando um cliente recebe uma visão para analisá-la e, possivelmente modificá-la, o sistema não bloqueia estes dados, permitindo que outros clientes possam gerar outras visões XML que compartilhem todos ou uma parte dos dados, podendo também modificá-los. Outros usuários também podem modificar a base de dados através de outras aplicações. Neste contexto, podem ocorrer operações concorrentes que, se executadas em conjunto, produzirão resultados indesejados aos usuários ou colocará o banco de dados em um estado inconsistente. Esta situação é chamada de conflito.

Dentro do contexto deste trabalho, um conflito ocorre quando uma visão XML original, usada como base para as modificações feitas por um usuário, não corresponde mais ao estado atual da base de dados, podendo gerar resultados inadequados ou afetar modificações feitas por outros usuários. Numa situação de conflito, existem duas modificações concorrentes: as modificações feitas pelo usuário sobre a visão original e as modificações que ocorreram na base de dados que tornam a visão original desatualizada.

A detecção de conflitos não é uma tarefa fácil, uma vez que um conflito pode ter diferentes impactos dependendo da aplicação. No caso dos pedidos, por exemplo, a exclusão de um produto da base de dados significa que o cliente não pode mais requisitá-lo. Suponha que em um sistema acadêmico, o departamento D deseja corrigir o nome de algumas disciplinas, mas não está interessado nos dados de cada estudante ligado ao departamento, uma vez que os dados dos estudantes na aplicação são apenas para fins estatísticos, para, por exemplo, checar a necessidade de novas turmas para disciplinas. Nesse caso, a exclusão de um estudante da base de dados enquanto se está analisando dados sobre o departamento D não deverá gerar um conflito grave. Quando o departamento D retornar ao sistema a visão atualizada, nenhuma das operações requeridas terá efeito colateral devido ao estudante ter sido removido. Note que, apesar de o estudante estar previamente na visão entregue ao departamento D, este não será inserido novamente na base de dados quando a visão retornar.

O *Detector de Deltas* utiliza o X-Diff para gerar o *delta* entre a visão XML original O e a *visão'*, que expressa o estado atual da base de dados. Se este *delta* for vazio, então não há nenhuma modificação feita na base de dados que tenha efeito sobre a visão entregue ao cliente. Nesse caso, as atualizações feitas pelo cliente podem ser feitas sem geração de conflitos. As operações são traduzidas para a linguagem de atualização do PATAXÓ, que faz o mapeamento para as atualizações na base de dados relacional.

Caso o *delta*, entre a visão original e visão', não seja vazio, ou seja, a visão original e a visão' não forem iguais, então ocorreram modificações na base de dados que geraram efeito sobre a visão entregue ao cliente. Nesse caso existe um conflito na transação.

4.1 Tratamento de conflitos

Devido à dependência das aplicações, a criação de um tratamento de conflito único é uma tarefa difícil, uma vez que a dependência semântica dos dados pode variar.

Na abordagem proposta existem três modalidades operacionais que determinam como a dependência entre os dados ocorre na visão XML. A escolha de um dos modos pode ficar sob responsabilidade de um administrador da base de dados ou de um usuário que conheça a aplicação.

O primeiro modo é o **modo restritivo**, no qual nenhuma atualização pode ser efetuada quando existe alguma diferença entre a visão atualizada e a visão'. Neste modo, qualquer elemento da visão XML pode potencialmente ter sido usado na decisão de se criar alguma modificação na visão. Dessa forma, todos os dados na visão XML são dependentes entre si. No exemplo da visão de pedidos de produtos, um usuário poderia afirmar que alterações no preço de qualquer produto, não necessariamente o produto que se está sendo comprado, influencia na sua decisão de compra.

O segundo modo é o **modo relaxado**, que oferece mais liberdade de atualizações concorrentes. Neste modelo, as operações que não causam conflitos são transferidas para a base de dados e as outras são abortadas. Para manter a consistência da base de dados, assume-se que algumas operações sobre um dado podem não depender de uma outra operação. Neste caso entende-se que estas funções podem ser efetuadas sem causar inconsistência na base de dados. Para reconhecer tais casos, foi criado um conjunto de regras de dependência de dados, baseadas apenas na estrutura da visão XML. Note que o sistema não solicita ao usuário qualquer informação semântica dos dados na visão XML ou da base de dados. Por não ter esse conhecimento, em alguns casos as regras colocam as operações como conflitantes mesmo que semanticamente não o sejam. As regras foram criadas tomando a saída mais conservadora visto que não se tem conhecimento da semântica dos dados.

O terceiro modo (**modo de chave**) é uma simplificação do **modo relaxado**, no qual apenas a chave primária identifica o elemento. As regras são bastante semelhantes, porém difere do modo relaxado por utilizar apenas a chave como dependência de dados para um objeto da base de dados.

4.2 Regras de Detecção de Conflitos - Modo Relaxado

Nesta seção serão apresentadas as regras para tratamento de conflitos no modo relaxado. Antes de apresentar as regras de detecção de conflitos, deve-se considerar os pressupostos abaixo:

A) O usuário ao fazer uma modificação, pode levar em conta valores de atributos que não são chaves. Na verdade o usuário não necessariamente sabe quais são os atributos chaves de uma visão XML. Por exemplo, um cliente pode decidir aumentar a quantidades de canetas azuis no seu pedido, levando em conta o preço do produto. Caso o preço do produto seja alterado na base de dados, é possível que o usuário não queira mais aumentar a quantidade pedida. Nesse exemplo, o atributo *price* (que não é chave) foi usado como parâmetro na decisão de se modificar o atributo *quantidade* na visão XML.

B) As regras para detecção dos conflitos são baseadas na estrutura da visão gerada. Por isso, as visões geradas devem ser bem-aninhadas, ou seja, serem atualizáveis de acordo com as regras apresentadas em (BRAGANHOLO, 2004).

numOrder	custId	name	prodId	quantity	price
123	995	Company B	BLUEPEN	200	0.05
123	995	Company B	REDPEN	200	0.05
...	...	...	...	...	...

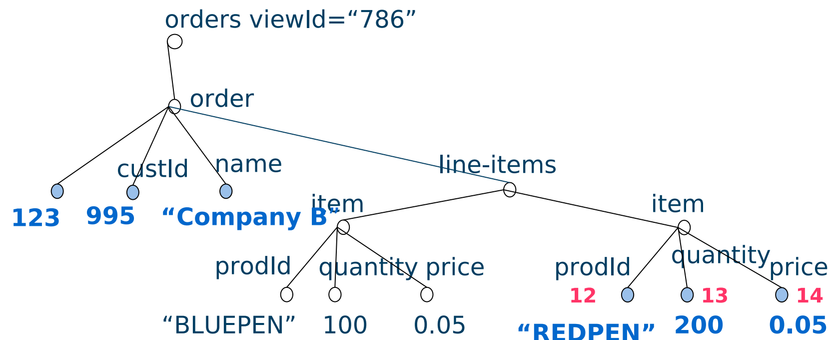


Figura 4.1: Exemplo de Tupla na base de dados relacional e na visão XML. Os elementos em negrito na visão XML correspondem a tupla hachurada na Tabela

4.2.1 Regra 1: Nodos Folha pertencentes ao mesmo elemento-estrela

Na estrutura da visão XML, nodos folha descendentes do mesmo nodo-estrela (veja Seção 2.1.2) formam uma mesma tupla na base de dados. A Figura 4.1 apresenta um exemplo de um objeto na base de dados relacional e na visão XML. Os nodos 12, 13 e 14 (que vêm do mesmo registro) pertencem ao mesmo elemento-estrela. Formalmente pode-se escrever:

Definição 4.1 *Seja $F = \{f_1, ..., f_n\}$ ($n \geq 1$) o conjunto de nodos folha descendentes do nodo-estrela e em uma dada visão XML v. Além disso, garante-se que e é o ancestral mais próximo dos nodos em F. Se qualquer f_i em F for modificado na visão atualizada e qualquer f_j estiver modificado na visão', então a modificação na visão atualizada não poderá ser efetuada.*

