ALDO MONTEIRO DO NASCIMENTO

UM MODELO PARA INTEGRAÇÃO DE DOCUMENTOS XML EM NÍVEL DE INSTÂNCIA

Dissertação apresentada como requisito parcial à obtenção do grau de Mestre em Inform'atica. Programa de Pós-Graduação em Informática, Setor de Ciências Exatas, Universidade Federal do Paraná.
Orientadora: Profa. Carmem Satie Hara

CURITIBA

2008

SUMÁRIO

LISTA DE FIGURAS

RESUMO

Um repositório integrado de dados é um repositório de dados provenientes de diversas fontes. Na construção de um repositório integrado há dois grandes problemas para agrupar instâncias: ambiguidade na identificação de entidades e conflito de valor de atributos. Nesta dissertação é proposto um modelo de dados que facilita a resolução de conflitos de valor de atributos representando-os explicitamente na estrutura integrada. Neste modelo, o repositório integrado é uma árvore XML gerada a partir de dados importados de uma ou mais fontes de dados XML, e os nodos são anotados com informações de proveniência. Essas anotações têm dois propósitos. Primeiro, elas representam a origem de cada elemento no repositório integrado. Esta informação é essencial para determinar a qualidade e confiança que podem ser atribuídas aos dados. Segundo, elas permitem que a porção da árvore XML oriunda da fonte de dados e armazenada no repositório integrado seja reconstruída. Essa capacidade é importante para a comparação do documento original com novas versões da mesma fonte possibilitando a atualização da base de dados local. Algoritmos para instanciar o repositório integrado de acordo com o modelo proposto e reconstruir a fonte de dados são apresentados nesta dissertação. Resultados de um estudo experimental conduzido para determinar o impacto das anotações no tamanho do repositório integrado, bem como o desempenho dos algoritmos propostos são também discutidos.

ABSTRACT

A datawarehouse is a repository of data imported from different sources. There are two major problems for merging instances from different sources in order to build a datawarehouse: entity identification ambiguity and attribute value conflict. In this dissertation we propose a data model that facilitates the resolution of value attribute conflicts by explicitly representing them in the integrated schema. In this model, the datawarehouse is an XML tree populated with data imported from one or more XML sources, and nodes are annotated with provenance information. The purpose of annotations are twofold: first, they represent the origin of every element in the datawarehouse. This information is essential for determining the quality and amount of trust one places on the data. Second, they allow the portion of source XML tree used to populate the warehouse to be reconstructed. This capability is important if one needs the original document to compare with new releases from the same source in order to update the local database. Algorithms for populating the warehouse according to the proposed model and for reconstructing the source data are presented. We also present results from an experimental study conducted to determine the impact of the annotations on the size of the warehouse and the performance of the proposed algorithms.

CAPÍTULO 1

INTRODUÇÃO

A forma como é realizada a publicação de dados através da Web está em evolução constante. Inicialmente, ela era baseada em HTML, que adota uma abordagem de documentos hiper-texto. Mais recentemente, ela passou a ser baseada em XML [Bray et al. 1998], promovendo uma abordagem mais focada nos dados. XML tem se tornado o padrão para a representação e troca de dados na Internet e, sendo um formato de dados semi-estruturado, também pode ser usado para a integração de dados. XML também tem se tornado a opção em sistemas de gerenciamento de dados e documentos, devido à sua capacidade de representar dados irregulares enquanto mantém a estrutura dos dados, caso ela exista [Sawires et al. 2005]. A flexibilidade do formato XML permite que documentos sejam estendidos ou reduzidos para descrever conteúdo de qualquer tamanho, até mesmo quando a estrutura do documento muda.

A criação de repositórios integrados de dados que armazenam documentos XML nativos é uma consequência do avanço da utilização do formato XML. Um repositório integrado de dados é gerado a partir dos dados existentes em diversas fontes de dados [Pokorný 2002]. Os repositórios integrados aumentam a flexibilidade das fontes de dados, adaptando os dados ao usuário ou às necessidades das aplicações que os utilizam. Tais repositórios são freqüentemente materializados para aumentar a velocidade das consultas quando os dados de origem estão remotamente localizados, ou o tempo de resposta às consultas é de natureza crítica. Um repositório é dito materializado quando ele é armazenado em uma base de dados, ao invés de ser computado de suas fontes de dados no momento em que consultas são realizadas. Combinar essas tecnologias, repositório integrado de dados e XML, é um passo na direção de se obter a flexibilidade que uma aplicação de integração de dados necessita.

Repositórios integrados de dados têm sido criados em diversos ramos do conhecimento.

Pode-se citar como um exemplo um projeto da área biológica. O processo de criação de uma base de dados relevante a algum campo de estudo envolve a transformação, integração e limpeza dos erros de dados de múltiplas fontes de dados externas, assim como a adição de novo material e anotações. Por exemplo, o EpoDB [Davidson e Liefke 2001] é uma base de dados criada no Centro de Bioinformática da Universidade da Pensilvânia. O EpoDB foi projetado para estudar a regulação de genes durante a diferenciação e o desenvolvimento de células de vertebrados com sangue vermelho. Para a construção do EpoDB, dados pertinentes às células do sangue vermelho foram extraídas de outras bases dados. Uma vez extraídos, os erros foram retirados dos dados através de uma abordagem semi-automatizada. Com os dados já sem erros, foi feita a integração entre eles e informações adicionais e anotações foram incluídas manualmente ou automaticamente à visão integrada.

O EpoDB é um exemplo típico de um repositório integrado de dados. A criação e manutenção dessas visões de bases de dados geram algumas questões:

1. Como especificar e implementar a transformação e integração das fontes de dados para a visão materializada?

2. Como garantir a consistência dos dados no processo de atualização?

3. Como localizar a origem ou proveniência dos dados armazenados no repositório integrado de dados?

O primeiro problema, que é transformar e integrar as fontes de dados para produzir a visão dos dados, vem sendo objeto de estudo de muita pesquisa na comunidade de bancos de dados. O problema torna-se complexo pelo fato das fontes de dados serem armazenadas em diferentes modelos de dados ou formatos de dados tais como o formato ASN.1, o formato texto, o modelo relacional, dentre outros. Soluções para esse problema incluem: a criação de bases de dados de ligação [Etzold e Argos 1993]; utilização de tecnologia de bancos de dados relacionais comerciais; adoção de uma visão complexa dos dados orientada a objetos, como nos sistemas Kleisli [Wong 2000], Garlic [Haas 1990], ou abordagens de integração CORBA; utilização de uma estratégia de integração semi-estruturada, como no sistema Tsimmis [Papakonstantinou et al. 1995, Garcia-Molina et al. 1997], ou propostas

com XML [Abiteboul 1999, Suciu 1999].

O segundo problema, que trata da consistência dos dados no processo de integração de instâncias de fontes de dados diferentes envolve dois grandes problemas [Prabhakar et al. 1993]: ambiguidade na identificação de entidades e conflito de valores de atributos. Estes são problemas que na literatura recente têm sido tratados como *data cleaning* [Rahm e Do 2000]. Identificação de entidade refere-se ao problema de identificar dados que se sobrepõem em diferentes fontes de dados. Esse assunto é tratado em diversos trabalhos que utilizam o modelo de dados relacional [Lim et al. 1996], entidade-relacionamento [Menestrina et al. 2006] e XML [Poggi e Abiteboul 2005]. Conflito de valor de atributo refere-se ao problema de duas ou mais fontes de dados conterem informações da mesma entidade ou atributo, mas com valores conflitantes. Algumas abordagens existentes para o tratamento desse problema incluem *data profiling*, mineração de dados e técnicas baseadas em restrições [Prabhakar et al. 1993, Rahm e Do 2000].

O último problema, que é localizar a origem ou proveniência dos dados tem sido objeto de muito estudo na literatura [Cui e Widom 2000, Woodruff e Stonebraker 1997, Lee et al. 1998, Bernstein e Bergstraesser 1999]. Informações sobre proveniência constituem a prova de corretude dos dados e determina a qualidade e confiança que se deposita sobre os mesmos [Tan 2007].

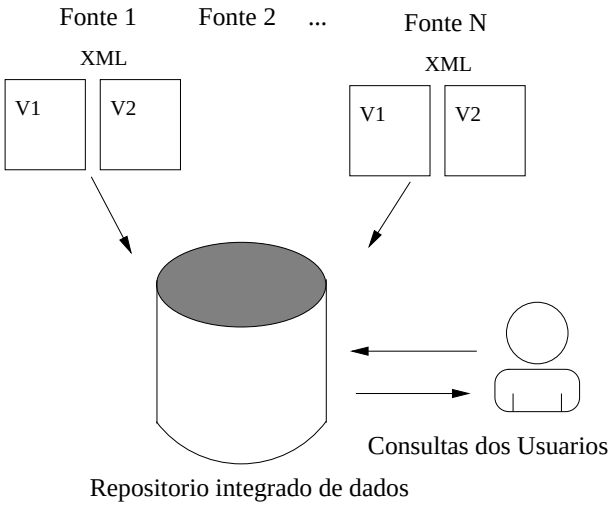


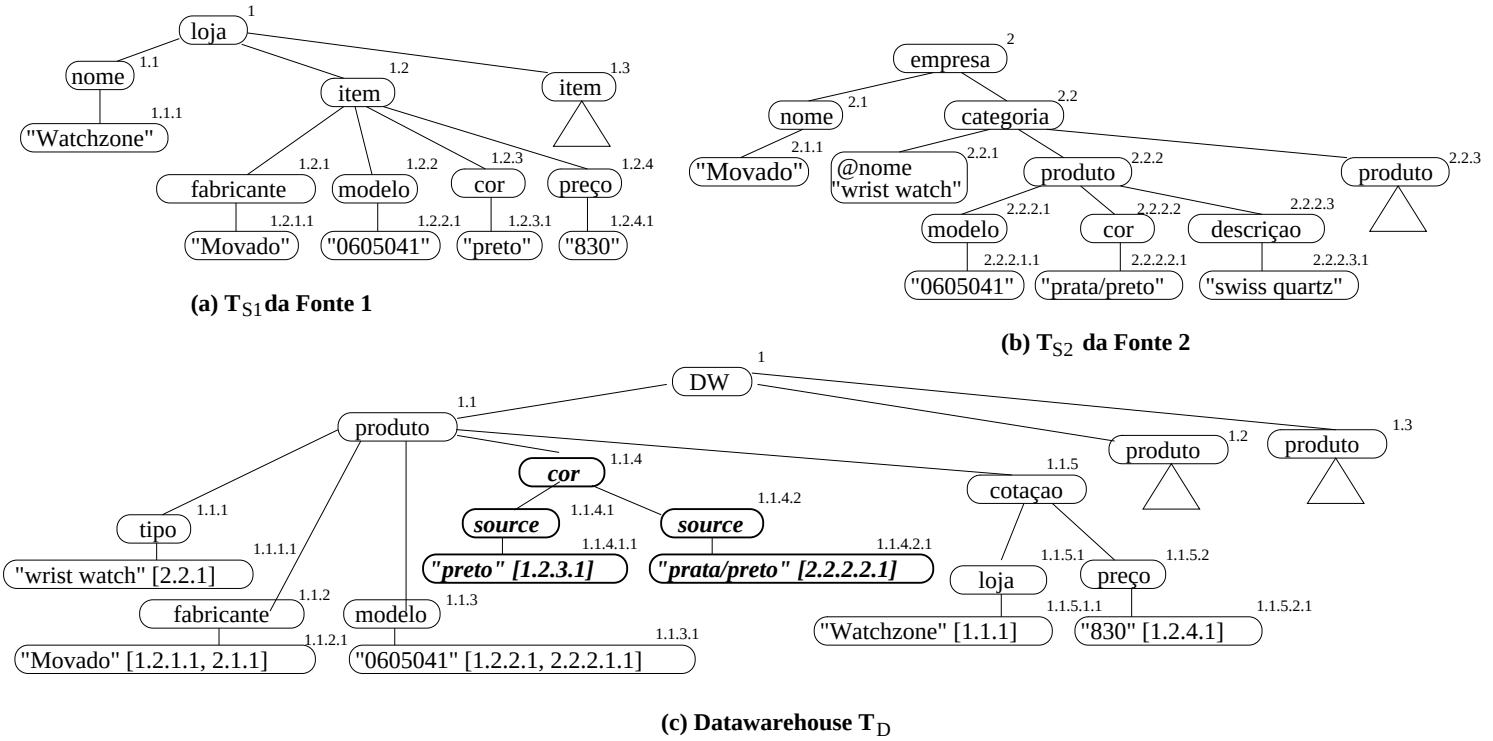
Figura 1.1: Estrutura de um repositório integrado de dados.

Uma visão geral do problema é ilustrada na Figura 1.1. Um repositório integrado

de dados armazena documentos XML oriundos de diversas fontes de dados, que podem ser desde bancos de dados até *sites* na Internet. Os dados passam por uma filtragem e limpeza de erros antes de serem armazenados. Quando novas versões dos dados estiverem disponíveis elas são lidas da fonte de dados para que seja possível identificar as alterações que devem ser materializadas. Freqüentemente, as fontes de dados que atualizam o repositório integrado de dados possuem informações semelhantes, ou até mesmo, fontes de dados distintas possuem as mesmas informações. Nesse caso, o sistema deve ter condições de identificar quando informações oriundas de diversas fontes de dados são semanticamente as mesmas e automaticamente integrá-las ao repositório de dados. A habilidade de automatizar o processo de atualização é muito importante porque o tamanho do repositório integrado de dados pode ser muito grande, dificultando qualquer processo que necessite de grande interferência manual.

Neste trabalho é proposto um modelo de dados que facilita a resolução de conflitos de valor de atributo através da representação explícita desses conflitos no esquema integrado. Como um exemplo, considere duas fontes de dados no domínio de produtos, ilustradas nas Figuras 1.2(a) e (b). O documento identificado como *Fonte 1* contém informações sobre produtos obtidas numa loja *on-line*, enquanto a *Fonte 2* contém informações providas por um fabricante. Um mapeamento da *Fonte 1* para o repositório integrado de dados com a estrutura detalhada na Figura 1.2(c) define que o nome da loja é mapeado para a loja de uma cotação do produto; um item vendido pela loja é mapeado para um produto, e seus sub-elementos fabricante, modelo e cor são mapeados para sub-elementos de produto; o preço de um item, por sua vez, é colocado sob um elemento cotação. O mapeamento da *Fonte 2* para o repositório integrado de dados é definido de forma similar. Dado que no repositório integrado de dados é definido que sempre que dois produtos possuem o mesmo valor de fabricante e modelo, eles se referem à mesma entidade no mundo real, a árvore resultante é dada pela Figura 1.2(c). Neste trabalho é adotada a definição de chaves proposta em [Buneman et al. 2001] para definir como elementos das fontes de dados são combinados com os elementos do repositório integrado de dados.

Observe que as fontes de dados não possuem o mesmo valor de cor do produto. Esta



(a) T_{S1} da Fonte 1

(b) T_{S2} da Fonte 2

(c) Datawarehouse T_D

Figura 1.2: Integração de duas fontes de dados.

inconsistência de valores é representada através da criação de dois elementos distintos para cada valor, como está destacado na Figura 1.2. O objetivo é facilitar o processo de resolução do problema de conflito de atributo. Conflitos identificados podem então ser eliminados por um processo de limpeza de dados.