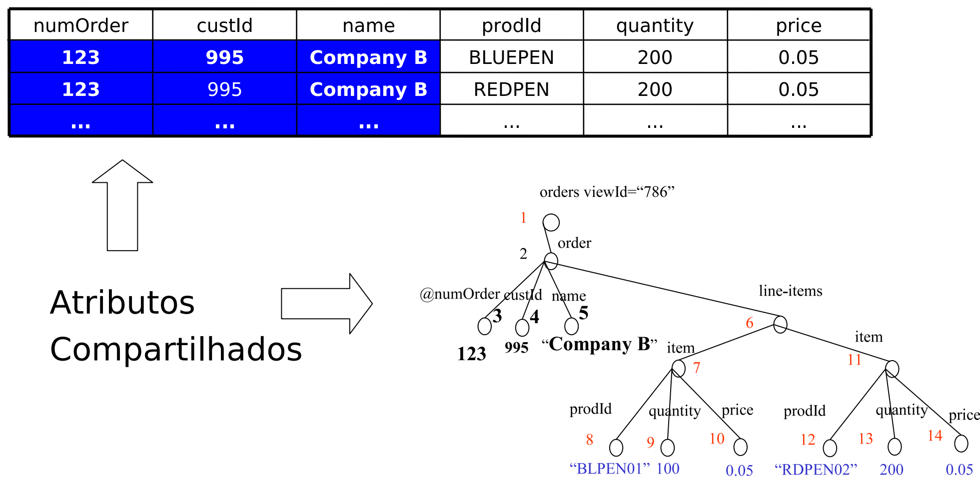
Um exemplo de tal caso pode ser visto na Figura 1.5. O nodo 9 (quantidade de canetas azuis) é modificado de 100 para 200 unidades. Esta operação não poderá ser efetuada pois entra em conflito com a operação de modificação sobre o nodo 10, na visão' (3.3).

Modificações em nodos que pertencem a diferentes elementos-estrela podem coexistir sem conflitos, uma vez que pertencem a diferentes tuplas na base de dados relacional. Um exemplo é a modificação do nodo 15 (quantidade de canetas vermelhas) de 200 para 300 unidades (1.5), que pode ser efetuada sem entrar em conflito com a modificação do nodo 11 (preço das canetas azuis na Figura 3.3) que foi modificada na visão'.

4.2.2 Regra 2: *Sub-árvores estrela* dependentes

As sub-árvores estrela podem conter outras sub-árvores estrela como descendentes. Isto ocorre quando existem junções entre duas tabelas na definição da visão XML. No exemplo da Figura 1.3, a visão XML é formada pela junção das tabelas *Customer*, *Order* e *LineOrder* (a definição da visão pode ser vista nas Figura 2.1).

Enquanto na visão relacional os campos das tabelas *Customer* e *Order* se repetem para cada tupla de LineOrder, isso não acontece na visão XML, uma vez que o aninhamento da estrutura é feito para eliminar as redundâncias, conforme a propriedade das visões atualizáveis (veja Seção 2.1.3). Dessa forma, nota-se no exemplo que os elementos de *item* estão semanticamente ligados aos elementos de *order* Na Figura 4.2 cada sub-árvore *item* representa parte de uma tupla da base de dados relacional. Os nodos *numOrder*, *custId* e *name*, da sub-árvore *order*, são compartilhados na visão XML, ou seja, o mesmo elemento representa em parte mais de uma tupla a base relacional.



numOrder	custId	name	prodId	quantity	price
123	995	Company B	BLUEPEN	200	0.05
123	995	Company B	REDPEN	200	0.05
...	...	...	...	...	...

Figura 4.2: Exemplo de Tupla na base de dados relacional e na visão XML. Os elementos hachurados representam parte de mais de uma tupla na relação

Sendo assim, modificações feitas na base de dados sobre a sub-árvore *order* entram em conflito com atualizações do usuário sobre alguma sub-árvore *item*. A situação inversa, quando na base de dados informações sobre algum item foram modificadas e o usuário faz alguma modificação sobre *order*, esta operação pode ser feita sem gerar conflitos. Formalmente, pode-se escrever:

Definição 4.2 *Seja e_1 e e_2 duas sub-árvores estrela em uma dada visão XML. Seja F_1 = $\{f_{1_1}, ..., f_{1_n}\}$ ($n \geq 1$) o conjunto de nodos folha descendentes de e_1 mas não de sua sub-árvore estrela e F_2 = $\{f_{2_1}, ..., f_{2_k}\}$ ($k \geq 1$) o conjunto de nodos folha descendentes de e_2, mas não de sua sub-árvore estrela. Adicionalmente, seja e_1 um ancestral de e_2. Se algum f_{2_i} em L_2 é modificado na visão atualizada, e algum f_{1_j} em L_1 for modificado na visão', então as operações entram em conflito e a modificação em f_{2_i} é abortada.*

Note que em todas as regras acima, é necessário que a correspondência entre os nodos na visão atualizada e na *visão'* seja conhecida. Por exemplo, é necessário saber que o nodo 12 na visão atualizada (Figura 1.5) corresponde ao nodo 12 na *visão'* (Figura 3.3). A correspondência pode ser feita usando uma variação do algoritmo apresentado na Seção 4.4.

4.2.3 Regra 3: Nodo-estrela e seus nodos folha descendentes.

Assim como expresso na regra 1, os nodos folha de um mesmo elemento-estrela não podem ser atualizados concorrentemente, uma vez que cada nodo folha representa um atributo de uma tupla. Da mesma forma, o nodo-estrela representa em si a própria tupla da base de dados. Logo, de acordo com o pressuposto A, a exclusão de um nodo-estrela

não pode ocorrer concorrentemente a uma modificação de um nodo-folha deste nodo-estrela.

Um exemplo pode ser visto na Figura 4.3. Nesse caso o cliente requer a exclusão das canetas azuis. Porém este elemento foi modificado na base de dados, como é apresentado na Figura 3.3. Nesse caso, a operação gera um conflito, uma vez que o elemento a ser apagado não é mais o mesmo e o valor do atributo alterado na base de dados (no caso o preço), pode ter sido o critério utilizado para que o cliente tomasse a decisão de excluir o item. Formalmente pode-se escrever:

Definição 4.3 *Seja* $L = \{l_1, ..., l_n\}$ *(*$n \geq 1$*) o conjunto de nodos folha descendentes do nodo-estrela* s *na visão XML* v*. Garante-se que não há um nodo-estrela ancestral a* s *em* L*. Se qualquer* l_i *pertencente a* L *foi modificado em* visão'*, então a exclusão do nodo* s *em* $atualizada$ *é rejeitada (modo restritivo). Se o nodo* s *foi excluído em visão', então qualquer modificação em qualquer* l_i *pertencente a* L *é rejeitada.*

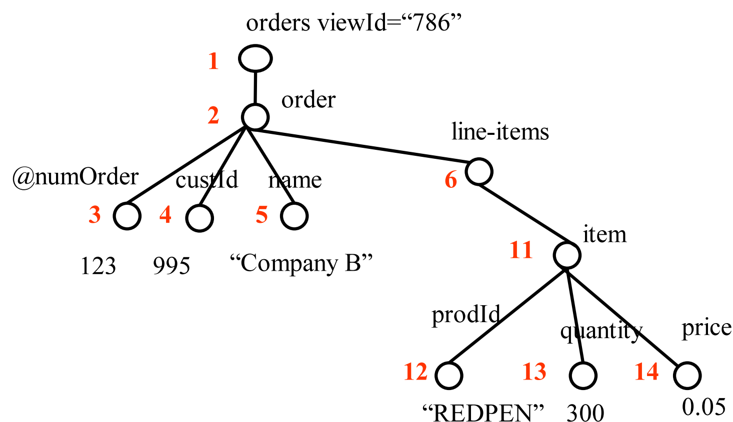


Figura 4.3: Visão retornada pelo cliente, na qual o elemento BLUEPEN é excluído.

4.2.4 Regra 4: Nodos-estrela dependentes

Assim como apresentado na regra 2, os *nodos-estrela* podem conter outros *nodos-estrela* como seus descendentes, que ocorrem quando existem junções entre duas tabelas na definição da visão. A exclusão ou inserção de uma sub-árvore (*nodo-estrela*) é analisada usando os mesmos argumentos. Dessa forma, quando um nodo-folha de um *elemento-estrela* é modificado (de acordo com o pressuposto A), não corresponde mais ao mesmo objeto na base de dados, portanto qualquer alteração feita com base no seu estado anterior não deve ser efetuada.

Por exemplo, se em visão' um elemento *Order* teve um dos seus nodos folha modificados, como o nodos *numOrder*, qualquer inserção ou exclusão de um elemento *item* aninhado a este elemento *Order* gerará conflito e não será efetuada.