Os elementos do repositório integrado de dados são anotados com informação de proveniência. Os propósitos das anotações são dois: primeiro, a possibilidade de conseguir determinar a origem de cada elemento no repositório integrado de dados. Essa informação é essencial para determinar a qualidade e quantidade de confiança que pode-se atribuir aos dados [Tan 2007] e pode ser usado como um parâmetro para resolver um conflito. No exemplo da Figura 1.2, pode-se presumir que a cor do produto fornecida pelo fabricante está correta e, desta forma, dar prioridade a este valor em detrimento daquele dado por uma loja. Segundo, a possibilidade de reconstruir a porção da árvore XML usada para inserir dados no repositório integrado. Esta reconstrução é importante para automatizar o processo de atualização do repositório integrado. Mais especificamente, dada a nova versão do documento e a versão reconstruída, é possível executar um algoritmo de detecção de diferenças para determinar quais são as atualizações a serem aplicadas ao

repositório integrado de dados para mantê-lo atualizado.

Nas próximas seções são descritas brevemente a motivação e as contribuições desta dissertação e é definida a organização do trabalho.

1.1 Motivação

A principal motivação desta dissertação é propor um modelo para integração de documentos XML que integre diversas abordagens existentes na literatura, de modo a solucionar problemas que tais abordagens isoladamente não tratam. A proposta desta dissertação utiliza uma estratégia em que os dados dos documentos XML são fisicamente armazenados num repositório integrado de dados após um processo de integração dos dados (Seção 2.3). É utilizada uma abordagem de identificação de entidades que determina quando dados devem se sobrepor ou não (Seção 2.4). Finalmente, para que se possa localizar dados de uma determinada fonte no repositório integrado de dados, utiliza-se uma estratégia de anotação de proveniência dos dados em conjunto com um esquema de endereçamento dos elementos na sua fonte de origem (Seção 2.7).

1.2 Contribuições

Esta dissertação está inserida no contexto de atualizações automatizadas de repositórios integrados de dados. Neste contexto, as principais contribuições do trabalho são:

- Um modelo de dados em que fontes de dados com esquemas distintos atualizam um repositório integrado de dados com esquema fixo. Uma linguagem de mapeamento e algoritmos que realizam essa tarefa também são apresentados.

- Uma estrutura que permite a representação explícita de inconsistências oriundas do processo de integração de fontes de dados distintas.

- Um algoritmo de reconstrução de fontes de dados armazenadas no repositório integrado de dados que dispensa o armazenamento da fonte de dados original para futuras comparações com novas versões.

- Um estudo experimental que quantifica a quantidade de espaço de armazenamento que o modelo de dados desta dissertação consegue economizar em relação a outras abordagens tradicionais e outro que mostra o desempenho melhorado do tempo de criação e atualização de um repositório integrado de dados com a utilização do conceito de chaves XML [Buneman et al. 2001] e reestruturação das fontes de dados.

- A publicação de um artigo em um evento científico [Nascimento e Hara 2008].

1.3 Organização do Trabalho

O restante do trabalho está organizado da seguinte maneira. O Capítulo 2 apresenta alguns problemas relacionados a este trabalho, que são questões relacionadas à integração de instâncias em documentos XML, uma estratégia de identificação de entidades, que são as chaves XML, e um trabalho que utiliza, entre outras abordagens, as chaves XML num contexto semelhante a esse trabalho. Além disso são abordados alguns aspectos e estratégias sobre limpeza de dados, proveniência de dados e endereçamento de elementos. O Capítulo 3 apresenta o modelo de dados deste trabalho, abordando cada um dos componentes que integram o modelo. Além disso, são apresentados os algoritmos utilizados para a integração de documentos XML. O Capítulo 4 apresenta um estudo experimental sobre a quantidade de espaço de armazenamento e o desempenho do algoritmo na presença de diferentes tamanhos de documentos XML. E, finalmente, o Capítulo 5 conclui esta dissertação e apresenta algumas sugestões de trabalhos futuros.

CAPÍTULO 2

PROBLEMAS RELACIONADOS

Neste capitulo são apresentadas algumas abordagens existentes na literatura para problemas que são relacionados ao modelo proposto nesta dissertação. A Seção 2.1 apresenta a terminologia utilizada nesta dissertação. A Seção 2.2 apresenta uma breve introdução da linguagem XML. A Seção 2.3 apresenta estratégias de integração de instâncias das fontes de dados. São apresentadas as abordagens visão global e visão local, que privilegiam a atualidade dos dados e o desempenho na obtenção de dados através da consulta a base de dados integrada, respectivamente. Além disso são analisadas duas técnicas de integração de dados. Outro tópico abordado é a questão de identificação de entidades, apresentada na Seção 2.4, que é o problema de identificar dados que se referem à mesma entidade do mundo real para que possam ser agrupados. É estudada especificamente a abordagem de chaves XML. Na Seção 2.5 é apresentado um trabalho que utiliza as chaves XML para identificação de entidades e uma estratégia de representação de versões em um documento XML. A Seção 2.6 apresenta uma visão geral do problema de limpeza de dados. Na Seção 2.7 é apresentada uma análise de técnicas de arquivamento de proveniência de dados e três abordagens de endereçamento de elementos.

2.1 Terminologia

Nesta dissertação, o termo repositório integrado de dados é utilizado para definir um sistema de armazenamento de dados que pode ser atualizado por diversas fontes de dados. O sistema de armazenamento prevê a integração dos dados dessas diversas fontes, através de técnicas de identificação de entidades. O termo fonte de dados refere-se a artefatos que possuem dados e que são utilizados apenas para leitura desses dados. O termo base de dados refere-se a outros sistemas de armazenamento que não integrem dados, mas que também não são utilizados exclusivamente para leitura. Um banco de dados relacional é

um exemplo de base de dados, pois os dados que ele armazena podem tanto ser consultados quanto atualizados. O termo documento XML é utilizado para referenciar fontes de dados que estejam originalmente no formato XML. No escopo deste trabalho, todo documento XML é uma fonte de dados. No entanto, nem toda fonte de dados é necessariamente um documento XML.

2.2 A Linguagem XML

A Linguagem XML [Bray et al. 1998], um subconjunto da Linguagem de Marcação Padrão Generalizada (SGML - *Standard Generalized Markup Language*) [ISO 2004], foi criada em 1996 e recomendada pelo W3C (*World Wide Web Consortium*) [W3C 1994] em 1998. O objetivo de seus projetistas era descrever uma linguagem de marcação flexível, escalonável e adaptável, indo diretamente de encontro às limitações que a Linguagem de Marcação para Hipertexto (HTML - *Hipertext Markup Language*) [Ragget et al. 1999] possui frente a constante expansão da Web.

A linguagem XML tornou-se um padrão para a troca de dados heterogêneos (semi-estruturados) pela Web, devido principalmente ao fato de fazer uma distinção entre os três principais elementos constituintes de um documento da Web: estrutura, conteúdo e apresentação. A estrutura diz respeito às regras que determinam se o documento está bem formado. O conteúdo são os dados de negócio, ou seja, as informações que se quer enviar. A apresentação determina como o conteúdo deve ser disposto ao receptor do documento, a fim de que torne mais clara a sua legibilidade. A própria linguagem XML é responsável pelo conteúdo do documento. A parte de apresentação é tratada pela Linguagem de Estilo Extensível (XSL - *eXtensible Style Language*) [Clark 1999]. A estrutura ou esquema do documento pode ser definida utilizando a linguagem de Definição de Tipo de Documento (DTD - *Document Type Definition*) ou XML Schema [Pemberton et al. 2002, Fallside 2000], dentre outros. A HTML, ao contrário, não faz tal distinção, concatenando o conteúdo com a apresentação, e não há regras que invalidem um documento deste tipo.

Nesta dissertação, é proposto um modelo para um repositório integrado de dados que

armazena dados provenientes de documentos XML. Os documentos XML são armazenados obedecendo a estrutura do repositório integrado de dados. Além disso, documentos XML cujos dados já estão armazenados no repositório integrado de dados são reconstruídos através de informações de proveniência existentes no repositório.

2.3 Integração

Integração de dados é o problema de combinar dados existentes em diferentes fontes, provendo ao usuário uma visão , dentro de um esquema global, que é independente das fontes e pode ser consultado pelos usuários [Poggi e Abiteboul 2005]. Os sistemas de integração de dados que esta dissertação considera são caracterizados por uma arquitetura baseada em um esquema global e um conjunto de fontes. As fontes contêm os dados reais, enquanto o esquema global provê uma visão integrada dessas fontes, podendo ou não armazenar os dados. Modelar a relação entre as fontes e o esquema global é um aspecto crucial.

Duas abordagens básicas são possíveis para esse propósito. A primeira abordagem, chamada visão global [Lenzerini 2002], requer que o esquema global seja expresso em termos das fontes de dados. A segunda abordagem, chamada de visão local [Lenzerini 2002], requer que o esquema global seja especificado independentemente das fontes, e os relacionamentos entre o esquema global e as fontes sejam estabelecidos definindo que cada fonte seja uma visão sobre o esquema global.

A abordagem visão global também pode ser chamada de integração virtual. A integração virtual é definida por um ou mais `esquemas mediados` que não são usados para armazenar dados, mas apenas para consultá-los [Draper et al. 2001]. Um esquema mediado, segundo [Pottinger e Bernstein 2002], pode ser conseguido através do mapeamento de dois ou mais esquemas distintos, gerando um único esquema integrado que descreva todos os dados em ambos os esquemas de entrada. Quando uma consulta é submetida em uma solução de integração virtual, ela é traduzida em um conjunto de consultas submetidas às fontes de dados.

Esse processo de tradução das consultas é a principal desvantagem da abordagem

visão global. Dependendo do nível de reestruturação pelo qual a fonte de dados integrada tem que passar, o processo de tradução de consultas pode ser bastante dispendioso. Entretanto, através dessa abordagem, os dados retornados das consultas sempre são os mais recentes disponíveis nas fontes de dados. Um cenário adequado para a aplicação dessa abordagem é formado por fontes de dados que sofrem atualizações constantes e as consultas submetidas às bases de dados integradas não requerem um baixo tempo de resposta.

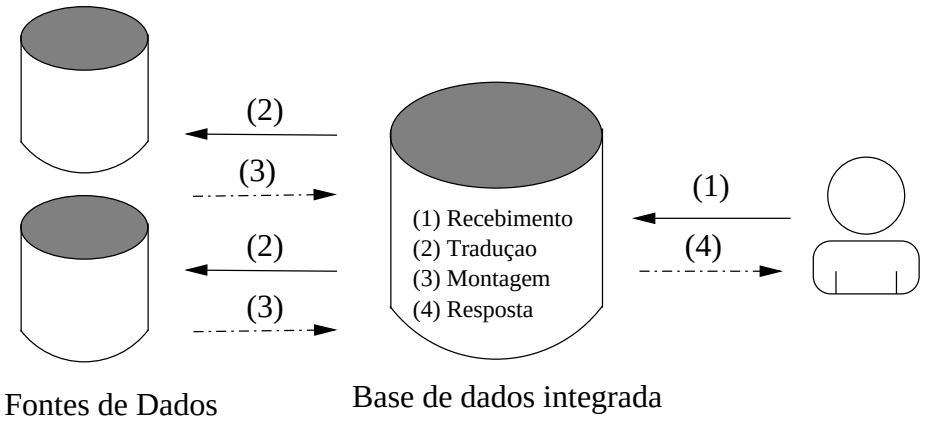


Figura 2.1: Ilustração da abordagem visão global.

A Figura 2.1 ilustra um processo de consulta a uma base de dados integrada virtualmente. Em (1) o usuário submete a consulta à base de dados integrada. Em (2) a base de dados traduz a consulta do usuário para as fontes corretas, (3) monta a resposta da consulta com base nos retornos das fontes de dados e (4) retorna o resultado ao usuário.

Todavia, uma solução de integração de dados tem como premissa a utilização da abordagem visão local, pois os dados são armazenados fisicamente no repositório integrado de dados. Essa abordagem deve prover funcionalidades que permitam o efetivo armazenamento dos dados da fonte de dados e funcionalidades que possam mapear os dados armazenados das fontes e atualizá-los.

A principal vantagem dessa abordagem é o desempenho do processamento de consultas. Visto que o repositório integrado de dados é independente das fontes de dados, as consultas são diretamente atendidas pelo repositório integrado de dados. As principais desvantagens são que os dados podem não estar totalmente atualizados e, nos casos em que o esquema do repositório integrado de dados é estático, torna-se mais difícil acomodar novas versões

de fontes de dados ou variações entre as fontes de dados existentes. Deve haver um compromisso para que a periodicidade do processo de atualização da base dados integrada seja realizada de tal forma que os dados disponibilizados aos usuários não se tornem demasiadamente antigos. Além disso, o esquema do repositório integrado de dados deve acompanhar a evolução do esquema das fontes de dados.

O processo de integração de documentos XML, que é o processo final na integração de esquemas, também pode ser classificada [la Fontaine 2002]. São identificadas duas abordagens: a integração de dois níveis e a integração de três níveis. A diferença entre essas formas depende da existência de dois documentos independentes a serem integrados de dois níveis ou se há também um documento *b*ase do qual os outros são derivados (de três níveis). A integração de três níveis possui o potencial de prover uma solução mais precisa quando um documento base existe.

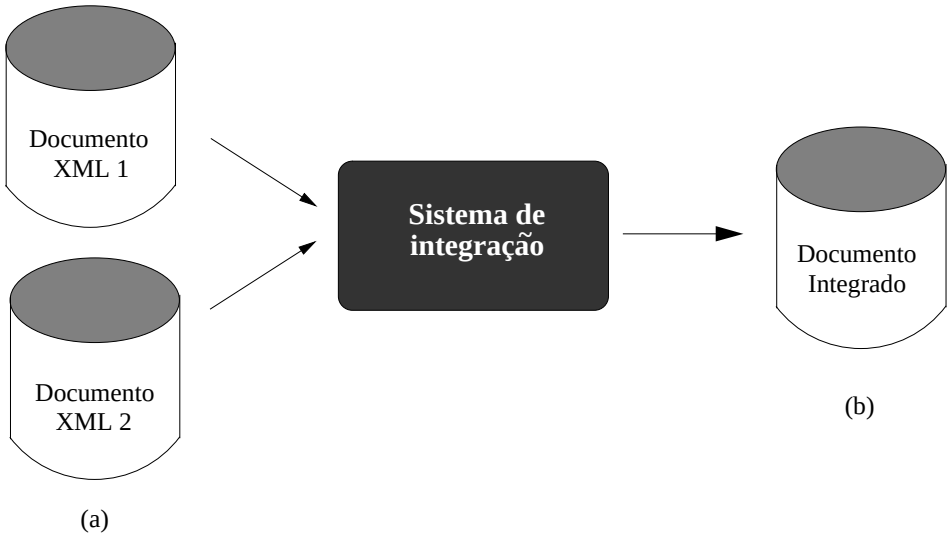


Figura 2.2: Ilustração do processo de integração de dois níveis.