Por outro lado, se em visão' foi inserido ou excluído algum elemento-estrela, uma modificação no seu *elemento-estrela* pai pode ser efetuada sem gerar conflito. Por exemplo, a exclusão de um elemento $item$ em visão' não altera o objeto *Order*, portanto a modificação requerida na visão atualizada (modificação um atributo *numOrder*) é permitida.

4.2.5 Aplicação das Regras para Detecção dos Conflitos

A Tabela 4.1 apresenta os pré-requisitos para a execução de uma operação, baseadas nas regras apresentadas anteriormente. A operação requisitada pelo usuário (visão atualizada) aparece verticalmente e a operação já realizada, vista em Visão', aparece horizontalmente.

A seguir, cada um dos casos será explicado mais detalhadamente. A dupla de operações será representada na forma **(operação sobre visão', operação sobre visão atualizada)**.

Tabela 4.1: Pré-requisitos para execução no modo "relaxado". Na horizontal há operações em *visão'* e na vertical na visão atualizada.

		Insert	Modify	Delete
Operações sobre a visão atualizada	Insert	OK	Regra 4	Regra 4
	Modify	OK	Regras 1 e 2	Regras 3 e 4
	Delete	OK	Regras 3 e 4	OK
				Operações sobre visão'

Combinações não-Conflitantes

(Insert, Insert) Duas operações de inserção não são conflitantes. Porém, podem ocorrer casos de reutilização de alguma chave primária. Este caso, no entanto, assim como erros de chave estrangeira e outros, não são problemas de conflito mas erros de SQL e serão tratados pelo PATAXÓ.

(Insert, Modify) e (Insert, Delete) Estas operações não são conflitantes, pois não é possível que uma instância inserida por *visão'*, ou seja, não existente durante a geração da visão original, tenha sido alterada ou excluída pela visão $atualizada$.

(Delete, Delete) Duas operações de exclusão podem ocorrer concorrentemente, uma vez que seu resultado será o mesmo caso as duas visões estejam alterando uma mesma instância. Para o caso de instâncias sub-árvores estrela também não há conflito, pois a exclusão de um elemento implica na exclusão de seus sub-elementos.

Combinações Conflitantes

(Modify, Modify) Conforme os pressupostos A e B, as modificações sobre atributos de um mesmo objeto do mundo real não devem ser feitas concorrentemente. Por isso, as modificações devem passar pelas regras 1 e 2 para verificação de conflitos.

(Modify, Insert) e (Modify, Delete) Uma vez que qualquer informação contida em um elemento pode ter sido usada como critério para uma atualização sobre ele, operações de inserção ou exclusão requeridas pelo cliente podem entrar em conflito caso algum elemento tenha sido modificado em visão'.

No caso da inserção por parte do usuário, um exemplo de conflito (seguindo a visão original da Figura 1.3 seria a inserção de um elemento *item* em uma visão cujo nome da empresa Cliente tenha sido modificado. Esse conflito seria detectado pela regra 4.

No caso de exclusão por parte do usuário, além da regra 4 (semelhante ao exemplo acima) deve-se considerar a regra 3. Por exemplo, caso um cliente exclua um *item* cujo preço foi modificado na base de dados, será gerado um conflito detectado pela regra 3.

(Delete, Insert) Esta combinação de operações pode gerar conflitos detectados pela regra 4. Por exemplo, não poderá ser efetivada uma operação de inserção de um item por parte do cliente se o pedido como um todo foi excluído da base de dados. Nesse caso, um conflito deve ser gerado e esta operação ser abortada. O caso inverso (a exclusão de um item de um pedido enquanto a base de dados está recebendo um novo pedido de uma outra empresa) pode ser feito sem nenhum tipo de conflito

(Delete, Modify) Esta combinação é analisada pelas regras 3 ou 4. Por exemplo, uma operação de alteração do preço de um item no pedido não poderá ser efetuada caso este item tenha sido excluído do pedido. Da mesma forma, essa operação não será efetuada se o pedido tiver sido excluído por completo da base de dados.

4.3 Regras de Detecção de Conflitos - Modo de Chaves

O modo de chaves possui um conjunto de regras mais simplificado e é utilizado para aplicações onde apenas a chave primária identifica o elemento e é a única usada para decisões de atualizações da base de dados. Este modelo possui apenas duas regras simples, que serão apresentadas a seguir.

4.3.1 Regra 5: Modificação concorrente de mesmo nodo folha

Na estrutura da visão XML, um mesmo nodo folha não pode ser alterado concorrentemente pois isto geraria alterações concorrentes em um mesmo elemento na base de dados relacional. Por exemplo, se o usuário quiser atualizar o atributo *price* de um item mas este já tiver sido atualizado na base de dados, sua requisição de atualização não será aceita. Formalmente pode-se escrever:

Definição 4.4 *Seja $F = \{f_1, ..., f_n\}$ ($n \geq 1$) o conjunto de nodos folha em uma dada visão XML v. Se qualquer f_i em F for modificado na visão atualizada e estiver modificado na visão', então a modificação na visão atualizada não poderá ser efetuada.*

4.3.2 Regra 6: Nodos Chaves pertencentes ao mesmo elemento-estrela

Como apresentado na regra 1, na estrutura da visão XML os nodos folha descendentes do mesmo nodo estrela formam uma mesma tupla na base de dados. Neste modelo, se qualquer nodo chave do elemento-estrela for modificado na base de dados, nenhuma operação sobre os outros nodos deste mesmo elemento-estrela poderá ser efetuada. Por exemplo, se na base de dados for modificado o atributo *prodId*, uma modificação em *quantity* deste elemento não poderá ser executada.

Formalmente pode-se escrever:

Definição 4.5 *Seja $F = \{f_1, ..., f_n\}$ ($n \geq 1$) o conjunto de nodos folha descendentes do nodo-estrela e em uma dada visão XML v e seja $C = \{c_1, ..., c_n\}$ ($n \geq 1$) o subconjunto de F que contém os nodos folha que são chaves do nodo-estrela e. Além disso, garante-se que e é o ancestral mais próximo dos nodos em F. Se qualquer c_i em C for modificado*

na visão atualizada e qualquer f_j estiver modificado na visão', então a modificação na visão atualizada não poderá ser efetuada.

4.3.3 Aplicação das Regras para Detecção dos Conflitos - Modo de Chaves

A Tabela 4.2 apresenta os pré-requisitos para a execução de uma operação, baseadas nas regras apresentadas anteriormente. A operação requisitada pelo usuário (visão atualizada) aparece verticalmente e a operação já realizada, vista em Visão', aparece horizontalmente.

A seguir, cada um dos casos será explicado mais detalhadamente. A dupla de operações será representada na forma **(operação sobre visão', operação sobre visão atualizada)**.

Tabela 4.2: Pré-requisitos para execução no modo de chaves. Na horizontal há operações em *visão'* e na vertical na visão atualizada.

Operações sobre a visão atualizada		Insert	Modify	Delete
	Insert	OK	OK	Regra 5
	Modify	OK	Regras 5 e 6	Regras 5
	Delete	OK	Regras 5	OK

Operações sobre visão'

As combinações (Insert, Insert), (Insert, Modify), (Insert, Delete) e (Delete, Delete) não são conflitantes, pelos mesmo motivos apresentados no modo relaxado (seção 4.2.5).

A combinação (Modify, Insert) não é conflitante neste conjunto de regras, pois não há como ter sido modificado um nodo na base de dados que está para ser inserido pelo usuário na visão atualizada. Caso ocorra problemas de chaves, este será um erro na base de dados, reportado pelo PATAXÓ.

Se um elemento for excluído ou modificado na base de dados, então nenhuma modificação nos nodos poderá ser efetuada, portanto as combinações (Delete, Insert) e (Delete, Modify) devem consultar a regra 5. Por exemplo, se uma sub-árvore *order* foi excluída na base de dados, nenhuma sub-árvore *item* poderá ser inserida neste *order* pela visão atualizada.

Se na base de dados for modificada uma chave, então os elementos que dependem daquela chave não poderão ser modificados, conforme acontece na combinação (Modify, Modify).

4.4 Notificação do Usuário

Qualquer que seja o modo de configuração do sistema, é necessário informar ao usuário quais operações foram passadas para a base de dados e quais foram abortadas.