A integração de dois níveis envia os documentos fonte diretamente para o sistema de integração que encarrega-se de realizar as transformações necessárias e gerar um documento resultante. A Figura 2.2 ilustra um exemplo do processo de integração de dois níveis. O documento integrado (Figura 2.2(b)) possui o conteúdo comum dos documentos 1 e 2 (Figura 2.2(a)) de forma compartilhada, e também o conteúdo adicional de cada um dos dois documentos. Para a realização da integração de dois níveis, os requisitos básicos podem ser estabelecidos de um modo informal: o documento integrado deve con-

ter todos os dados existentes nos documentos originais, sem duplicação quando houver sobreposição. A dificuldade está em definir quando essa sobreposição ocorre e garantir que ela seja tratada corretamente.

Os requisitos podem também ser definidos de uma maneira mais formal e especificamente para XML. Os requisitos básicos podem ser resumidos como a seguir:

- Os atributos de qualquer elemento devem ser a união dos atributos nos dois documentos para os elementos correspondentes. Quando houver conflitos, eles devem ser tratados.

- Os elementos filhos de um dado elemento devem ser a união ordenada dos elementos correspondentes nos dois documentos.

- Um algoritmo de comparação é necessário para identificar elementos correspondentes nos dois documentos em cada nível da estrutura da árvore. É desejável que essa correspondência possa ser controlável, por exemplo, utilizando-se uma chave que evite qualquer ambiguidade nas situações onde uma correspondência mais precisa é necessária.

- A integração deve, de preferência, ser capaz de administrar elementos ordenados e desordenados.

- Nos casos em que elementos texto sofrem mudanças, deve haver uma escolha entre executar uma integração dos conteúdos ou escolher uma ou outra das representações textuais.

- Conflitos devem ser identificados.

A integração de três níveis parte do princípio que existe um documento base que define o esquema para o qual as fontes de dados devem ser traduzidas. O processo consiste de uma tradução dos esquemas das fontes de dados para o esquema da base de dados final e integração dos dados nesse esquema. O resultado final é um documento no formato do documento *b*ase com os dados oriundos dos documentos fonte, que transformam-se

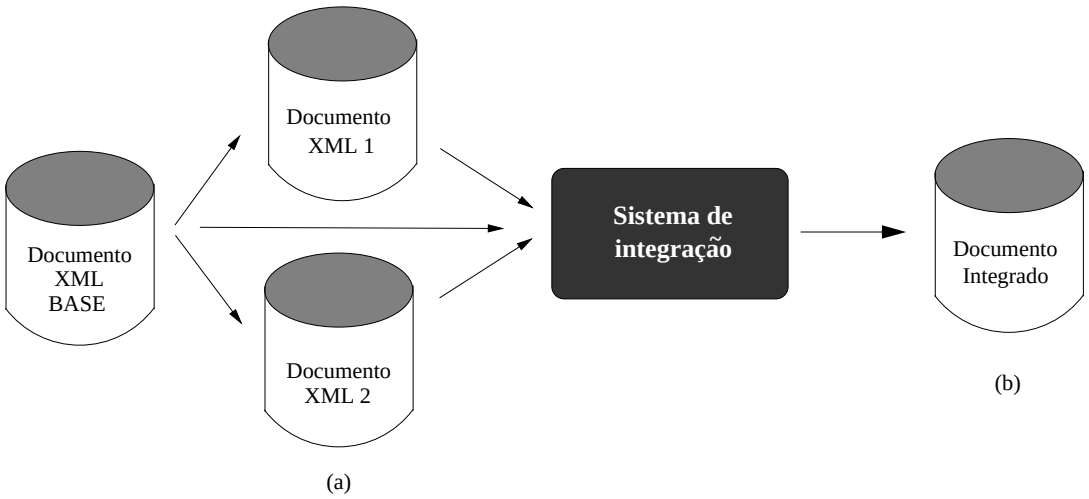


Figura 2.3: Ilustração do processo de integração de 3 níveis.

em ramificações do documento resultante. A Figura 2.3 ilustra o processo de integração da abordagem de 3 níveis. O sistema de integração precisa conhecer além das fontes, a estrutura do documento *b*ase para que possa gerar o documento integrado no formado correto. Requisitos adicionais podem ser identificados no caso da integração de três níveis, como a seguir:

- Toda mudança ocorrida em qualquer elemento em ambas as ramificações deve estar presente no documento integrado.
- Elementos excluídos em ambas as ramificações devem ser excluídos no documento integrado.
- Atributos excluídos em ambas as ramificações devem ser excluídos no documento integrado.

Os requisitos definidos para as abordagens de integração de dois níveis e de três níveis são os mesmos identificados na elaboração da proposta deste trabalho, detalhados no Capítulo 3. Todavia, a abordagem de integração de dois níveis está mais próxima ao modelo de integração desta dissertação pelo fato de a integração dos documentos ocorrer sem a interferência de uma terceira entidade, neste caso o documento *b*ase da abordagem de três níveis. Além disso, dentre os requisitos da abordagem de integração de dois níveis,

um deles requer que exista uma ferramenta de comparação que identifique entidades correspondentes. Na próxima seção o problema de identificação de entidades é discutido.

2.4 Identificação de Entidades

Quando dados oriundos de diferentes fontes de dados são integrados, é possível que vários dados das fontes refiram-se a mesma entidade do mundo real. *A Identificação de Entidades* considera o problema de identificar os dados que referem-se (ou provavelmente referem-se) a mesma entidade e como realizar a integração desses dados [Menestrina et al. 2006].

Um dado *combinado* é criado a partir de composição de dados de diversas fontes de dados. Em geral, um dado combinado precisa ser comparado e possivelmente integrado a outros dados, pois a composição da informação pode tornar possível identificar novos relacionamentos. O Exemplo 1 ilustra essa situação.

Exemplo 1 Suponha que um dado d_1 possui o nome e o número do RG de uma pessoa, enquanto o dado d_2 possui um endereço e o mesmo RG. É possível combinar d_1 e d_2 baseando-se no RG. Agora existe um nome e um endereço para essa pessoa, e essa informação combinada pode tornar possível conectar esse dado combinado com d_3, por exemplo, contendo um nome e endereço similares. Note que nem d_1 nem d_2 poderiam ser combinados com d_3 isoladamente, pois eles não continham a informação combinada que o dado combinado possui. □

A próxima seção apresenta uma estratégia para identificação de dados baseada em chaves XML. Esta estratégia permite identificar entidades do mundo real em fontes de dados distintas e, se for o caso, fazer a integração e complementação dos dados num repositório integrado de dados.

2.4.1 Chaves XML

Existem diversas propostas para a especificação de chaves XML. Entre elas pode-se citar a DTD [Pemberton et al. 2002] e XML Schema [Fallside 2000]. Em uma DTD, a idéia principal de chaves está baseada na utilização de atributos identificadores (ID). Com

esses atributos pode-se identificar unicamente um elemento dentro de um documento XML. Entretanto, a utilização de atributos ID tem efeito real muito mais próximo ao de "ponteiros" internos do que de chaves propriamente ditas. Por exemplo, atributos ID não têm escopo. Ao contrário de chaves, eles são únicos no documento todo e não dentro de um conjunto determinado de elementos. Um efeito prático dessa limitação é que, por exemplo, não se pode atribuir o mesmo CPF a um elemento estudante e a um elemento pessoa como chave. Além disso, a utilização de atributos ID como chaves significa que só é possível a criação de chaves unárias e jamais definidas sobre elementos (apenas sobre atributos). Finalmente, pode-se especificar, no máximo, um atributo ID para um tipo de elemento, enquanto que, na prática, pode-se precisar de mais de uma chave.

Já a proposta XML Schema foi criada de uma forma mais elaborada, especialmente com relação a chaves. Ela permite especificar chaves em termos de expressões XPath [Clark e DeRose 1999]. XPath dá suporte a uma variedade de eixos que permitem não apenas descer em uma árvore XML, mas também mover para seus ancestrais e irmãos. Além disso, é possível embutir predicados e até funções em XPath. Por exemplo, a expressão $/A/B[last()]/C/D/E/ancestor :: *$ seleciona todos os nodos ancestrais no caminho $A.B.C.D.E$ a partir da raiz. Observe que um predicado é especificado na expressão: B deve ser o último B filho de A. Com tal complexidade, questões como equivalência de expressões e inclusão de expressões também tornam-se complexas. Outra questão técnica diz respeito à igualdade de valores. XML Schema restringe igualdade a nodos texto. Porém existem outras possibilidades. Um exemplo é definir uma chave nome para elementos pessoa, pois nome pode ter uma estrutura mais complexa, formada por subelementos primeiro nome e último nome.

Para superar essas limitações, [Buneman et al. 2001] propuseram uma nova estrutura de chaves XML, definida em termos de uma ou mais expressões de caminho. Logo, podem envolver mais de um atributo, subelementos ou estruturas mais genéricas. Essas chaves podem ser definidas para um subconjunto dos dados (chaves relativas) ou para todo o documento (chaves absolutas). Esta especificação de chave é independente de DTD ou qualquer outra definição de esquema.

As características básicas dessa nova proposta de chaves XML consistem da definição de igualdade de nodos, expressões de caminho, definição de chaves, inferência de chaves e chaves relativas e absolutas.

Igualdade de nodos. A forma escolhida de igualdade é a igualdade de árvores. Um exemplo que justifica essa escolha é o caso de um elemento `nome` ser chave para pessoa. Esse elemento pode ser formado por uma cadeia de caracteres ou ainda por uma estrutura complexa de subelementos que podem compreender elementos `primeiro-nome` e `sobrenome`. Para implementar a igualdade de árvores, o valor de um nodo é especificado através de (1) um conjunto S de endereços relativos dos seus subnodos, (2) uma função parcial de S para nomes, e (3) uma função parcial de S para *strings*. Dois nodos n_1 e n_2 são de valor igual, denotado por $=_v$, se eles coincidirem nos valores de (1), (2) e (3). Com relação à representação textual de um elemento XML, esta definição estabelece que a ordem dos atributos não é importante para a definição de igualdade.

Expressões de caminho. A linguagem adotada para criar as expressões de caminho é um subconjunto de XPath e expressões regulares [Hopcroft e Ullman 1979]. Há a operação de concatenação P/Q, que é o resultado de seguir o caminho P e depois o caminho Q. Além disso, a sintaxe das expressões de caminho define o caminho vazio "ϵ" e o símbolo "//" para um caminho arbitrário. Uma expressão de caminho é simples se não contém o símbolo "//".

Definição de chaves. Para definir uma chave são necessárias duas características: um conjunto no qual está definida a chave e os valores que juntos identificam unicamente elementos no conjunto. Uma chave é um par $(Q, \{P_1, ..., P_n\})$ onde Q é uma expressão de caminho e $\{P_1, ..., P_n\}$ é um conjunto de expressões de caminho simples. A idéia é que a expressão de caminho Q identifica um conjunto de nodos, o qual é chamado de conjunto alvo, e o conjunto $P_1, ..., P_n$ são os caminhos chaves. Por exemplo, considere a seguinte definição de chave: $(pessoa/empregados, \{nome/primeironome, nome/ultimonome\})$

O caminho alvo $pessoa/empregados$ identifica um conjunto de nodos no documento. Este é o conjunto alvo. Cada um desses nodos define uma sub-árvore com um rótulo `empregados` na raiz. Em cada sub-árvore são encontrados zero ou mais caminhos chave

nome/primeironome e zero ou mais caminhos chave *nome/ultimonome*. Dois nodos n_1 e n_2 no conjunto alvo são distintos se eles se diferenciarem em algum dos nodos alcançáveis através do caminho chave *nome/primeironome* ou eles se diferenciarem em algum dos nodos alcançáveis através do caminho chave *nome/ultimonome*.

Inferência de chaves. O problema de inferência de chaves XML é tratado em [Buneman et al. 2002]. Um conjunto de regras de inferências I é definido tal que para qualquer conjunto de chaves Σ e uma única chave φ, é possível determinar se Σ deriva φ, representado por $\Sigma \models \varphi$ se φ pode ser obtido de Σ aplicando regras em I. Alguns exemplos de regras de inferência são listados abaixo:

(1) Se (Q, S) é uma chave e $S \subset S'$, então (Q, S') também é chave.

(2) Se $(Q_1/Q_2, \{P\})$ é uma chave então $(Q_1, \{Q_2/P\})$ também é chave.

Chaves relativas. Para descrever as estruturas de chaves hierárquicas é introduzida a noção de chaves relativas. Chaves relativas consistem do par (Q, K) onde Q é uma expressão de caminho e K é uma chave. Utiliza-se a notação $n[\![P]\!]_T$ para representar o conjunto de nodos (endereços de nodos) alcançáveis a partir de n e seguindo o caminho P na árvore T. Algumas vezes é usada a abreviação $[\![P]\!]_T$ para representar $raiz[\![P]\!]_T$, ou seja, o conjunto de nodos alcançáveis a partir do nodo raiz seguindo o caminho P. Um documento satisfaz uma chave relativa $(Q_1, (Q_2, S))$ se, e somente se, para todos os nodos n em $[\![Q_1]\!]_T$, n satisfaz a chave (Q_2, S). Em outras palavras, (Q, K) é uma chave relativa se K é uma chave para toda sub-árvore com raiz em um nodo em $[\![Q_1]\!]_T$. Por exemplo:

- $(pais/estado/cidade, (bairro, \{nome\}))$. Esta chave define que o nome de um bairro o identifica unicamente dentro de uma cidade; porém, podem existir bairros com o mesmo nome em cidades distintas.

- $(rua(casa, \{numero\}))$. O número de uma casa a identifica unicamente em uma rua. Se houver somente um nodo rua abaixo da raiz, esta chave equivale a $(rua/casa, \{numero\})$.

Nesta dissertação a técnica utilizada para identificação de entidades é baseada em chaves XML, utilizando as principais características abordadas nesta seção. Na próxima seção é apresentado um trabalho que utiliza as chaves XML como ferramenta de identificação de entidades, para exemplificar um contexto diverso em que essa abordagem é

aplicável.

2.5 Utilização de Chaves em Contextos Diversos

Em [Buneman et al. 2004] é proposta uma arquitetura para armazenamento de versões de um documento XML que sofre atualizações. Através de um estudo sobre bases de dados científicas, foi definido um modelo de dados que, segundo os autores, atende aos principais requisitos que esse tipo de base de dados exige. O sistema proposto arquiva múltiplas versões de dados hierárquicos em um documento compactado através do uso de três técnicas principais, que são (1) identificação de mudanças baseada em chaves, (2) representação combinada de versões baseada em chaves e (3) herança de rótulos de tempo.

A definição de chaves utilizadas pelos autores é a mesma utilizada neste trabalho e apresentada na Seção 2.4.1. Entretanto, foram definidas algumas restrições adicionais. A primeira delas determina que toda chave definida em um nodo é relativa ao seu pai e que chaves são definidas até uma determinada profundidade da árvore. A restrição de que chaves definidas em um nodo devem ser definidas em relação ao pai desse nodo utiliza a noção das chaves de inserção amigável [Buneman et al. 2001], que também é adotada nesta dissertação.

Exemplo 2 Considere o seguinte conjunto de chaves no domínio de uma universidade:

$(\epsilon, (universidade, \{nome\}))$
$(universidade, (departamento/funcionario, \{matricula\}))$

Suponha que um documento d possui esse conjunto de chaves definido. Para identificar um elemento `funcionário` no documento d é necessário conhecer apenas o `nome` da `universidade` e a `matrícula` do `funcionário`. Entretanto, para incluir um novo `funcionário` no documento, é necessário determinar um `departamento` para o `funcionário`. No entanto, não há uma maneira de determinar o `departamento` e, desta forma, existem diversas maneiras de adicionar um `funcionário`. Assim, uma chave é dita de inserção amigável se for sempre possível inserir elementos em uma parte do documento que possua chave sem ambiguidade, especificando onde inserir o elemento através de chaves. □

Outra restrição importante toma por base os nodos de fronteira. Os nodos de fronteira são nodos que fazem parte de caminhos de fronteira. Um caminho de fronteira é um caminho que não faz parte do prefixo de qualquer outro caminho na árvore. A restrição é que todos os elementos que estejam acima dos elementos de fronteira possuam chaves. Ou seja, não pode existir nenhum nível da árvore que possua elemento sem chave, exceto os nodos folha. Caso existam elementos que não atendam essa restrição, o algoritmo proposto combina esses elementos com outros existentes em outras versões.

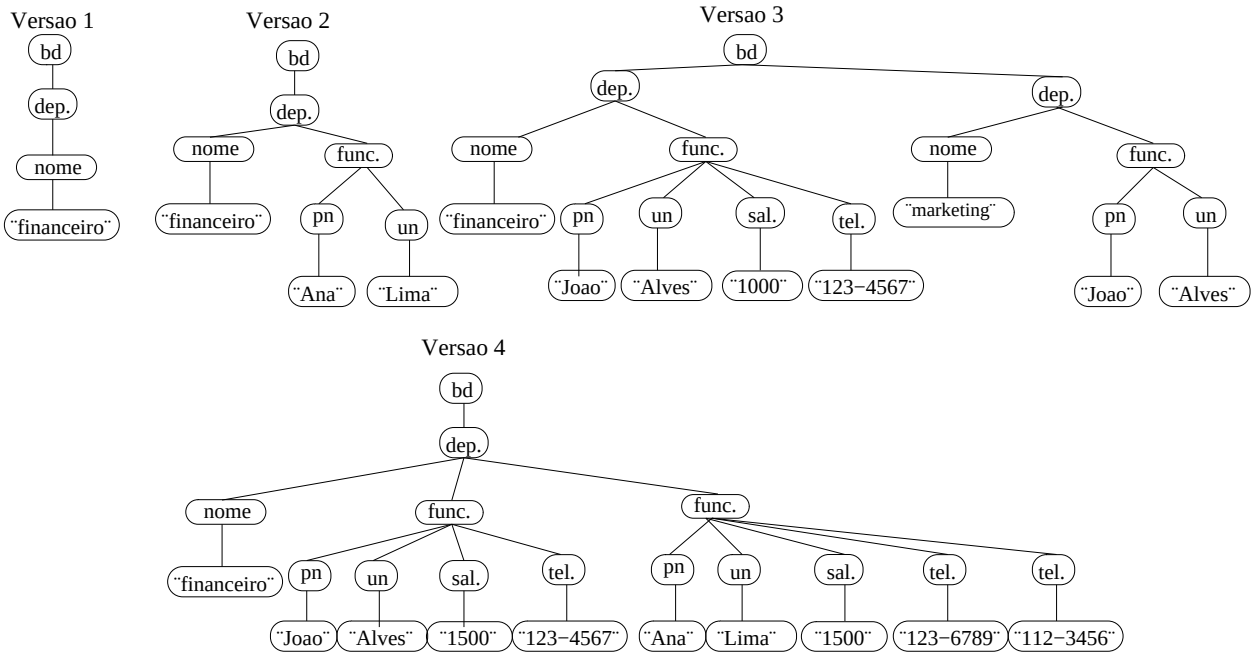


Figura 2.4: Uma seqüência de versões de um documento.

Para atender o requisito de guardar o histórico de versões do documento, é utilizada uma abordagem que utiliza herança de rótulos de tempo. Conceitualmente, um rótulo de tempo é armazenado em cada elemento para indicar a existência desse elemento em vários pontos no histórico. No modelo proposto, um rótulo de tempo é armazenado em um elemento filho somente se for diferente do rótulo de tempo do elemento pai. Partindo-se do princípio que a maior parte das mudanças são inserções de novos elementos, é grande a probabilidade que um elemento no documento possua o mesmo rótulo de tempo que seu elemento pai. Desta forma, uma quantidade considerável de espaço de armazenamento é economizada através da herança de rótulos de tempo.

A Figura 2.4 ilustra uma seqüência de versões de um documento de uma empresa

contendo informações sobre seus empregados em cada departamento. A Figura 2.5 ilustra a forma na qual as versões são representadas em um único documento através da anotação com rótulos de tempo. No nível mais alto, cada versão possui apenas um nodo raiz e eles são correspondentes. Os nodos existentes em mais de uma versão são representados uma única vez e o nodo é anotado com os números de versão dos documentos originais. O rótulo de tempo de cada nodo no documento resultante é passado como herança para suas respectivas sub-árvores. Este procedimento é chamado recursivamente para os filhos, até que as folhas sejam alcançadas. O nodo raiz no documento é usado para determinar se uma versão arquivada está vazia. Por exemplo, suponha que o documento está vazio na versão 5. Então, o nodo raiz terá o rótulo de tempo t=[1-5], enquanto o nodo db terá o rótulo de tempo t=[1-4], indicando que a versão 5 é um documento vazio.

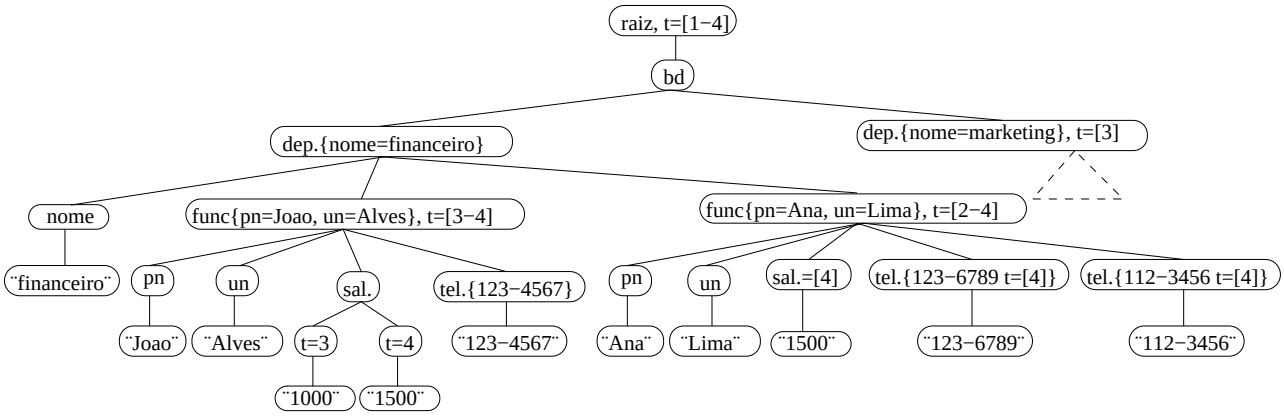


Figura 2.5: Rótulos de tempo anotados nos nodos arquivados.

Observe que no documento resultante da Figura 2.5, os nodos que aparecem em várias versões são armazenados uma única vez. Se um nodo ocorre na versão i, então o rótulo de tempo do nodo correspondente no documento contém i. Se um nodo não tem um rótulo de tempo, assume-se que seu rótulo de tempo é herdado do elemento pai desse nodo. Por exemplo, o nodo departamento financeiro herda o rótulo de tempo t=[1-4] do seu nodo pai db.

O sistema que implementa o modelo proposto é composto por dois módulos principais, conforme mostrado na Figura 2.6, que são: (1) anotação de chaves nos nodos e (2) integração de nodos. O módulo de anotação de chaves adiciona um novo atributo aos elementos contendo o valor da(s) chave(s) desse elemento, permitindo que o módulo de

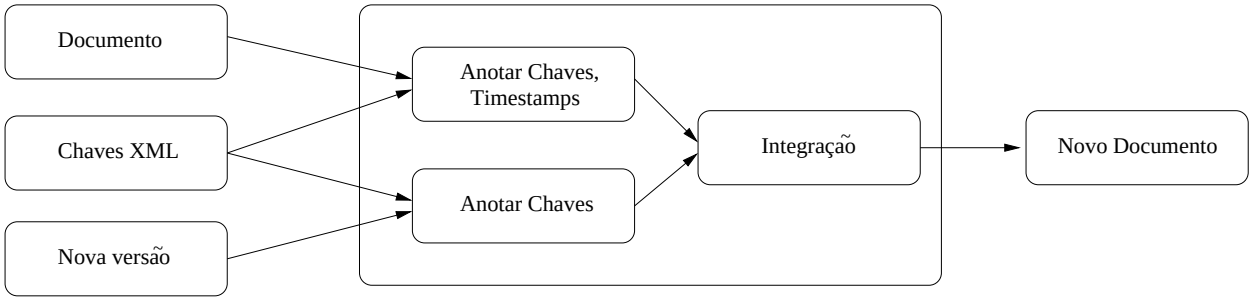


Figura 2.6: Módulos do sistema de arquivamento.

integração identifique imediatamente os elementos a serem combinados nas árvores.

Anotação de Chaves. Um exemplo do resultado do algoritmo de anotação de chaves são os elementos func{pn=João,un=Alves} e func{pn=Ana,un=Lima} na Figura 2.7, na qual os valores que fazem parte de chave do nodo func são os nodos pn e un. A idéia por trás desse algoritmo é varrer todo o documento à procura de nodos que possuam chaves e os nodos que contenham os valores dos caminhos-chave das respectivas chaves. Dada uma chave $(Q, (Q', \{P_1, .., P_k\}))$, um nodo que está no caminho Q/Q' é um nodo que possui chave e um nodo que está no caminho $Q/Q'/P_i$, onde $i \in [1, k]$, é um nodo que possui o valor do caminho-chave. Sempre que um nodo que contém o valor de um caminho-chave é encontrado, toda a sub-árvore desse nodo é memorizada e essa sub-árvore é armazenada no nodo que é identificado pela chave.

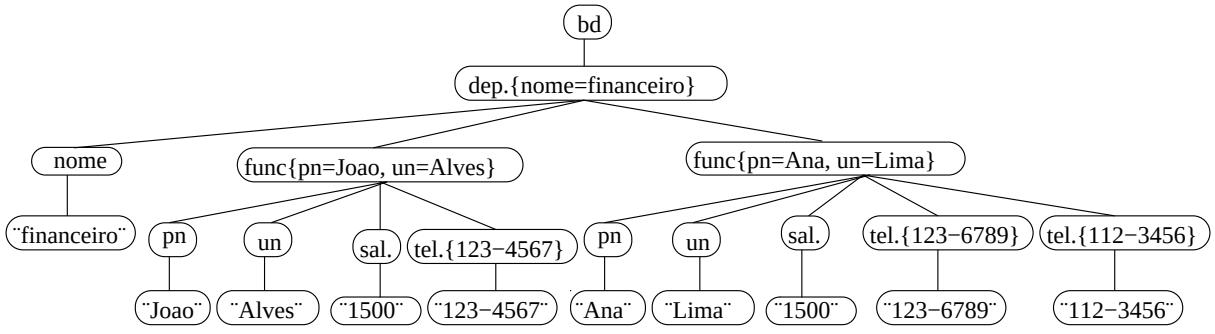


Figura 2.7: Exemplo de chaves anotadas nos elementos.

Algoritmo de Integração. A idéia por trás desse algoritmo é combinar recursivamente os nodos vindos da nova versão S com os nodos do documento armazenado A que possuam os mesmos valores de chave, iniciando pela raiz. Quando um nodo y de S é combinado com um nodo x de A, o rótulo de tempo de x é aumentado com i, o novo

número da versão. As sub-árvores dos nodos x e y são então recursivamente combinadas. Nodos em S que não possuam um correspondente em A são simplesmente adicionados a A com o novo número da versão como seu rótulo de tempo. Nodos em A que não mais existam na atual versão S terão seus rótulos de tempo terminados apropriadamente, isto é, esses nodos não contém o rótulo de tempo i.

Nesta dissertação, a forma com que as chaves XML são utilizadas é bastante semelhante à forma apresentada nesta seção. Uma restrição que é compartilhada é que as chaves devem ser de inserção amigável. Além dessa restrição, nesta dissertação é definida uma restrição adicional que diz que todos os níveis da árvore devem conter chaves. Entretanto, o processo de anotação dos valores dos elementos que possuem chave não é utilizado nesta dissertação, pois cria uma sobrecarga adicional que não representa um grande benefício no modelo de dados que esta dissertação propõe. Isto porque o processo de busca dos elementos que identificam as chaves ocorre num espaço reduzido (dentro de um único elemento) e a restrição de que todos os níveis devem possuir chaves faz com que todos os níveis de elementos sejam pesquisados.

2.6 Limpeza de Dados

O processo de limpeza de dados trata da detecção e remoção de erros e inconsistências dos dados a fim de melhorar sua qualidade. Existem diversas abordagens para limpeza de dados propostas na literatura. No contexto de integração de fontes de dados, o processo de limpeza já foi tratado de uma maneira simplista e direta, no qual é criada uma ordenação simples das fontes de dados que estão sendo integradas, seguida por uma fase de eliminação de inconsistências dando prioridade aos valores de acordo com a lista ordenada [Bitton e DeWitt 1983]. Entretanto, quando as fontes de dados envolvidas são heterogêneas, ou seja, não compartilham o mesmo esquema ou as entidades do mundo real são representadas de formas distintas, o problema de integração de dados se torna mais complexo.

O problema de integração quando as fontes de dados possuem esquemas diferentes tem sido extensivamente explorado na literatura e é conhecido como o problema de integração

de esquema [Batini et al. 1986]. Esta dissertação está inserida no contexto de integração de instâncias, que trata da representação heterogênea dos dados e suas implicações quando múltiplas fontes de dados são integradas.

O problema fundamental na integração de instâncias é que os dados fornecidos por várias fontes incluem identificadores e dados em formato texto, que não coincidem em todas as fontes de dados ou que apresentam erros devido a uma variedade de razões, incluindo erros tipográficos ou de transcrição [Hernandez et al. 1998].

A maioria das abordagens existentes assume que a representação de uma mesma entidade do mundo real em diferentes fontes de dados é única e facilmente identificável através da existência de uma chave comum. Entretanto, devido às diferentes implementações dos componentes de fontes de dados, este normalmente não é o caso. A resolução da ambiguidade na identificação de correspondências entre entidades normalmente não pode ser alcançada sem a disponibilidade e uso de informação adicional [Prabhakar et al. 1993].