Para isso o sistema gera um *merge* dos dados atualizados e os que representam o estado da base de dados. O algoritmo, que é apresentado a seguir, utiliza como base a visão original.

1. Pegue cada operação de exclusão u = Delete(x) em $E(O \rightarrow$ *visão'*) e marque x na visão XML original. A marca é feita adicionando-se um novo nodo pai *pataxo:DB-DELETE* a x, onde *pataxo* é um *namespace* pré-fixado. Este novo elemento é conectado ao nodo pai de x.

2. Pegue cada operação de inserção u = Insert(T_x, y) em $E(O \rightarrow$ *visão'*), insira T_x abaixo de y e adicione um novo nodo pai *pataxo:DB-INSERT* a T_x. Coloque o nodo recém criado como filho de y. De forma semelhante deve ser feito para inserção de nodos folha: pegue cada operação u = Insert(x(name, value), y), insira-o na visão original e coloque como pai o nodo *pataxo:DB-INSERT* em x e coloque-o como filho de y.

3. Pegue cada operação de atualização u = Update(x, novo-valor) in $E(O \rightarrow$ *visão'*), insira um novo elemento com o valor igual a *new-value*. Coloque o nodo *pataxo:DB-MODIFY* como filho de x. f x.

Após este algorimo, é necessário aplicar as operações extraídas do *delta* entre a visão original e atualizada. Nesta etapa, os elementos de marcação do sistema recebem um atributo STATUS, que mostra se a operação foi efetivada ou abortada.

1. Adicione a cada operação de exclusão u = Delete(x) em $E(O \rightarrow U)$ um novo nodo *pataxo:CLIENT-DELETE STATUS="ACCEPT"* a x e coloque-o como pai de x.

2. Para cada operação de inserção u = Insert(T_x, y) em $E(O \rightarrow U)$, adicione T_x abaixo de y e adicione um novo nodo pai *pataxo:CLIENT-INSERT STATUS="ACCEPT"* a T_x. Coloque o novo elemento criado como filho de y. Em seguida, para cada operação u = Insert(x(name, value), y), insira-o na visão original, adicione o novo nodo *pataxo:CLIENT-INSERT STATUS="ACCEPT"* a x e coloque-o como filho de y.

3. Para cada operação de modificação u = Update(x, novo-valor) em $E(O \rightarrow U)$, adicione um novo elemento *pataxo:CLIENT-MODIFY* com o valor igual a *novo-valor*. Coloque o elemento *pataxo:CLIENT-MODIFY* como filho de x. Se u está marcado em E, então adicione o atributo *STATUS* em *pataxo:CLIENT-MODIFY* com valor igual a *ABORT*. Se não, então adicione o atributo *STATUS* com valor igual a *ACCEPT*.

A Figura 4.4 apresenta o resultado do *merge* de exemplo. Podem haver elementos com mais de uma marcação de conflito. Por exemplo, suponha que um cliente tenha alterado o preço da caneta azul para 0.02. Neste caso, o elemento preço terá duas marcações.

```
<preço>0.05
   <pataxo:DB-MODIFY>0.10</pataxo:DB-MODIFY>
   <pataxo:CLIENT-MODIFY STATUS="ABORT">0.02</pataxo:CLIENT-MODIFY>
</preço>
```

O conflito causado pelo aumento do preço das canetas vermelhas não foi resolvido automaticamente porque pode ser que esta resolução de conflito possa não ser desejável em todas as aplicações.

Após a execução do algoritmo de *merge*, o Gerenciador de Transações recebe a nova visão *merged*[1]. O sistema gera uma nova visão O (visão original), agora com os dados atualizados, e envia para o cliente junto com a visão *merged*. Através destes dados o cliente pode verificar o resultado de sua transação e gerar uma nova transação a partir do novo estado do banco para, por exemplo, submeter novamente operações anteriormente abortadas.

[1]Esta visão não faz parte da transação própriamente dita, mas é uma visão auxiliar usada apenas para reportar o resultado das operações. Por isso não é considerada como parte do grupo de visões da transação e não segue a mesma DTD das outras visões XML tratadas na transação.

```
<orders viewId="786">
   <order numOrder="123">
      <custId>995</custId>
      <name>Company B</name>
      <line-items>
         <item>
            <prodId>BLUEPEN</prodId>
            <quantity>100
                <pataxo:CLIENT-MODIFY STATUS="ABORT">
                200
                </pataxo:CLIENT-MODIFY>
            </quantity>
            <price>0.05<pataxo:DB-MODIFY>0.10</pataxo:DB-MODIFY></price>
         </item>
         <item>
            <prodId>REDPEN</prodId>
            <quantity>200
                <pataxo:CLIENT-MODIFY STATUS="ACCEPT">
                300
                <pataxo:CLIENT-MODIFY>
            </quatity>
            <price>0.05</price>
         </item>
         <pataxo:CLIENT-INSERT STATUS="COMMIT">
            <item>
              <prodId>NTBK</prodId>
              <quantity>100</quantity>
              <price>3.50</price>
            </item>
         </pataxo:CLIENT-INSERT>
      </line-items>
   </order>
</orders>
```

Figura 4.4: Resultado do algoritmo de *merge*.

5 IMPLEMENTAÇÃO DA ARQUITETURA

A fim de validar as técnicas propostas neste trabalho, foi implementado um protótipo para a arquitetura proposta. Este capítulo apresenta de forma geral o protótipo e alguns exemplos de sua utilização.

O sistema foi desenvolvido em Java (SUN, 2002), usando o pacote Xerxes (APACHE, 2005) para manipulação dos documentos XML. A construção do programa segue a mesma estrutura da arquitetura apresentada no Capítulo 3, integrando o sistema PATAXÓ e os módulos do X-Diff para detecção do *delta*.

O módulo Controle de Transação é o que reúne maior número de funcionalidades. A detecção e tratamento de conflito é dividida dentro do Controle de Transação nos sub-módulos apresentados na Figura 5.1. O Detector de conflito recebe as operações já adaptadas ao formato de atualização do PATAXÓ e faz uma série de testes, fornecidos pelo Módulo de Regras. O Módulo de Regras implementa as Tabelas 4.1 e 4.2 apresentadas no Capítulo 4. Para conhecer detalhes como quais são os nodos-estrela ou se os nodos pertencem ao mesmo objeto do mundo real, o Detector de Conflito consulta o módulo de Manipulação da *Query Tree*[1]. A saída ao usuário é gerada pelo módulo de Retorno ao Usuário.

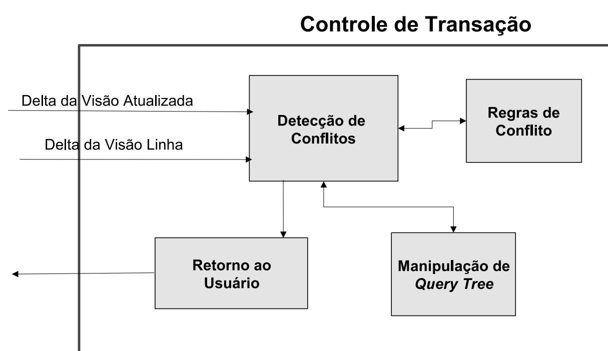


Figura 5.1: Módulos responsáveis pela detecção de conflitos, contidos no módulo Controle de Transação.

O repositório temporário (local onde são armazenadas as visões XML, as definições UXQuery e as *query trees*) é atualmente formado por um grupo de diretórios específicos. Cada arquivo recebe como nome o número de sua transação. Um exemplo de repositório é apresentado na Figura 5.2. O módulo de repositório é bastante simples, responsável apenas por acessar e armazenar tais arquivos.

[1]A *query tree* é gerada pelo PATAXÓ e fica armazenada no Repositório Temporário.

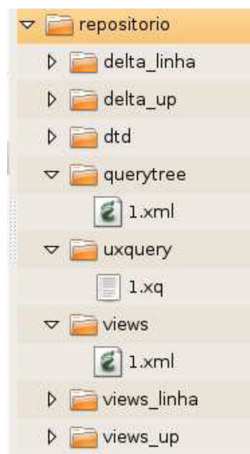


Figura 5.2: Exemplo de repositório temporário.

A seguir é apresentado um exemplo de uso do sistema, abordando o uso concorrente de transações.

5.1 Locadora On-Line

O exemplo é formado por uma base de dados simples (Figura 5.3), para ser usada em uma locadora onde os clientes podem, via dispositivos móveis, ver suas reservas de filmes e cancelar reservas. Suponha que, para reservar outros filmes, o cliente via uma aplicação própria, "cópia e cola" novos filmes disponíveis no seu documento XML. Na mesma aplicação um funcionário pode consultar o catálogo de filmes por categoria e também tem autorização para alterar o cadastro. Todos os dados apresentados são gerados pelo protótipo apresentado anteriormente.

A consulta UXQuery apresentada na Figura 5.4 gera a visão XML com as reservas feitas pelo cliente de identificação "1" (André Vargas). A partir dessa consulta o sistema gerou a visão XML apresentada na Figura 5.5.

Suponha que o cliente cancele sua reserva do filme "Tratamento de Choque" e coloque todas as reservas para o dia 15 de abril. A atualização fica conforme apresentado na Figura 5.6. Neste ambiente, não ocorreu nenhuma atualização na base de dados que afetasse a visão XML. Nesse caso não há conflito. O sistema gera a saída conforme apresentado na Figura 5.7[2].

Nesta aplicação, um funcionário pode acessar o cadastro dos filmes, usando uma visão XML com uma estrutura diferente. A visão abrange os filmes agrupados por categoria. O funcionário então decide consultar os filmes de suspense. A consulta UXQuery é apresentada na Figura 5.8 e a visão gerada é apresentada na Figura 5.9.

O funcionário, utilizando a visão XML, verifica que o DVD do filme "Os Outros" está danificado e deve ser excluído temporariamente do catálogo. O funcionário decide fazer a operação através da visão XML. A visão atualizada é apresentada na Figura 5.10. Simultaneamente, um outro funcionário, através de uma outra aplicação, decide atualizar o nome de um filme, conforme apresentado na Figura 5.11.

Suponha uma situação diferente, em que esta visão usada pelo funcionário foi enviada

[2]Esta saída é informada no sistema para fins de validação. O formato apresentado na seção 4.4 dificulta o entendimento no contexto deste capítulo.

Filme

filmeId	Titulo	categoriaId
1	Senhor dos Anéis	6
2	Os Outros	5
3	Sexto Sentido	5
4	Lavador de Almas	1
5	Munique	1
6	Matrix Revolution	6
7	Tratamento de Choque	3

Cliente

nome	clienteId	email
André Vargas	1	apvargas@inf.ufrgs.br
Valdir Xavier	2	valdix@terra.com
Raquel Prisco	3	raquel@supermail.net

Categoria

categoriaId	nome_categoria
1	Drama
2	Terror
3	Comédia
4	Documentário
5	Suspense
6	Aventura

Reserva

clienteId	filmeId	Data
1	1	12/04/07
1	2	13/04/07
1	7	13/04/07
2	7	20/04/07

Chaves Primárias:

Filme : **filmeId**

Cliente : **clienteId**

Categoria : **categoriaId**

Reserva: **filmeId,data**

Chaves Estrangeiras:

Filme : **categoriaId -> (Categoria)**

Reserva: **filmeId --> (fime)**
cienteId --> (cliente)

Figura 5.3: Base de Dados de Exemplo.

```
<reservas>
{for $c in table(’cliente’)
 where $c/clienteid = 1
 return
   <cliente clienteid=’{$c/clienteid/text()}’>
            <nome>{$c/nome/text()}</nome>
            <filmes>
            {for $r in table(’reserva’),$f in table(’filme’)
             where $r/filmeid = $f/filmeid and $r/clienteid = $c/clienteid
             return
                <filme>
                   <filmeid>{$f/filmeid/text()}</filmeid>
                   <titulo>{$f/titulo/text()}</titulo>
                   <data>{$r/data/text()}</data>
                </filme>
       }
          </filmes>
   </cliente>
}
</reservas>
```

Figura 5.4: Definição em UXQuery da visão XML de reservas de filmes.

ao banco de dados antes do cliente, com sua reserva de filmes, enviar sua visão XML. Nesse caso há um conjunto de operações diferente do apresentado anteriormente, na Figura 5.7. Com mudanças no banco de dados, há risco de que ocorra conflitos. A Figura 5.12 apresenta a saída gerada neste contexto com possibilidade de conflito.

Note que o primeiro conflito detectado impede que a reserva do filme “Senhor dos Anéis” seja feita, uma vez que o título do filme foi trocado na base de dados. Conforme a regra 1, apresentada na seção 4.2, o cliente pode ter usado como parâmetro o título do filme (que não é uma chave) para fazer sua reserva. Ao saber que o título real do filme é outro, sua decisão quanto à reserva pode mudar, portando o sistema o avisa de tal mudança. O segundo conflito ocorre por uma violação da regra 3 (seção 4.2). O filme que já estava reservado foi excluído do sistema, portanto não há como efetuar as modificações

```
<?xml version="1.0" encoding="UTF-8"?>
<reservas viewid="2">
   <cliente clienteid="1">
      <nome>André Vargas</nome>
      <filmes>
         <filme>
            <filmeid>1</filmeid>
            <titulo>Senhor dos Anéis</titulo>
            <data>2007-04-12</data>
         </filme>
         <filme>
            <filmeid>2</filmeid>
            <titulo>Os Outros</titulo>
            <data>2007-04-13</data>
         </filme>
         <filme>
            <filmeid>7</filmeid>
            <titulo>Tratamento de Choque</titulo>
            <data>2007-04-13</data>
         </filme>
      </filmes>
   </cliente>
</reservas>
```

Figura 5.5: Visão XML original da consulta de reservas de filmes.

```
<?xml version="1.0" encoding="UTF-8"?>
<reservas viewid="2">
   <cliente clienteid="1">
      <nome>André Vargas</nome>
      <filmes>
         <filme>
            <filmeid>1</filmeid>
            <titulo>Senhor dos Anéis</titulo>
            <data>2007-04-15</data>
         </filme>
         <filme>
            <filmeid>2</filmeid>
            <titulo>Os Outros</titulo>
            <data>2007-04-15</data>
         </filme>
       </filmes>
   </cliente>
</reservas>
```

Figura 5.6: Visão XML com as reservas de filmes, atualizada pelo cliente.

```
Não ocorreram atualizações na base de dados. Não há conflito
---------------Operações feitas pelo Cliente-------------------
t = {modify}
ref = {/reservas/cliente[@clienteid="1"]/filmes/filme[filmeid="1"]/data}
delta = {2007-04-15}

t = {modify}
ref = {/reservas/cliente[@clienteid="1"]/filmes/filme[filmeid="2"]/data}
delta = {2007-04-15}

t = {delete}
ref = {/reservas/cliente[@clienteid="1"]/filmes/filme[filmeid="7"]}
delta = { }
```

Figura 5.7: Operações detectadas pelo sistema, e notificação sobre conflitos.

na reserva. A detecção desta alteração é informada ao usuário.

5.2 Considerações Finais

Este capítulo apresenta os resultados da implementação das técnicas de detecção de conflito. Nos casos apresentados, mostrou-se que as técnicas podem ser usadas obteram

```
<catalogo>
{for $c in table('categoria')
 where $c/categoriaid = 5
 return
   <categoria categoriaid='{$c/categoriaid/text()}'>
            <nome>{$c/nome_categoria/text()}</nome>
            <filmes>
            {for $f in table('filme')
             where $c/categoriaid = $f/categoriaid
             return
                <filme>
                   <filmeid>{$f/filmeid/text()}</filmeid>
                   <titulo>{$f/titulo/text()}</titulo>
                </filme>
       }
          </filmes>
   </categoria>
}
</catalogo>
```

Figura 5.8: Definição em UXQuery da visão XML de catálogo de filmes de suspense.

```
<?xml version="1.0" encoding="UTF-8"?>
<catalogo>
   <categoria categoriaid="5">
      <nome>Suspense</nome>
      <filmes>
         <filme>
            <filmeid>2</filmeid>
            <titulo>Os Outros</titulo>
         </filme>
         <filme>
            <filmeid>3</filmeid>
            <titulo>Sexto Sentido</titulo>
         </filme>
      </filmes>
   </categoria>
</catalogo>
```

Figura 5.9: Visão XML original da consulta de catálogo de filmes.

```
<?xml version="1.0" encoding="UTF-8"?>
<catalogo>
   <categoria categoriaid="5">
      <nome>Suspense</nome>
      <filmes>
         <filme>
            <filmeid>3</filmeid>
            <titulo>Sexto Sentido</titulo>
         </filme>
      </filmes>
   </categoria>
</catalogo>
```

Figura 5.10: Visão XML de catálogo de filmes, atualizada pelo usuário.

```
//Modificar o nome do filme
UPDATE filme
SET titulo = "Senhor dos Anéis - Retorno do Rei"
WHERE filmeId = 1;
```

Figura 5.11: Atualizações feitas sobre a base de dados da locadora.

bons resultados mesmo com operações concorrentes, com visões XML formadas por estruturas diferentes e operações diretas no banco de dados.