Surgem assim problemas relacionados à qualidade dos dados, que determina, em linhas gerais, que quanto menos conflitos e inconsistências houver em uma fonte de dados, maior será a qualidade dos dados. Problemas de qualidade de dados estão presentes em bases de dados isoladas, tais como documentos e bancos de dados, por motivos diversos, como erros ortográficos durante entrada de dados e informações incompletas. Quando múltiplas fontes de dados precisam ser integradas, como em um repositório integrado de dados, a necessidade de limpeza de dados aumenta significativamente. Isto porque as fontes frequentemente contêm dados redundantes com diferentes representações. Para prover acesso a dados consistentes e corretos, a consolidação de diferentes representações de dados e eliminação de informações duplicadas torna-se necessária.

Uma abordagem de limpeza de dados deve satisfazer diversos requisitos. Inicialmente, ela deve detectar e remover todos os principais erros e inconsistências nas fontes de dados individuais. A abordagem deve ser auxiliada por ferramentas para limitar a inspeção manual e ser extensível para ser utilizada por fontes adicionais. Além disso, a limpeza de dados não deve ser realizada isoladamente, mas em conjunto com transformações de dados de esquemas baseadas em meta-dados compreensivos. Funções de mapeamento

para limpeza de dados e outras transformações de dados devem ser especificadas de forma declarativa e ser reusáveis para outras fontes de dados da mesma forma que o processamento de consultas. No caso de repositórios integrados de dados, uma infra-estrutura de regristro de fluxo de transformações deve existir para executar todos os passos de transformação de dados para múltiplas fontes e grandes conjuntos de dados de uma maneira eficiente e confiável.

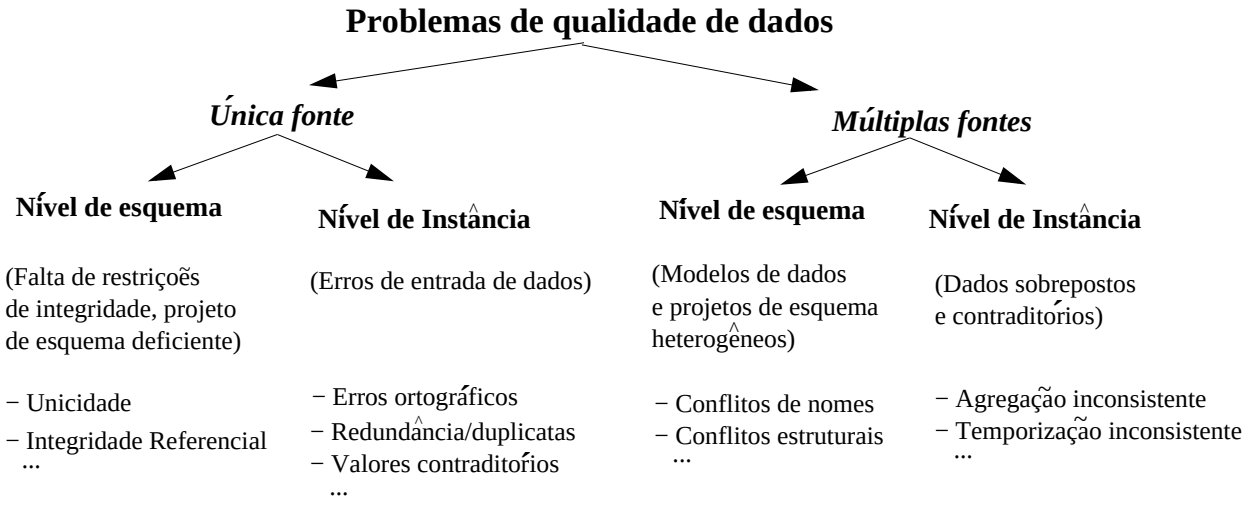


Figura 2.8: Classificação dos problemas de qualidade de dados em fontes de dados [Rahm e Do 2000].

A Figura 2.8 apresenta uma classificação dos problemas de qualidade de dados em uma única fonte e em múltiplas fontes de dados, que ocorrem no nível de esquema e de instância.

Problemas de fonte única. A qualidade dos dados de uma fonte depende muito da existência de uma definição de esquema e de restrições de integridade que controlam a validade dos seus dados. Para fontes sem esquema, como arquivos - estrutura essa que não possui todas as restrições existentes num banco de dados relacional, por exemplo - há poucas restrições sobre quais dados podem ser armazenados, aumentando a probabilidade de erros e inconsistências.

Sistemas de banco de dados, por outro lado, reforçam as restrições de um modelo de dados específico assim como restrições de integridade específicas da aplicação. Problemas de qualidade de dados relacionados ao esquema ocorrem pela falta de um modelo específico ou restrições de integridade. A baixa qualidade pode ser ocasionada por li-

mitações do modelo de dados, um projeto do esquema mal definido, ou pela falta de definição de restrições de integridade. Problemas específicos de instância referem-se aos erros e inconsistências que não podem ser prevenidos pelo esquema, como por exemplo, erros ortográficos.

Dado que limpar fontes de dados é um processo custoso, prevenir as inconsistências dos dados no momento da entrada é obviamente um passo importante para reduzir o problema da limpeza. Isto requer um projeto apropriado do esquema da base de dados e restrições de integridade de modo a vislumbrar essa questão. Também, a descoberta de regras de limpeza de dados durante o projeto do repositório integrado de dados pode sugerir melhoramentos às restrições impostas pelos esquemas existentes.

Problemas de múltiplas fontes. Os problemas presentes em fontes únicas são agravados quando múltiplas fontes precisam ser integradas. Cada fonte pode conter erros e os dados nas fontes podem ser representados de forma distinta, sobrepostos ou contraditórios. Isto porque as fontes são desenvolvidas, depuradas e mantidas independentemente para atender necessidades específicas. O resultado é um alto grau de heterogeneidade dos sistemas de gerenciamento de dados, modelos de dados, projetos de esquema e os dados em si.

No nível de esquema, diferenças entre o modelo de dados e o projeto do esquema são dirimidas pelos passos da tradução do esquema e integração do esquema, respectivamente. Os principais problemas com relação ao projeto do esquema são conflitos de nomes e estruturais. Conflitos de nome surgem quando o mesmo nome é usado por objetos diferentes (homônimos) ou nomes diferentes são usados para o mesmo objeto (sinônimos). Conflitos estruturais ocorrem em diversas formas e referem-se a representações diferentes do mesmo objeto em fontes distintas.

Além dos conflitos no nível de esquema, muitos conflitos aparecem somente no nível de instância. Todos os problemas do caso de fonte única podem ocorrer em fontes diferentes. Além disso, mesmo quando as fontes possuem os mesmos nomes de atributos e tipos de dados, podem existir diferentes representações de valor como por exemplo para o estado civil, ou interpretações diferentes dos valores, como por exemplo diferentes unidades de

medida. Mais ainda, informações nas fontes podem estar agregadas em diferentes níveis como por exemplo total de vendas por produto e total de vendas por grupo de produto, ou referir-se a diferentes períodos de tempo. Um exemplo é a representação do total de vendas diárias em uma fonte enquanto em outra, o total de vendas é semanal.

Um problema chave para a limpeza de dados de múltiplas fontes é a identificação de dados sobrepostos, os quais se referem à mesma entidade do mundo real. Este problema também é conhecido como problema da identidade do objeto [Galhardas et al. 2000], eliminação de duplicatas [Bitton e DeWitt 1983] ou *merge/purge* [Hernandez et al. 1998]. Freqüentemente, a informação é apenas parcialmente redundante e as fontes podem complementar uma a outra provendo informações adicionais sobre uma entidade. Assim, informações duplicadas devem ser eliminadas e informações complementares devem ser consolidadas e integradas para alcançar uma visão consistente das entidades do mundo real.

O processo de limpeza de dados compreende os seguintes passos [Rahm e Do 2000]:

- Análise dos dados: Para detectar quais os tipos de erro e inconsistências que serão removidos, é necessária uma análise detalhada dos dados. Além de uma inspeção manual dos dados ou de amostras dos dados, programas de análise podem ser usados para projetar metadados sobre as propriedades dos dados e detectar problemas de qualidade dos dados.

- Definição do fluxo de transformação e regras de mapeamento: Dependendo da quantidade de fontes de dados, seu grau de heterogeneidade e a quantidade de erros dos dados, um grande número de transformações nos dados e passos de limpeza podem ser executados. Algumas vezes, uma tradução de esquema é usada para mapear fontes para um modelo de dados comum. Passos preliminares de limpeza de dados podem corrigir problemas de instância de fonte única e preparar os dados para a integração. Posteriormente, os problemas de integração de esquemas e limpeza de instâncias são tratados.

  As transformações de dados relativas ao esquema assim como os passos de limpeza

de dados devem ser especificados por uma consulta declarativa e uma linguagem de mapeamento sempre que possível, permitindo a geração automática do código de transformação. Além disso, deve ser possível chamar o código de limpeza escrito pelo usuário e ferramentas de propósito específico durante um fluxo de transformação de dados. Os passos de transformação podem requisitar um *feedback* do usuário em instâncias de dados para as quais não há uma lógica de limpeza implementada.

- Verificação: A correção e eficácia de um fluxo de transformação e definições de transformação devem ser testadas e avaliadas em uma amostra ou cópia da fonte de dados. Múltiplas iterações dos passos de análise, projeto e verificação podem ser necessários para os casos nos quais erros só se tornam visíveis após a aplicação de algumas transformações.

- Transformação: A transformação corresponde à execução dos passos de transformação, executando um fluxo de *extração, transformação e carga dos dados* num repositório integrado de dados.

- Fluxo de volta dos dados limpos: Após a remoção de erros de fonte única, os dados limpos podem substituir os dados inconsistentes nas fontes originais para que as aplicações legadas possam também usufruir dos dados melhorados e para evitar ter que refazer o processo de limpeza em futuras extrações de dados.

Para garantir a qualidade dos dados, informações detalhadas sobre o processo de transformação devem ser registradas, tanto no repositório como nas instâncias transformadas. Em particular, informações sobre a completude e atualidade da fonte de dados e informações de proveniência sobre a origem dos objetos transformados, bem como as mudanças aplicadas sobre eles devem ser mantidas.

2.7 Proveniência dos Dados

Nesta seção são analisadas algumas estratégias presentes na literatura que dizem respeito ao nível de proveniência de dados que pode ser utilizado no processo de atualização de

uma base de dados. Posteriormente, são analisadas algumas estratégias de anotações que, além do propósito de representação da ordem de um documento XML, podem ser utilizadas no processo de anotação de proveniência dos dados.

2.7.1 Proveniência dos dados

A palavra proveniência é usada como sinônimo da palavra *linhagem* na comunidade de bancos de dados. É também algumas vezes identificada como *atribuição de fonte* ou *etiquetação de fonte*. Informações sobre proveniência constituem a prova de corretude dos dados e, por sua vez, determinam a qualidade e quantidade de confiança que são atribuídas aos dados [Tan 2007]. Por estas razões, a proveniência é considerada tão importante quanto os dados em si. Há duas granularidades de proveniência consideradas na literatura: proveniência de fluxo (granulagem grossa) e proveniência de dados (granulagem fina). A proveniência de fluxo refere-se ao registro de toda a história da derivação da saída final de uma seqüência de eventos. A proveniência de fluxo é importante na computação científica, mas não é a principal preocupação em bases de dados materializadas, como por exemplo em um repositório integrado de dados. Por esse motivo, nesta dissertação o foco está na proveniência de dados (ou de granulagem fina), a qual descreve como os dados são importados e transportados entre bases de dados.

Mais especificamente, é considerado o problema de rastreamento e gerenciamento de proveniência, descrevendo as ações do usuário envolvidas na construção de uma base de dados materializada. Isto inclui tanto o registro de modificações locais na base de dados (inserção, exclusão e alteração de dados), quanto operações globais, tal como copiar dados de fontes de dados externas.

A proveniência de dados fornece um extrato relativamente detalhado da derivação de uma parte dos dados que está no resultado de um passo de transformação. Um caso particular de proveniência de dados que é do interesse da comunidade de bancos de dados é quando a transformação é especificada por uma consulta. Mais precisamente, suponha que uma transformação na base de dados D é especificada pela consulta Q. A proveniência dos dados t que pertencem ao resultado da consulta Q sobre D, representado por $Q(D)$

é a resposta para a seguinte questão: Quais os dados da fonte D contribuíram para t de acordo com Q?

a)

(1) delete	c5	from T;
(2) copy	S1/a1/Y	into T/c1/Y;
(3) insert	{c2 : {]}	into T;
(4) copy	S1/a2	into T/c2;
(5) insert	{Y : {]}	into T/c2;
(6) copy	S2/b3/Y	into T/c2/Y;
(7) copy	S1/a3	into T/c3;
(8) insert	{c4 : {]}	into T;
(9) copy	S2/b2	into T/c4;
(10) insert	{Y : 12}	into/c4;

b)

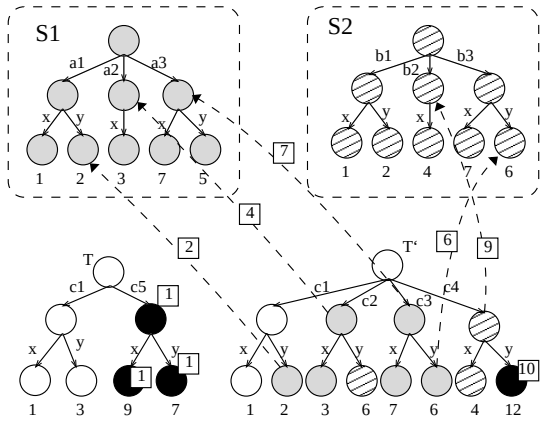


Figura 2.9: (a) Um exemplo de operação copiar e colar. (b) Exemplo da execução das operações descritas em (a) [Buneman et al. 2004].

a) Prov

Tid	Op	Loc	Src
121	D	T/c5	⊥
121	D	T/c5/x	⊥
121	D	T/c5/y	⊥
122	C	T/c1/y	S1/a1/y
123	I	T/c2	⊥
124	C	T/c2	S1/a2
124	C	T/c2/x	S1/a2/x
125	I	T/c2/y	⊥
126	C	T/c2/y	S2/b3/y
127	C	T/c3	S1/a3
127	C	T/c3/x	S1/a3/x
127	C	T/c3/y	S1/a3/y
128	I	T/c4	⊥
129	C	T/c4	S2/b2
129	C	T/c4/x	S2/b2/x
130	I	T/c4/y	⊥

b) Prov

Tid	Op	Loc	Src
121	D	T/c5	⊥
121	D	T/c5/x	⊥
121	D	T/c5/y	⊥
121	C	T/c1/y	S1/a1/y
121	C	T/c2	S1/a2
121	C	T/c2/x	S1/a2/x
121	C	T/c2/y	S2/b3/y
121	C	T/c3	S1/a3
121	C	T/c3/x	S1/a3/x
121	C	T/c3/y	S1/a3/y
121	C	T/c4	S2/b2
121	C	T/c4/x	S2/b2/x
121	I	T/c4/y	⊥

c) HProv

Tid	Op	Loc	Src
121	D	T/c5	⊥
122	C	T/c1/y	S1/a1/y
123	I	T/c2	⊥
124	C	T/c2	S1/a2
125	I	T/c2/y	⊥
126	C	T/c2/y	S2/b3/y
127	C	T/c3	S1/a3
128	I	T/c4	⊥
129	C	T/c4	S2/b2
130	I	T/c4/y	⊥

d) HProv

Tid	Op	Loc	Src
121	D	T/c5	⊥
121	C	T/c1/y	S1/a1/y
121	C	T/c2	S1/a2
121	C	T/c2/y	S2/b3/y
121	C	T/c3	S1/a3
121	C	T/c4	S2/b2
121	I	T/c4/y	⊥

Figura 2.10: Tabelas de proveniência da atualização da Figura 2.9(a) [Buneman et al. 2004].