No exemplo, deve-se destacar que a implementação é capaz de detectar conflitos e dependências para atualizações mesmo quando as transações concorrentes são efetuadas através de visões XML com estruturas diferentes ou através de outras aplicações que atuam sobre a base de dados, mantendo a consistência dos dados. A detecção das diferen-

ças, através do X-Diff, também mostrou adequada.

```
---------------Operações detectadas em Updated-------------------
t = {modify}
ref = {/reservas/cliente[@clienteid="1"]/filmes/filme[filmeid="1"]/data}
delta = {2007-04-15}

t = {modify}
ref = {/reservas/cliente[@clienteid="1"]/filmes/filme[filmeid="2"]/data}
delta = {2007-04-15}

t = {delete}
ref = {/reservas/cliente[@clienteid="1"]/filmes/filme[filmeid="7"]}
delta = { }

---------------Operações detectadas em Linha---------------------
t = {modify}
ref = {/reservas/cliente[@clienteid="1"]/filmes/filme[filmeid="1"]/titulo}
delta = {Senhor dos Anéis - Retorno do Rei}

t = {delete}
ref = {/reservas/cliente[@clienteid="1"]/filmes/filme[filmeid="2"]}
delta = { }

-----------------------------------------------------------------
---------------Conflitos-----------------------------------------

Operacao do Usuario
      t = {modify}
      ref = {/reservas/cliente[@clienteid="1"]/filmes/filme[filmeid="1"]/data}
      delta = {2007-04-15}
entra em conflito com a operacao da base de dados
      t = {modify}
      ref = {/reservas/cliente[@clienteid="1"]/filmes/filme[filmeid="1"]/titulo}
      delta = {Senhor dos Anéis - Retorno do Rei}
pela regra 1

Operacao do Usuario
      t = {modify}
      ref = {/reservas/cliente[@clienteid="1"]/filmes/filme[filmeid="2"]/data}
      delta = {2007-04-15}
entra em conflito com a operacao da base de dados
      t = {delete}
      ref = {/reservas/cliente[@clienteid="1"]/filmes/filme[filmeid="2"]}
      delta = { }
pela regra 3
```

Figura 5.12: Operações detectadas pelo sistema, e notificação sobre conflitos.

6 CONCLUSÃO

Este trabalho apresentou uma arquitetura para um sistema de exportação e importação de visões XML, tratando os problemas relacionados a este tipo de sistema. Neste sistema as visões XML são enviadas ao usuário, que pode alterá-las em seu próprio dispositivo (geralmente um dispositivo móvel). O usuário altera sua visão XML, estando desconectado da base de dados central. Somente após fazer suas alterações, o usuário envia sua visão XML atualizada. Através da visão atualizada o sistema detecta as alterações que devem ser feitas na base de dados.

O trabalho foi proposto para ser usado como extensão do sistema PATAXÓ, embora possa ser usado em outros contextos. Em relação ao sistema PATAXÓ, as principais vantagens do uso dessa arquitetura são:

- O usuário não precisa conhecer nenhuma linguagem específica para submeter suas alterações na visão XML. O usuário também não precisa ter em seu dispositivo uma aplicação específica para lidar com as visões. Tal característica torna muito maior o campo de aplicação do sistema, visto que ele pode usar aplicações legadas ou mesmo não utilizar aplicações especializadas para edição da visão XML.

- Enquanto o usuário está de posse da visão XML, outros usuários podem acessar e alterar a base de dados. Estas alterações podem ser feitas tanto por outras visões XML quanto por operações SQL. Dessa forma a utilização deste sistema tende a ser mais prática nas aplicações reais onde vários usuários podem acessar os mesmos dados.

O diferencial desta proposta está na utilização de trabalhos nas áreas de detecção de *deltas* em XML, atualização de visões XML, versões em bases de dados e tratamento de conflitos e uni-los no desenvolvimento de uma única arquitetura. Unindo abordagens nestas três áreas pôde-se criar uma arquitetura multiusuário e sem necessidade de aplicações específicas no cliente.

O primeiro problema tratado foi a detecção das atualizações feitas pelo usuário. Para tal, adotou-se a estratégia de armazenar a visão original, aquela que é entregue ao usuário, e compará-la com a que o usuário retorna, chamada de visão atualizada. Dessa forma o problema se resumiu em detectar a diferença (*delta*) entre dois documentos XML. Neste trabalho foi feito um estudo sobre as diversas técnicas de detecção de *delta*. Diferente de outros trabalhos na literatura, este teve como foco apresentar os diversos conceitos envolvidos nestes algoritmos. O comparativo entre eles foi focalizado principalmente nos aspectos importantes para este tipo de aplicação, onde o objetivo não é gerar um outro documento XML, mas detectar as alterações realizadas.

O algoritmo adotado para esta arquitetura, o X-Diff (WANG, 2003), foi escolhido principalmente por ter boa qualidade em sua saída, encontrando o *delta* de menor custo, conforme apresentado na seção 2.2.5 e por possuir um conjunto de operações equivalente ao utilizado neste trabalho.

Algumas características tiveram que ser trabalhadas no X-Diff, sendo um dos trabalhos realizados as adaptações deste algoritmo. A principal delas foi a transformação do formato de saída para um equivalente ao utilizado pelo PATAXÓ. Enquanto o X-Diff possui uma saída na forma de instruções inseridas no documento XML, o PATAXÓ utiliza instruções em uma linguagem própria e com identificação XPath.

A principal contribuição deste trabalho está no tratamento de conflitos baseados em regras de dependência. Nesta abordagem, são identificadas relações de dependência baseando-se na estrutura da visão XML. De forma diferente da encontrada na literatura, as regras de dependência não são baseadas em registros ou na transação, mas em manter a consistência do objeto como um todo.

A atomicidade e o isolamento das transações é definido através das regras de conflito. Uma configuração mais restritiva determina que qualquer conflito na visão XML anula toda a transação, ou seja, possui atomicidade e isolamento elevados. Uma outra configuração mais relaxada pode permitir que, dentro de uma transação, alterações concorrentes sobre o mesmo objeto sejam executadas, desde que mantidos os nodos que são chaves na base de dados relacional. Esta última possui níves mais baixos de atomicidade e isolamento pois parte da transação pode ser executada e parte ser abortada. Além destas configurações foi criado um modo intermediário, mais adaptado ao modelo de visões XML. Nessa configuração, as operações podem ser efetuadas dependendo do tipo da operação e da posição em que o nodo se encontra na estrutura da visão XML. A tabela 4.1 apresenta resumidamente estas regras.

A utilização de regras de dependência baseadas na estrutura da visão XML torna a abordagem mais prática e mais abrangente, visto que é menos dependente da semântica dos dados que outras abordagens, como a construção de funções de compensação, apresentadas na seção 2.3.2.

6.1 Publicações

Duas publicações foram geradas a partir deste trabalho:

1. VARGAS, A.; BRAGANHOLO, V.; HEUSER, C. Conflict Resolution and Delta Detection in Updates through XML views DataX - Second International Workshop on Database Technologies for Handling XML Information on the Web, realizado em conjunto com EDBT 2006, Munich, Alemanha. O artigo também foi publicado em LNCS (Lecture Notes in Computer Science - 2006) (VARGAS, 2006)

2. VARGAS, A.; BRAGANHOLO, V.; HEUSER, C. Suporte a Transações Longas em Atualizações Através de Visões XML. IV WTDBD - IV WorkShop de Teses e Dissertações em Bancos de Dados, realizado em conjunto com SBBD 2005, Uberlândia, MG, Brasil. (VARGAS, 2005)

Também foi desenvolvido um relatório técnico. Este é uma versão mais detalhada dos artigos, contendo maiores explicações sobre a detecção de diferenças e tratamento de conflitos:

- VARGAS, A.; BRAGANHOLO, V.; HEUSER, C. Conflict Resolution and Difference Detection in Updates through XML Views. Technical Report RP-352. PPGC, UFRGS. Porto Alegre, RS, Brazil. 2004. (VARGAS, 2005)

6.2 Implementação

Foi implementado um protótipo com as técnicas desenvolvidas neste trabalho integrado com o PATAXÓ e o algoritmo de detecção X-Diff. O protótipo foi desenvolvido na linguagem de programação Java (SUN, 2002). A escolha desta linguagem se deu pelas suas características como suporte para orientação a objetos e portabilidade, além de possuir um abrangente conjunto de funcionalidades para lidar com documentos XML. A integração também foi facilitada, uma vez que tanto o PATAXÓ quanto o X-Diff possuem implementações em Java.

A construção do programa segue a mesma estrutura da arquitetura apresentada no Capítulo 3. O repositório temporário (local onde são armazenadas as visões XML, as definições UXQuery e as *query trees*) são atualmente armazenadas em diretórios específicos. Cada arquivo recebe como nome o número de sua transação. Porém a implementação permite que o repositório temporário seja uma base de dados em um SGBD.

O tratamento de conflito fica em um único módulo que faz as operações passarem por uma série de testes de conflito. As tarefas de leitura e manipulação dos documentos XML são implementadas utilizando as funcionalidades do pacote Xerxes (APACHE, 2005) para Java.

6.3 Trabalhos Futuros

Existem algumas propostas para trabalhos futuros, que são apresentados a seguir:

Desenvolvimento de Novas Regras Personalizadas para o Tratamento de Conflitos Este trabalho apresenta três níveis de relaxamento para o tratamento de conflitos. Porém, tais regras podem não satisfazer por completo muitas aplicações. Uma proposta seria criar módulos para tratamento de conflitos, que poderiam ser configurados por um administrador da base de dados. Os módulos poderiam dessa forma contemplar exceções e questões próprias de um determinado banco de dados. Novos conjuntos de regras para identificação de conflitos poderiam considerar tipos de dados e níveis de usuários ao permitir, ou não, a execução de uma operação.

Implementação da Arquitetura na forma de um Serviço Web Neste trabalho a arquitetura foi implementada como protótipo a fim de verificar a sua funcionalidade. Porém, esta proposta visa a implementação de um sistema completo que poderia ser utilizado na forma de serviços Web (BOOTH, 2004), por exemplo. Tal software poderia ser direcionado para aplicações práticas nos campos de computação móvel e bancos de dados. A utilização da tecnologia de serviços Web aumentaria a interoperabilidade da aplicação com SGBDs e principalmente com os dispositivos clientes.

REFERÊNCIAS

APACHE. **Xerces java parser**. 2005. Disponível em: <http://xerces.apache.org/xerces-j/>. Acesso em: 12 mar. 2007.

BERESON, H. et al. A critique of ansi isolation levels. **ACM SIGMOD Record,** New York, v.24, n.2, May 1995. Trabalho apresentado na ACM SIGMOD International Conference on Management of Data, SIGMOD, 1995.

BOAG, S. et al. **XQuery** 1.0: an XML query language. Disponível em: http://www.w3.org/TR/2005/WD-xquery-20050404> Acesso em: 10 mar. 2007.

BOOTH, D.; HAAS, H.; MCCABE, F. **Web Services Architecture**. 2004. Disponível em: ws-arch/http://www3.org/TR/ ws-arch/. Acesso em: 10 mar. 2007.

BRAGANHOLO, V**. From XML to Relational View Updates**: applying old solutions to solve a new problem. 2004. Tese (Doutorado em Ciência da Computação) - Instituto de Informática, UFRGS, Porto Alegre, RS, Brasil.

BRAGANHOLO, V.; DAVIDSON, S.; HEUSER, C. Pataxó: a framework to allow updates through xml views. **ACM Transactions on Database Systems, TODS**, New York, v. 31, n. 3, p. 839-486, Sept. 2006.

BRAGANHOLO, V.; DAVIDSON, S. B.; HEUSER, C. A. On the Updatability of XML Views over Relational Databases. In: INTERNATIONAL WORKSHOP ON THE WEB AND DATABASES, 6., 2003, San Diego. **Proceedings…** New York: ACM, 2003. p. 31–36.

BRAGANHOLO, V.; DAVIDSON, S. B.; HEUSER, C. A. Uxquery: building updatable XML views over relational databases. In: SIMPÓSIO BRASILEIRO DE BANCO DE DADOS, SBBD, 18., 2003, Manaus. **Anais…** Belo Horizonte: DCC/UFMG, 2003. p. 26–40.

BRAGANHOLO, V.; DAVIDSON, S. B.; HEUSER, C. A. From xml view updates to relational view updates: old solutions to a new problem. In: INTERNATIONAL CONFERENCE ON VERY LARGE DATA BASES**,** VLDB, 30., 2004, Toronto, Canadá. **Proceedings...** [S.l.: s.n.], 2004. p. 276–287.

BRAGANHOLO, V.; DAVIDSON, S. B.; HEUSER, C. A**. Propagating XML View Updates to a Relational Database**. Porto Alegre, RS: Instituto de Informática da UFRGS, 2004. (TR-341).

CAREY, M. J. et al. Xperanto: Middleware for publishing object-relational data as XML documents. In: INTERNATIONAL CONFERENCE ON VERY LARGE DATA BASES, VLDB. Cairo, Egypt. **Proceedings…** San Francisco: Morgan Kaufmann, 2000. p. 646–648.

CAREY, M. J.; MUHANNA, W. A. The performance of multiversion concurrency control algorithms. **ACM Transactions on Computer Systems (TOCS),** New York, NY, USA, v. 4, n. 4, p. 338–378, 1986.

CEDERQVIST, P. e. a. **CVS - Concurrent Versions System.** [S.l.], 2003. Disponível em: <https://www.cvshome.org/docs/manual/cvs-1.11.20/cvs.html>. Acesso em: 10 mar. 2007.

CHAWATHE, S. S.; GARCIA-MOLINA, H. Meaningful change detection in structured data. **SIGMOD Record**, New York, v.26, n.2, p.26-37, June 1997. Trabalho apresentado na ACM SIGMOD International Conference on Management of Data, 1997, Tucson, USA.

CHAWATHE, S. S. et al. Change detection in hierarchically structured information. **ACM SIGMOD, Record**, New York, v.25, n.2, p.493-504, June 1996. Trabalho apresentado na ACM SIGMOD International conference on Management of Data, SIGMOD, 1996.

CHIEN, S.-Y.; TSOTRAS, V.; ZANIOLO, C. Version Management of XML Documents. In: INTERNATIONAL WORKSHOP ON THE WORLD WIDE WEB AND DATABASES, WEBDB, 3., 200, Dallas. **Selected Papers.** Berlin: Springer-Verlag, 2001. p.184-200. (Lecture Notes in Computer Science, v . 1997).

CLARK, J.; DEROSE, S. **Xml path language (XPath) version 1.0**. 1999. Disponível em : <http://www.w3.org/TR/xpath>. Acesso em: 10 mar. 2007.

COBENA, G. **Change management of semi-structured data on the Web.** 2003. Tese — Ecole Polytechnique, Paris, France.

COBENA, G.; ABDESSALEM, T.; HINNACH, Y. **A Comparative Study for XML Change Detection.** [S.l.], 2002. (Technical Report 221). Disponível em : <ftp://ftp.inria.fr/INRIA/Projects/verso/VersoReport-221.pdf>. Acesso em: 10 mar. 2007.

COBENA, G.; ABITEBOUL, S.; MARIAN, A. Detecting changes in XML documents. In: INTERNATIONAL CONFERENCE ON DATA ENGINEERING, ICDE, San Jose, California. **Proceedings…** [S.l.]: IEEE Computer Society, 2002. p. 41–52.

COLLINS-SUSSMAN, B.; FITZPATRICK, B.; PILATO, M. **Version Control with Subversion.** 2nd ed. Disponível em : <http://svnbook.red-bean.com/en/1.2/svnbook. html>. Acesso em: 10 mar. 2007.

CURBERA, F.; EPSTEIN, D. Fast difference and update of XML documents. In: XTech. **Proceedings…** San Jose, California: [s.n.], 1999.

DAYAL, U.; BERNSTEIN, P. A. On the correct translation of update operations on relational views**. ACM Trans. Database Syst**., New York, USA, v. 7, n. 3, p. 381–416, 1982.