Em [Buneman et al. 2006], são examinadas quatro formas de efetuar anotações de proveniência de dados segundo o nível de atualização que se deseja armazenar. São elas: a Proveniência Direta, a Proveniência Transacional, a Proveniência Hierárquica e a Proveniência Transacional-Hierárquica.

Exemplo 3 Considere as operações de atualização do tipo "copiar e colar" mostradas na

Figura 2.9(a). Estas operações copiam alguns dados de S_1 e S_2 e modificam alguns valores de dados. O resultado da execução dessa operação de atualização na base de dados T, que possui as fontes de dados S_1 e S_2, é mostrado na Figura 2.9(b). A árvores superiores S_1 e S_2 são visões XML das fontes de dados; as árvores de baixo T e T' são parte da base de dados no começo e no fim da transação. Os nodos brancos são os inalterados e os nodos pretos foram inseridos ou apagados; os demais nodos foram importados de S_1 ou S_2, de acordo com sua cor. As linhas pontilhadas indicam os *links* de proveniência e os números em caixas indicam a operação relevante em (a).

A Figura 2.10 mostra as tabelas de proveniência da transação da Figura 2.9(a). A tabela da Figura 2.10(a) é o resultado da Proveniência Direta, enquanto a tabela da Figura 2.10(b) é o resultado da Proveniência Transacional. A tabela da Figura 2.10(c) mostra as anotações de proveniência necessárias na Proveniência Hierárquica, e, finalmente, a Figura 2.10(d) mostra a tabela de anotações de Proveniência Transacional-Hierárquica.

□

Esses quatro tipos de proveniência são detalhados a seguir.

Proveniência Direta. Este método armazena um registro de proveniência para cada nodo copiado, inserido ou apagado. Além disso, cada operação de atualização é tratada como uma transação separada. Esta técnica pode ser dispendiosa em termos de espaço, pois introduz um registro de proveniência para cada nodo inserido, apagado ou copiado durante a atualização. Entretanto, esse método retém o máximo possível de informação sobre as ações do usuário. De fato, a operação exata de atualização descrevendo a seqüência de ações do usuário pode ser recuperada da tabela de proveniência.

Proveniência Transacional. Na proveniência transacional, as operações de atualização são agrupadas em transações maiores que uma operação isolada. Somente *links* de proveniência são armazenados descrevendo o conjunto de mudanças resultantes de uma transação. Por exemplo, se o usuário copia dados de S_1, apaga-os em seguida, copia dados de S_2 no seu lugar, e finaliza a transação, isto tem o mesmo efeito que a simples cópia de dados de S_2. Desta forma, detalhes sobre estados intermediários ou armazenamento de dados temporários na base de dados não são mantidos.

A proveniência transacional pode ser menos precisa do que a abordagem direta. No entanto, a decisão de quando finalizar a transação e gravar os dados está nas mãos do usuário: gravações de dados freqüentes podem ser usadas para registrar estados intermediários importantes.

O custo de armazenamento da proveniência de uma transação é proporcional ao número de nodos na entrada e saída da transação. Isto é, o número de registros de proveniência transacional produzidos por uma transação de atualização t é $i + d + c$, onde i é o número de nodos inseridos na saída, d é o número de nodos excluídos da entrada e c é o número de nodos copiados na saída.

Proveniência Hierárquica. Tanto na proveniência direta como na transacional, grande parte da informação de proveniência tende a ser redundante (veja Figura 2.10(a) e 2.10(b)), uma vez que em muitos casos a anotação de um nodo filho pode ser inferida da anotação do seu pai. Por esse motivo, pode-se considerar uma outra técnica, chamada de proveniência hierárquica. O observação chave é que não é necessário armazenar todos os *links* de proveniência explicitamente, pois a proveniência de um filho de nodo copiado pode ser inferida da proveniência do seu pai.

Assim, na proveniência hierárquica são armazenados somente os *links* de proveniência que não podem ser inferidos. Estes *links* não inferíveis correspondem aos *links* de proveniência mostrados na Figura 2.9(b). Uma operação de copiar e colar que copia p dentro de q resulta em adicionar somente um único registro $HProv(t; C; q; p)$. A Figura 2.10(c) mostra a tabela de proveniência hierárquica correspondente à versão direta de $Prov$. Neste caso, a tabela é aproximadamente 25% menor do que $Prov$, mas ganhos maiores são possíveis quando registros inteiros ou sub-árvores são copiadas com poucas mudanças. Ao contrário da proveniência transacional, a proveniência hierárquica não descarta informações e não requer que o usuário agrupe operações em transações.

Proveniência Transacional-Hierárquica. Finalmente, pode-se considerar a combinação das técnicas de proveniência transacional e hierárquica. A Figura 2.10(d) mostra a proveniência transacional-hierárquica da proveniência da transação da Figura 2.9(a).

O número de registros para o armazenamento da proveniência transacional-hierárquica

é $i + d + C$, onde i e d são definidas como na proveniência transacional e C é o número de raízes das sub-árvores copiadas que aparecem na saída. A proveniência transacional-hierárquica pode ser mais concisa do que as abordagens transacional e hierárquica separadas.

Embora não possa ser diretamente comparado, o modelo de anotação de proveniência proposto nesta dissertação é mais próximo à proveniência direta. Apesar de ser o método de anotação de proveniência menos conciso, ele foi escolhido pelo fato de permitir que todo elemento inserido na base de dados seja identificado. Em um ambiente em que é possível definir diversas formas de reestruturação, conforme é discutido no Capítulo 3, a utilização dos métodos transacional ou hierárquico no modelo proposto seria complexa.

Além da anotação de proveniência, o modelo proposto nesta dissertação necessita realizar anotações que definam o posicionamento de um elemento dentro da sua fonte de dados antes de inseri-lo no repositório integrado de dados. O objetivo dessa anotação é viabilizar a reconstrução da fonte de dados, conforme discutido na Seção 3.4. Na próxima seção são analisadas algumas técnicas de endereçamento de nodos em árvores XML.

2.7.2 Endereçamento de nodos em árvores XML

Em [Tatarinov et al. 2002] são analisadas três formas de endereçamento de nodos em árvores XML que permitem manter a ordem dos elementos dentro do documento quando é necessário efetuar alguma transformação, como, por exemplo, reconstruir o documento a partir de um conjunto de mapeamentos.

As três formas de endereçamento são: Ordem Global, Ordem Local e Ordem Dewey.

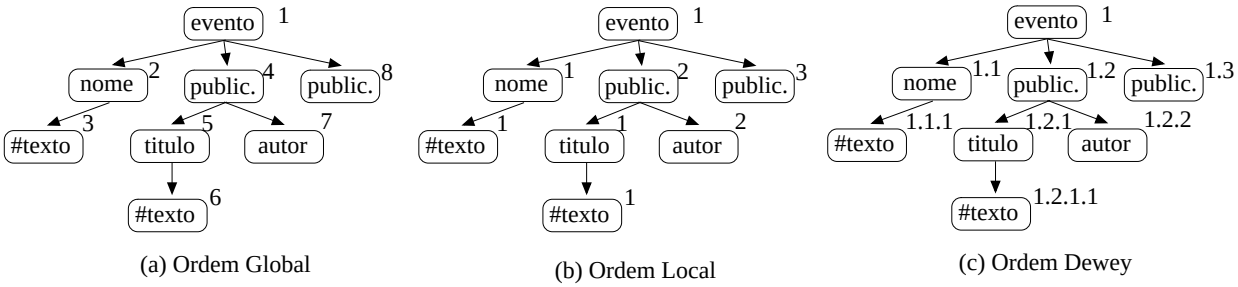


Figura 2.11: Ilustração dos métodos de codificação de ordem.

Ordem Global. Com a Ordem Global, a cada nodo é atribuído um número que representa sua posição absoluta no documento. Qualquer esquema de numeração pode ser usado desde que seja consistente com a ordem do documento. A Figura 2.11(a) apresenta um documento com nodos identificados de acordo com a Ordem Global.

A Ordem Global facilita o processamento de consultas XPath que usam eixos de ordem, como o eixo do `próximo elemento` e o eixo do `elemento anterior`. Além disso, tais consultas podem ser traduzidas em condições de comparação simples entre posições de nodos. Quando atualizações são realizadas em documentos endereçados com a Ordem Global, o desempenho pode ser melhorado utilizando-se uma numeração esparsa. Com a numeração esparsa, a exclusão de um fragmento XML não requer que os nodos remanescentes sejam renumerados.

Adicionalmente, intervalos são deixados entre valores de posição quando a numeração inicial é realizada. Como resultado, inserções podem não requerer renumeração para acomodar um novo fragmento XML. No pior caso, entretanto, o número de valores de posição disponíveis pode ser menor do que o número de nodos no fragmento XML que está sendo inserido. Neste caso, alguns (ou todos) os elementos que sucedem o fragmento inserido são renumerados, como ilustrado na Figura 2.12.

Pode ser interessante usar valores reais (ponto flutuante) ao invés de inteiros para representar uma posição. Em teoria, há um número infinito de valores reais entre qualquer par de valores. Assim, inserções nunca requereriam renumeração. Mas, se tanto valores reais quanto inteiros forem representados com o mesmo número de *bits*, há um número finito de valores entre quaisquer dois valores reais armazenados no computador. Assim, usar valores reais ao invés de inteiros não traz qualquer benefício.

Ordem Local. Com a Ordem Local, a cada nodo é atribuído um número que representa sua posição relativa entre seus irmãos, como ilustrado na Figura 2.11(b). A Ordem Local é suficiente para recriar a ordem do documento, uma vez que a combinação da posição de um nodo com aquela do seu ancestral forma um vetor de caminhos que identifica unicamente a posição absoluta do nodo dentro do documento. Em outras palavras, esse vetor de caminhos provê um ordenamento global dos nodos.

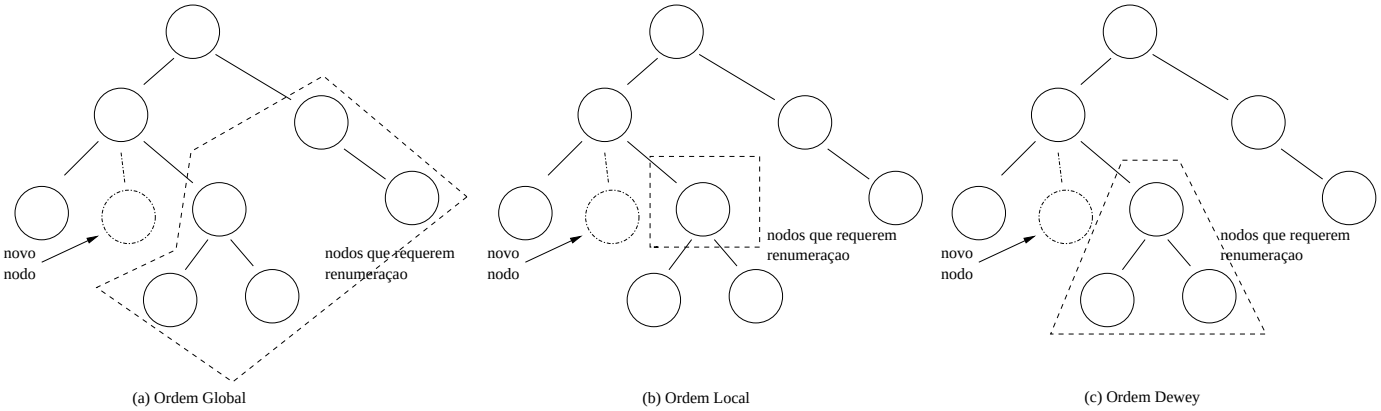


Figura 2.12: O pior caso nos cenários de renumeração para as ordens Global, Local e Dewey.

Como é mostrado na Figura 2.12, a vantagem da Ordem Local é a menor quantidade de renumerações causada pelas atualizações. Somente os irmãos seguintes do novo nodo precisam ser renumerados. Assim como na Ordem Global, a numeração esparsa pode melhorar o desempenho das atualizações. Entretanto, o baixo custo das inserções vem em detrimento do custo de avaliação de consultas XML que envolvem ordenações. Com Ordem Local, a ordem dos eixos, tais como `próximo elemento` e `elemento anterior` são difíceis de avaliar uma vez que nenhuma informação da ordem global está disponível diretamente no nodo.

Ordem Dewey. A Ordem Dewey é baseada na Classificação Decimal de Dewey desenvolvida para a classificação genérica de conhecimento [Chan e Mitchell 2003]. Com a Ordem Dewey, a cada nodo é atribuído um vetor que representa o caminho da raiz do documento até o nodo. Cada componente do caminho representa a ordem local de um nodo ancestral, como ilustrado na Figura 2.11. Cada caminho identifica unicamente a posição absoluta do nodo dentro do documento.

Como os caminhos Dewey provêem ordenação global dos nodos, o processamento de consultas em Ordem Dewey é similar à Ordem Global. Em termos de custo de atualizações, a Ordem Dewey representa o ponto médio entre as Ordens Global e Local. Somente o eixo do `próximo irmão` e seus descendentes precisam ser renumerados, como mostra a Figura 2.12. Apesar de a Ordem Dewey combinar muitas das vantagens da Ordem Global e Ordem Local, uma das suas potenciais desvantagens é o espaço extra

requerido para armazenar caminhos da raiz até o nodo.

Apesar dessa característica de utilizar mais espaço de armazenamento, a Ordem Dewey é a técnica utilizada para endereçamento dos nodos dentro de um documento XML na proposta desta dissertação. Como os documentos XML são armazenados em uma repositório integrado de dados, a característica da técnica de endereçamento Dewey que motivou a sua escolha é a possibilidade de um elemento ter sua sub-árvore reconstruída desde a raiz, mesmo quando algum elemento dos níveis superiores não estiver presente na base de dados integrada.

2.8 Resumo

Neste capítulo foram apresentados os desafios de integração de dados em nível de instância e analisadas duas abordagens: visão global e visão local. A abordagem visão global, também conhecida como integração virtual, caracteriza-se por integrar fontes de dados a uma base de dados que não armazena fisicamente os dados. Desta forma, quando consultas são realizadas na visão integrada, elas são submetidas às fontes de dados. Ela possui como vantagem principal o estado atualizado dos dados. Entretanto, sua principal desvantagem é o tempo de resposta para consultas submetidas à visão integrada. A outra abordagem analisada, visão local, armazena efetivamente os dados das fontes de dados integradas. Nessa abordagem, os maiores desafios encontram-se na evolução do esquema na presença de atualizações e o próprio processo de atualização. Deve existir um processo que persista as atualizações ocorridas nas fontes de dados e, ao contrário da abordagem anterior, não há garantia de que os dados disponíveis sejam sempre os mais atualizados. Além disso, foram apresentadas duas classificações para o sistema de integração de dados: na integração de dois níveis os dados das fontes de dados são integrados diretamente; já na integração de três níveis, existe um documento base que orienta em qual formato as fontes de dados devem ser integradas.