DELTAXML. **Software de detecção de diferenças deltaxml**. Disponível em: <http://www.deltaxml.com/>. Acesso em: 10 mar. 2007.

DEWITT, D. J.; WANG, Y.; CAI, J.-Y. **X-diff**: An fast change detection algorithm for XML documents - tech report. 2003. Disponível em : <http://citeseer.ist.psu.edu/cache/papers/cs/22518/ http:zSzzSzwww.cs.wisc.eduzSzniagarazSzpaperszSzxdiff.ps.gz/wang03xdiff.ps.gz>. Acesso em: 10 mar. 2007.

DUBINKO, M. et al. **Xforms 1.0**. 2003. Disponível em : <http://www.w3.org/TR/2003/RECxforms- 20031014/>. Acesso em: 10 mar. 2007.

ELMASRI, R.; NAVATHE, S. **Fundamentals of Database Systems**. 4th ed. [S.l.]: Addison Wesley, 2004.

FERNANDEZ, M. et al. **Xquery 1.0 and xpath 2.0 data model**. 2004. Disponível em: <http://www.w3.org/TR/2004/WD-xpath-datamodel-20040723>. Acesso em: 10 mar. 2007.

FERNÁNDEZ, M.; TAN, W.-C.; SUCIU, D. Silkroute: Trading between relations and xml. **Computer Networks**, Amsterdam, v.33, n.1, p.723, 2000.

FONTAINE, R. L. A delta format for xml: Identifying changes in xml files and representing the changes in xml. In: XML Europe, 2001. **Proceedings...** Berlin, Germany: [s.n.], 2001.

FOUNDATION, F. S. **Gnu diff**. 2007. Disponível em : <http://www.gnu.org/software/diffutils/diffutils.html>. Acesso em: 10 mar. 2007.

GARCIA-MOLINA, H.; CHAWATHE, S. S. **Meaningful change detection in structureddata - extendi**. 1997. Disponível em : <http://www.cs.umaine.edu/ chaw/pubs/mhdiff-ext1.ps>. Acesso em: 10 mar. 2007.

GARCIA-MOLINA, H.; SALEM, K. Sagas. In: ACM SIGMOD, 1987. **Proceedings…** [S.l.]: ACM Press, 1987. p. 249–259.

COBENA, G.; ABDESSALEM, T.; HINNACH, Y. **A comparative study for xml change detection**. 2002. Disponível em: <citeseer.ist.psu.edu/696350.html>. Acesso em: 10 mar. 2007.

HEGARET, P. L.; WHITMER, R.; WOOD, L. **Document object model (dom)**. 2006. Disponível em: <http://www.w3.org/DOM/>. Acesso em: 10 mar. 2007.

KHAN, L.; WANG, L.; RAO, Y. Change detection of xml documents using signaturess. In: INTERNATIONAL WORKSHOP - REAL WORLD RDF AND SEMANTIC WEB APPLICATIONS, 2002, Honolulu, Hawaii, USA. **Proceedings…** San Francisco: Morgan Kaufmann, 2002.

KLIEB, L. Distributed disconnected databases. In: ACM SYMPOSIUM ON APPLIED COMPUTING, SAC, 1996, New York, USA. **Proceedings…** New York: ACM Press, 1996. p. 322–326.

LEONARDI, E.; BHOWMICK, S. S. Detecting changes on unordered xml documents using relational databases: a schema-conscious approach. In: ACM INTERNATIONAL CONFERENCE ON INFORMATION AND KNOWLEDGE MANAGEMENT, CIKM, 2005, New York, USA. **Proceedings…** New York: ACM Press, 2005. p. 509–516.

LEONARDI, E.; BHOWMICK, S. S. Oxone: A Scalable solution for detecting superior quality deltas on ordered large xml documents. In: INTERNATIONAL CONFERENCE ON CONCEPTUAL MODELING, 25., 2006. **Conceptual Modeling - ER 2006**: proceedings. Berlin: Springer, 2006. p. 196 – 211. (Lecture Notes in Computer Science, v. 4215).

LEONARDI, E.; BHOWMICK, S. S. Xandy: A scalable change detection technique for ordered xml documents using relational databases. **Data Knowl. Eng**., Amsterdam, v. 59, n. 2, p. 476–507, 2006.

MARIAN, A. et al. Change-centric management of versions in an xml warehouse. In: INTERNATIONAL CONFERENCE ON VERY LARGE DATABASES, VLDB, 27., 2001, Roma, Italy. **Proceedings...** [S.l.: s.n.], 2001. p. 581–590.

MOUAT, A. **Diffxml software**. 2002. Disponível em: <http://diffxml.sourceforge.net>. Acesso em: 10 de mar. 2007.

ÖZSU, M. T.; VALDURIEZ, P. **Principles of distributed databases systems.** [S.l.]: Prentice Hall, 1999.

PETERS, L. Change detection in xml trees: a survey. In: TWENTE STUDENT CONFERENCE ON IT, 3., 2005. **Proceedings…** Enschede: University of Twente, Faculty of Electrical Engineering, Mathematics and Computer Science, 2005.

PHATAK, S. H.; BADRINATH, B. R. Conflict resolution and reconciliation in disconnected databases. In: INTERNATIONAL WORKSHOP ON DATABASE & EXPERT SYSTEMS APPLICATIONS, DEXA, 1999. **Proceedings…** Los Alamitos: IEEE Computer Society, 1999.

POSTGRESQL. **Postgresql documentation**. 2007. Disponível em : <http://www.postgresql.org/documentation>. Acesso em: 10 mar. 2007.

SATYANARAYANAN, M. et al. Coda: A highly available file system for a distributed workstation environment. **IEEE Trans. Computers**, New York, v. 39, n. 4, p. 447–459, 1990.

SUBRAMANIAN, S. N. et al. Xperanto: Publishing object-relational data as xml. In: INTERNATIONAL WORKSHOP ON THE WEB AND DATABASES, WEBDB, Dallas, Texas. **Proceedings…** [S.l.:s.n.], 2000. p. 105–110.

SUN. **Java technology**. 2002. Disponível em: <http://java.sun.com/>. Acesso em: 12 mar. 2007.

TERRY, D. B. et al. Managing update conflicts in bayou, a weakly connected replicated storage system. In: SYMPOSIUM ON OPERATING SYSTEMS PRINCIPLES, 1995. **Proceedings…** [S.l.: s.n.], 1995. p. 172–183.

ULLMAN, J. D.; WIDOM, J.; GARCIA-MOLINA, H. **Database System The Complete Book**. [S.l.]: Prentice Hall, 2002.

VARGAS, A.; BRAGANHOLO, V.; HEUSER, C. **Conflict Resolution and Difference Detection in Updates through XML Views**. 2005. RP-352. Disponível em: <www.cos.ufrj.br/ vanessa/artigos/RT-352.pdf>. Acesso em: 10 mar. 2007.

VARGAS, A.; BRAGANHOLO, V.; HEUSER, C. Conflict resolution and delta detection in updates through xml views. In: INTERNATIONAL WORKSHOP ON DATABASE TECHNOLOGIES FOR HANDLING XML INFORMATION ON THE WEB, 2., 2006, Munich, Germany. **Current Trend in Database Technology – EDBT, 2006**: Revised selected papers. Berlin: Springer – Verlag, 2006. (Lecture Notes in Computer Science; 4254).

VARGAS, A.; HEUSER, C. A.; BRAGANHOLO, V. Suporte a transações longas em atualizações através de visões xml. In: WORKSHOP DE TESES E DISSERTAÇÕES EM BANCOS DE DADOS, WTDBD, 4., 2005. **Anais...** Uberlândia, MG, Brasil: [s.n.], 2005.

W3C: World wide web consortium. 2004. Disponível em : <http://www.w3.org/>. Acesso em: 25 mar. 2007.

WANG, Y.; DEWITT, D. J.; CAI, J.-Y. X-diff: An effective change detection algorithm for XML documents. In: INTERNATIONAL CONFERENCE ON DATA ENGINEERING, ICDE, 2003. Bangalore, India. **Proceedings…** [S.l.]: IEEE Computer Society, 2003. p. 519–530.

XYLEME, L. Xyleme: A dynamic warehouse for xml data of the web. In: INTERNATIONAL DATABASE ENGINEERING & APPLICATIONS SYMPOSIUM, IDEAS, 2001, Grenoble, France. **Proceedings…** [S.l.]: IEEE Computer Society, 2001. p. 3–7.