As chaves XML foram apresentadas como uma possível solução de identificação de elementos em um documento XML. A proposta de chaves XML visa definir uma restrição que seja utilizável em quaisquer documentos XML, independente do esquema.

Por ser um formato semi-estruturado, as asserções de chaves do modelo relacional não são aplicáveis quando se trabalha com XML. Assim, é necessário definir algumas características necessárias para que a especificação das chaves XML seja factível. Dentre tais características, as mais importantes são as chaves hierárquicas e as chaves relativas. As chaves hierárquicas vão ao encontro da própria estrutura do XML, que permite a criação de níveis ilimitados de elementos. Outra característica muito importante é a utilização de chaves relativas. As chaves relativas permitem que existam nodos identificáveis dentro de sub-árvores, e não em todo o documento.

Em seguida foi apresentada uma proposta que utiliza as chaves XML para o armazenamento de versões de um documento XML. Nessa proposta o processo de identificação dos elementos é facilitado através de um processo de varredura dos documentos e os valores que identificam um elemento com chave são anotados como atributos desse elemento. Além disso, é utilizada uma técnica de anotação de rótulos de tempo nos elementos atualizados, diferenciando em quais momentos determinados elementos foram atualizados através de um número da versão.

Foram estudadas também algumas classificações que influenciam na limpeza de dados em um repositório integrado de dados. Os problemas de inconsistências encontrados em bases de dados podem ser classificados em problemas de fonte única e problemas de múltiplas fontes. No nível de esquema os problemas de fonte única caracterizam-se pela falta de restrições de integridade ou por um projeto deficiente do esquema. Já no nível de instância, os problemas são basicamente encontrados na entrada dos dados e variam desde erros ortográficos até valores contraditórios. Já os problemas de múltiplas fontes, no nível de esquema, incluem conflitos estruturais e conflitos de nomes. No nível de instância podem-se considerar os problemas de fonte única, mais problemas originários da integração dos dados, como a integração inconsistente dos dados.

Finalmente foram apresentadas algumas técnicas para a determinação da proveniência dos dados e técnicas de anotação de informações relativas a ordem dos nodos de uma árvore XML. A proveniência dos dados pode ser determinada diretamente, inserindo um registro de proveniência para cada elemento extraído da fonte. Pode-se considerar também níveis

de proveniência mais altos, como a que determina proveniência no nível de transação, atribuindo essa informação somente quando as transações são gravadas. A proveniência hierárquica atribui dados de proveniência somente aos pais de nodos oriundos da mesma fonte, tornando necessário que, para determinar a proveniência de um determinado nodo, seja necessário consultar a proveniência do seu nodo pai. Há também a integração dessas duas últimas abordagens criando a proveniência transacional-hierárquica. que agrupa as características das proveniências transacional e hierárquica numa única técnica. Além disso, foram mostradas três técnicas distintas para a anotação da ordem dos elementos num documento XML, que são a Ordem Global, Ordem Local e Ordem Dewey. A Ordem Dewey, apesar de ser mais dispendiosa do ponto de vista de espaço de armazenamento, é a mais completa para determinar a posição de cada elemento, mesmo quando não se tem acesso a toda a estrutura do documento.

Tabela 2.1: Sumário das abordagens adotadas

Problema	Abordagens	Abordagem escolhida
Integração de instâncias	Visão Global e Visão Local	Visão Local
Integração de dados	Integração de 2 níveis e de 3 níveis	Integração de 2 níveis
Identificação de entidades	Chaves XML, DTD e XML Schema	Chaves XML
Proveniência de dados	Proveniência direta, transacional, hierárquica e transacional-hierárquica	Proveniência direta
Endereçamento de nodos	Ordem global, local e Dewey	Ordem Dewey

Dentre as abordagens estudadas, pode-se enumerar quais estão presentes no modelo proposto neste trabalho. A Tabela 2.1 sumariza os problemas encontrados e as abordagens escolhidas. No âmbito da integração de instâncias, é utilizada a abordagem visão local, onde as fontes de dados são efetivamente armazenadas no repositório integrado de dados. Para a integração de dados, a técnica utilizada é semelhante a técnica de integração de dois níveis, onde as fontes de dados são integradas diretamente. Para a identificação de entidades utilizam-se as chaves XML. Apesar de não ser a proposta deste trabalho promover a limpeza automatizada dos dados, os problemas de limpeza de dados desse modelo são os identificados como problemas de múltiplas fontes. Para a anotação da proveniência dos dados é utilizada a técnica de proveniência direta, onde cada nodo armazenado re-

cebe uma anotação de proveniência. E, finalmente, o processo de endereçamento dos nodos obedece a Ordem Dewey, que adiciona a cada nodo o identificador que representa o caminho completo desde a raiz da árvore. No próximo capítulo o modelo de dados dessa dissertação é apresentado.

CAPÍTULO 3

INTEGRAÇÃO DE DOCUMENTOS XML

Como o objetivo de um repositório integrado de dados é combinar dados oriundos de diversas fontes, realizar atualizações de dados diretamente nele é muito raro. Isto porque o processo de atualização se dá nas fontes dos dados e é apenas refletido no repositório integrado de dados. Mas novos dados podem ser inseridos. Além disso, à medida que novas versões dos documentos XML armazenados são geradas ou disponibilizadas é desejável a existência de um sistema que detecte as diferenças entre os dados armazenados e a nova versão e que gere automaticamente os comandos de atualização necessários para manter o repositório integrado atualizado.

Entretanto, para que a utilização de um repositório integrado seja possível, duas características básicas devem ser respeitadas: estruturação e integração de dados.

A estruturação de dados num repositório integrado é uma característica obtida através da definição de uma estrutura de dados fixa, que defina em qual formato os dados devem ser mantidos nesse repositório. A utilização de repositórios integrados que possuam dados oriundos de uma mesma organização facilita a tarefa de estruturação. Isto porque uma única organização pode controlar o formato dos dados de suas fontes de dados, criando um formato homogêneo que atualize o repositório integrado. Como consequência, a estrutura do repositório integrado é um espelho do formato de dados das fontes.

Entretanto, a possibilidade de organizações distintas cooperarem entre si para a criação de uma base de dados consolidada num determinado assunto afasta a possibilidade de assumir que as fontes de dados possuem a mesma estrutura. Cada organização pode estruturar os seus dados de maneira distinta e, mesmo assim, desejar que esses dados sejam integrados e complementados por dados de outras fontes.

Uma solução simples para o armazenamento de diferentes fontes de dados seria manter os dados no formato no qual eles foram originalmente estruturados em uma base central.

Todavia, essa solução apresenta restrições ao pleno funcionamento da base de dados. A primeira restrição diz respeito ao acesso dos dados.

Idealmente, o repositório integrado deve possibilitar a localização de uma informação específica de forma padronizada. Na situação em que cada fonte de dados é armazenada de forma isolada, o acesso aos dados seria prejudicado. Não existiria uma forma padronizada de acessar informações de múltiplas fontes. Cada fonte poderia apenas ter seus dados acessados isoladamente, desde que haja um conhecimento prévio do formato da fonte de dados.

Para que o acesso ao repositório integrado seja pleno, o modelo possui uma estrutura fixa e pré-determinada, provendo uma forma padronizada de acesso aos dados armazenados. Uma camada de conversão é utilizada para que a estrutura da fonte de dados seja convertida na estrutura do repositório integrado. Essa camada de conversão tem como principal componente uma *Linguagem de Mapeamento* e *Algoritmos de Conversão*.

A Linguagem de Mapeamento, além de prover mecanismos para a tradução do esquema das fontes de dados para o esquema do repositório integrado, também permite que os dados já armazenados no repositório integrado possam ser transformados novamente no esquema da fonte de dados. Através do mapeamento definido e anotações de proveniência, o sistema consegue resgatar os dados originários de uma determinada fonte e reconstruir o esquema da fonte de dados para que futuras versões da mesma fonte possam ser comparadas com as versões já armazenadas.

Já a integração de dados num repositório integrado é a característica que permite que dados de fontes distintas sejam agrupados formando uma única informação do ponto de vista semântico. Para que a integração de dados seja realizada é proposta a utilização de chaves XML em todos os níveis do repositório integrado fazendo com que os dados armazenados possam ser corretamente identificados. Os dados das fontes são convertidos para o esquema do repositório integrado e integrados com os dados já armazenados através da utilização das chaves XML definidas no repositório integrado.

Nas próximas seções são definidos o modelo de dados utilizado (Seção 3.1), o modelo de representação de inconsistências (Seção 3.2), a linguagem de mapeamento (Seção 3.3),

o processo de reconstrução das fontes de dados armazenadas (Seção 3.4) e, finalmente, os algoritmos utilizados (Seção 3.5).

3.1 O Modelo de Dados

O modelo de dados do repositório integrado é uma árvore XML contendo dados importados de uma ou mais fontes de dados XML. Todos os níveis da árvore possuem nodos identificáveis através de chaves XML, que determinam como identificar uma entidade em uma fonte e também como integrar dados oriundos de fontes distintas. O esquema do repositório integrado de dados é fixo, utilizando a abordagem de visão local definida na Seção 2.3. Para efeito de carga de dados, é proposta uma linguagem de mapeamento que descreve como converter o esquema de uma fonte de dados XML no esquema utilizado pelo repositório integrado de dados, definida na Seção 3.3. Os dados armazenados são anotados com informação de proveniência. A informação de proveniência é composta por um identificador da fonte de dados e um vetor de identificação interno que possibilita a localização do elemento relativamente a raiz da fonte de dados. Esse vetor de identificação utiliza a abordagem de endereçamento de Ordem Dewey, definido na Seção 2.7.2. Essa abordagem de anotações, apesar de ser mais dispendiosa do ponto de vista de armazenamento, foi escolhida por ser a mais viável no processo de reconstrução das fontes de dados. As anotações de proveniência são feitas em todos os nodos folha oriundos da fonte de dados. Desta forma, essa abordagem se assemelha mais com a abordagem de Proveniência Direta, apresentada na Seção 2.7.1.

A reconstrução das fontes de dados é utilizada para que, no momento em que uma fonte já armazenada disponibilizar uma nova versão, uma ferramenta externa de detecção de diferenças em documentos XML possa detectar quais são as atualizações que devem ser armazenadas. O processo de reconstrução a partir dos identificadores Dewey permite que a fonte seja reconstruída preservando a ordem do documento XML original, sem que seja necessário armazenar o documento original completo. Nas seções subseqüentes, tais abordagens são analisadas detalhadamente e os algoritmos são apresentados.

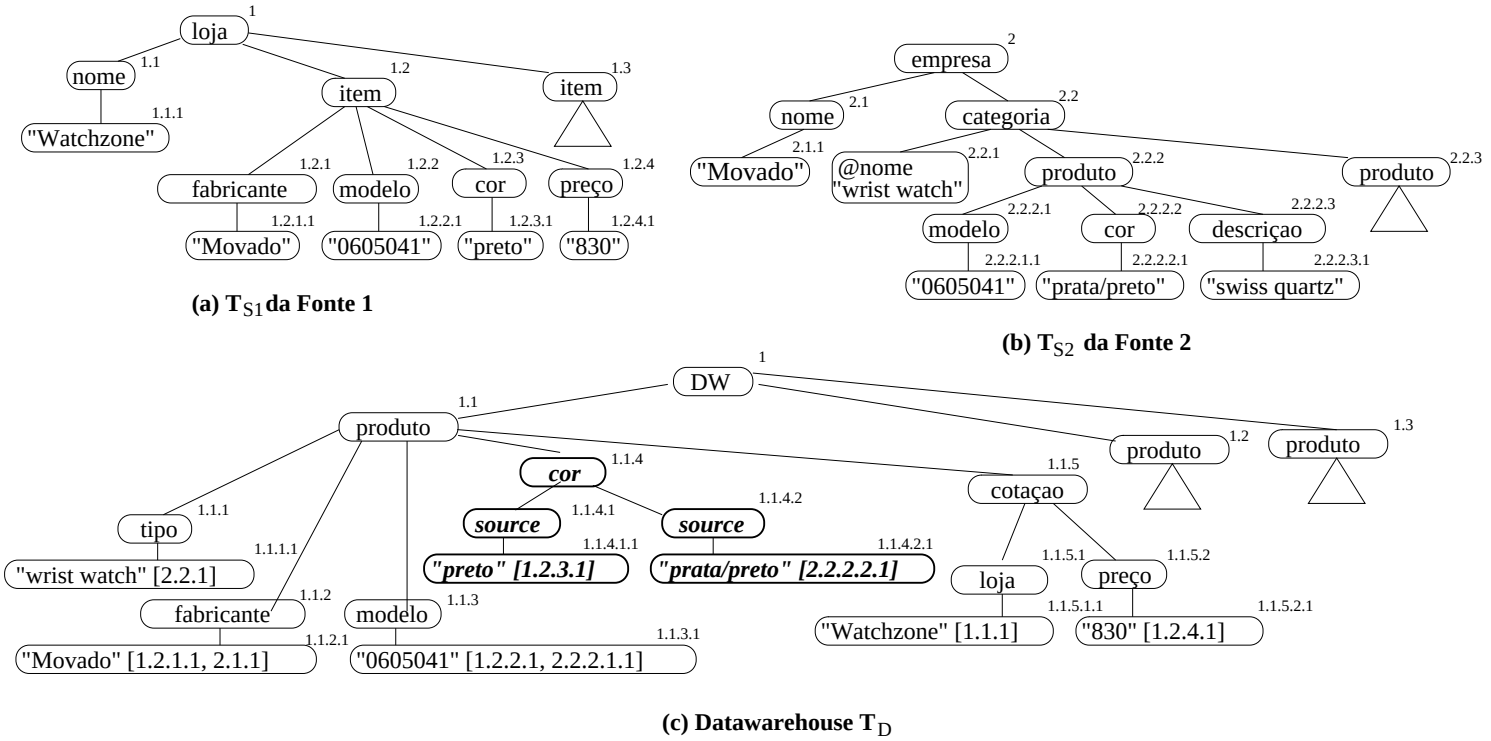
3.1.1 Árvore XML

Um documento XML pode ser modelado como uma árvore de nodos rotulados. São considerados três conjuntos de rótulos: **E** de nomes de elementos, **A** de nomes de atributos, e um conjunto {S} denotando texto (PCDATA).

Definição Uma *árvore XML* T é definida como $T = (source, V, r, id, lab, ele, att, val)$, onde

1. *source* é a identificação da fonte de dados;
2. V é um conjunto de nodos;
3. r é o nodo raiz;
4. id é a função que atribui identificadores únicos aos nodos em V; dado um nodo v, $id(v)$ retorna um vetor que representa o caminho de r até v.
5. lab é uma função $V \rightarrow \mathbf{E} \cup \mathbf{A} \cup \{\mathtt{S}\}$ a qual atribui um rótulo para cada nodo em V; um nodo v em V é chamado um *elemento* se $lab(v) \in \mathbf{E}$, um *atributo* se $lab(v) \in \mathbf{A}$, e um nodo texto se $lab(v) = \mathtt{S}$;
6. ele e att são funções parciais que definem a relação entre arestas de T para todo nodo v em V,
   - se v é um elemento então $ele(v)$ é uma *lista* de elementos e nodos texto em V e $att(v)$ é um *conjunto* de atributos em V; para cada v' em $ele(v)$ ou $att(v)$, v' é chamado *filho* de v e há uma aresta (direcionada) de v para v';
   - se v é um atributo ou um nodo texto então $ele(v)$ e $att(v)$ são indefinidos;
7. val é uma função parcial que atribui uma *string* para cada atributo e nodo texto: para cada nodo v em V, se v é um atributo ou nodo texto então $val(v)$ é uma *string*, e $val(v)$ é indefinido se v é um nodo elemento;

Uma árvore XML possui uma estrutura de árvore, isto é, para cada $v \in V$, há um único caminho de arestas da raiz r até v.



(a) T_{S1} da Fonte 1

(b) T_{S2} da Fonte 2

(c) Datawarehouse T_D

Figura 3.1: Exemplos de árvores XML.

Exemplos de árvore XML são dados nas Figuras 3.1 (a) e (b), onde cada nodo v é representado com seu identificador ($id(v)$), um rótulo ($lab(v)$) se ele é um nodo elemento ou atributo, e um valor ($val(v)$) se ele é um nodo atributo ou texto. A codificação adotada pela função identificadora id é a *Ordem Dewey*, a qual provê uma ordenação global dos nodos. É assumido que cada fonte de dados possui uma identificador de fonte *source* distinto, e que este valor é usado como o identificador do nodo raiz, isto é, $id(r) = source$. No exemplo da Figura 3.1(a), a `Fonte 1` possui identificador 1, assim $id(r) = 1$. A função *origin* é definida de forma que. dado um id do nodo de uma árvore XML T, ela retorna o identificador da fonte de dados *source*; isto é, $origin(id) = source(T)$.

A linguagem de caminhos adotada é um fragmento de expressões regulares. Um caminho p é uma seqüência de rótulos $l_1/\ldots/l_n$. Uma expressão de caminho Q define um conjunto de caminhos, enquanto "//" pode ser combinado com qualquer caminho. Dada uma árvore XML T, denota-se $Paths(T)$ o conjunto de caminhos em T. Como um exemplo, considere que T é a árvore XML da Figura 3.1(a). $Paths(T) = \{/, /nome, /item, /item/fabricante, /item/modelo, /item/cor, /item/preço\}$.

Definição Um repositório integrado de dados com dados importados de um conjunto de árvores XML S é definido como $D = (T_D, \mathcal{K}_D)$, onde (1) T_D é uma árvore XML que possui um conjunto de nodos V, tal que cada nodo folha $v \in V$ é anotado com um conjunto de pares (id_n, p), onde id_n é o identificador de um nodo n em uma árvore XML $T \in S$, usad para inserir v, e p é o caminho da raiz r de T até n; (2) $\mathcal{K}_D$ é o conjunto de chaves XML definidas em T_D. Qualquer nodo em T_D pode ser unicamente identificado de acordo com $\mathcal{K}_D$ ou ter um único nodo filho `S`.

Exemplo 4 Considere a árvore XML da Figura 3.1(c). Algumas chaves que podem ser definidas nesta árvore são:

- $K_1 : (\epsilon, (produto, \{fabricante, modelo\}))$: no contexto de todo o documento (ϵ denota a raiz), um produto é identificado pelo seu fabricante e modelo.
- $K_2 : (/produto, (cotação, \{loja\}))$: no contexto de qualquer sub-árvore com raiz em um nodo produto, uma cotação é identificada pelo nome de sua loja.
- $K_3 : (//produto, (tipo, \{\}))$: cada produto possui no máximo um tipo, e, similarmente $K_4 : (//produto, (cor, \{\}))$ para cada cor de um produto, e $K_5 : (//produto/cotação, (preço, \{\}))$ para o preço de uma cotação.

□

Na próxima seção será apresentada uma propriedade adicional do modelo desta dissertação, que é o método utilizado para a representação de inconsistências.

3.2 Inconsistência Temporária

O processo de armazenamento de um documento no repositório integrado de dados que pode sofrer atualizações de várias fontes utiliza uma estratégia semelhante à apresentada em [Buneman et al. 2004], na qual é proposto um modelo para o armazenamento do histórico das atualizações de um único documento. Esse modelo é detalhado na Seção 2.5. Todavia, nesta dissertação o foco é o armazenamento de diversos documentos de fontes distintas com estruturas heterogêneas.

Nesse cenário diversas fontes podem contribuir com dados sobre um mesmo elemento armazenado no repositório integrado de dados. Como cada fonte é atualizada sob seus próprios critérios, elementos compartilhados podem sofrer atualizações na fontes em períodos distintos. Invariavelmente, sempre que ocorrer essa situação, o sistema entra num estado de inconsistência.

As opções quando essa situação ocorre são três. A primeira opção é o sistema não atualizar o repositório integrado de dados e gerar uma inconsistência do repositório integrado de dados em relação à fonte que sofreu a atualização. A segunda opção é o sistema atualizar o repositório integrado de dados e gerar uma inconsistência com as demais fontes que não foram atualizadas. E, finalmente, a terceira opção e a escolhida para esse trabalho, é atualizar o repositório integrado de dados de forma a não gerar inconsistência nem com a fonte que enviou a atualização nem com as demais fontes que não enviaram a atualização. Entretanto, o dado atualizado é duplicado e o próprio repositório integrado de dados entra num estado de "inconsistência temporária".

Essa inconsistência temporária caracteriza-se pelo fato de o repositório integrado de dados duplicar valores que deveriam ser únicos. A Figura 3.2 apresenta o modelo escolhido para a representação de inconsistências. Na Figura 3.2(a) o nodo F corresponde a um elemento simples, que contém apenas um valor textual. Entretanto, após a inserção de algumas fontes com valores inconsistentes entre si sobre o mesmo elemento, esse nodo ganha sub-elementos com nodos que identificam as fontes que inseriram os valores, conforme ilustrado na Figura 3.2(b).

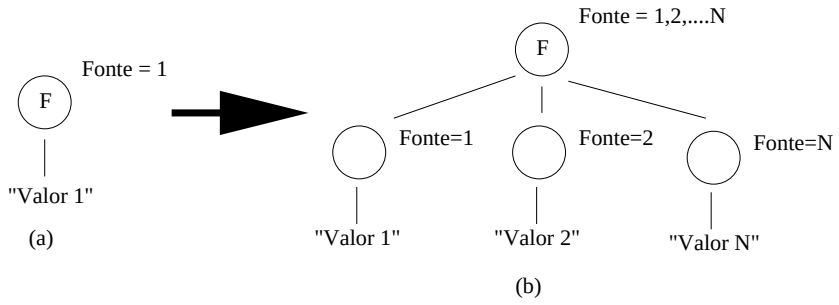
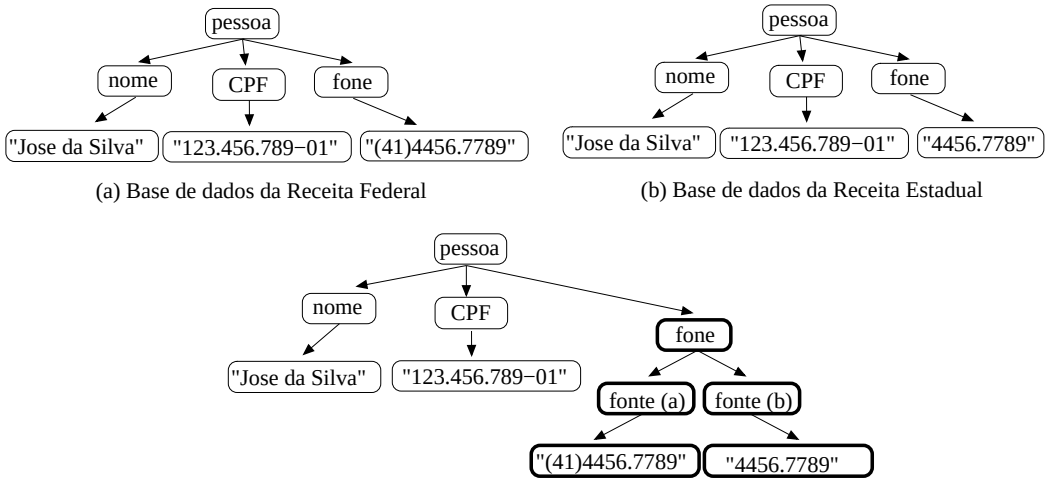


Figura 3.2: Modelo utilizado para a representação de inconsistências.

Exemplo 5 Imagine que a Receita Estadual possui a base de dados representada pela



(a) Base de dados da Receita Federal

(b) Base de dados da Receita Estadual

(c) Bases de dados integradas

Figura 3.3: (a) Extrato dos dados de um contribuinte mantidos na Receita Federal e (b) os dados do mesmo contribuinte mantidos na Receita Estadual. (c) Mostra a integração desses dados.

Figura 3.3(a), que utilizou a mesma estrutura existente na base de dados da Receita Federal, representada pela Figura 3.3(b). Em ambas as bases de dados uma `pessoa` é identificada através do seu `CPF`.

Para fins de maior poder de fiscalização, o governo determina que essas bases de dados sejam integradas. Através das chaves definidas em ambas as fontes, o resultado dessa integração é apresentado na Figura 3.3(c). Note que o elemento fone foi representado de formas distintas nas fontes de dados. Como o sistema não conhece o formato correto que deve ser representado, é gerada uma "inconsistência temporária" nesse elemento conforme destacado na Figura 3.3(c). □

Exemplo 6 O processo de reestruturação ilustrado na Figura 1.2 gerou uma inconsistência no elemento cor. Após os dados das fontes *Fonte 1* e *Fonte 2* serem inseridos no repositório integrado, identificou-se que um mesmo `produto` possui elementos cor com os valores preto e preto/prata. A chave $(//produto, (cor, \{\}))$ não permite que uma cor seja duplicada dentro de um produto. A opção de atualização no repositório integrado de dados é criar elementos source distintos abaixo do elemento cor, cada qual com valores de cada uma das fontes de dados. □

Entretanto, para que fontes de dados distintas sejam integradas, é necessária alguma estrutura que defina quais são as regras de integração. Uma forma simples de definir estas

regras é através de uma linguagem de mapeamento. Na próxima seção é analisada uma proposta de linguagem de mapeamento que define as regras de integração de fontes de dados distintas.

3.3 Linguagem de Mapeamento

Para descrever como os dados são extraídos das fontes, é utilizada uma estratégia de mapeamento de caminhos entre os documentos fonte e o repositório integrado de dados. Nessa estratégia, o algoritmo deve extrair os elementos de um dado caminho no documento fonte e inserí-los em um caminho no repositório integrado de dados. Para tanto, é utilizada uma linguagem bastante simples, que consiste de um conjunto de regras no seguinte formato:

$$null \leftarrow p_S$$
$$p_D \leftarrow p_S$$

onde *null* é uma diretiva de restrição de produto cartesiano, p_D é um caminho simples no repositório integrado de dados, e p_S é um caminho simples na fonte de dados.

Exemplo 7 Considere as seguintes regras de mapeamento:

$$null \leftarrow /C$$
$$/X/A \leftarrow /C/A$$
$$/X/B \leftarrow /C/B$$

Nesse mapeamento, define-se que o elemento C, embora não seja mapeado para nenhum elemento no repositório integrado de dados, é a raiz da sub-árvore da qual os elementos A e B são extraídos. Isto é, seja n um nodo em $[\![C]\!]$, $S_A = n[\![A]\!]$ e $S_B = n[\![B]\!]$. Cada elemento de S_A é combinado com todos os elementos de S_B na geração de um elemento X logo abaixo da raiz do repositório integrado de dados com sub-elementos A e B com valores extraídos da fonte. Caso um dos conjuntos esteja vazio, o elemento correspondente ao conjunto não é criado.

Como exemplo, considere a fonte de dados ilustrada na Figura 3.4(a). A Figura 3.4(b) mostra o resultado da conversão da estrutura conforme o mapeamento definido. A regra $null \leftarrow /C$ restringe o espaço de pesquisa dos sub-elementos de C que aparecem nos mapeamentos. Assim, para o primeiro elemento C, S_A contém os dois elementos A com valores "1"e "2" e S_B é vazio. Como conseqüência, são gerados no repositório integrado de dados os dois primeiros elementos X. Os demais elementos X são gerados a partir do segundo elemento C da fonte. Caso o mapeamento não possuísse a regra $null \leftarrow /C$, o resultado do processo de conversão seria o ilustrado na Figura 3.4(c), no qual é executado o produto cartesiano dos elementos em $[\![C/A]\!]$ da fonte com os elementos $[\![C/B]\!]$. □

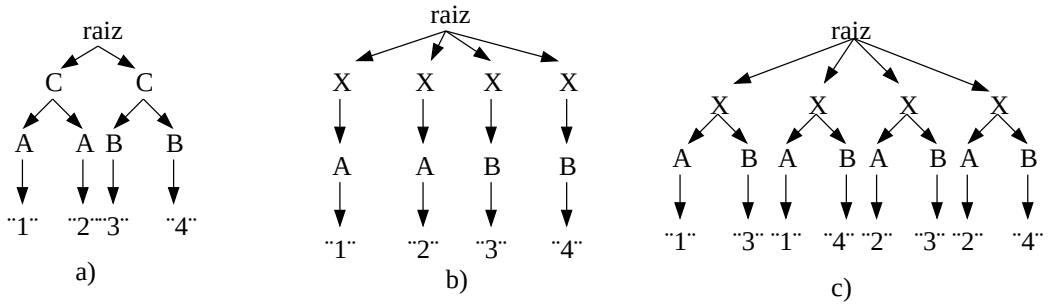


Figura 3.4: Mapeamento entre estruturas com e sem produto cartesiano.

Exemplo 8 Considere a fonte de dados *Fonte 1* e o repositório integrado de dados T_D mostrado na Figura 1.2(a) e (c), respectivamente. O mapeamento da *Fonte 1* para T_D pode ser especificado da seguinte maneira:

$$
\begin{aligned}
&/produto && \leftarrow /item\\
&/produto/fabricante && \leftarrow /item/fabricante\\
&/produto/modelo && \leftarrow /item/modelo\\
&/produto/cor && \leftarrow /item/cor\\
&/produto/cotação/loja && \leftarrow /nome\\
&/produto/cotação/preço && \leftarrow /item/preço
\end{aligned}
$$

□

Vários mapeamentos podem ser especificados para inserir dados no repositório integrado. Cada mapeamento é definido como um par $M = (source, R)$, onde *source* é um identificador de fonte e R é um conjunto de regras. Observe que o significado de um

mapeamento é dado pela estrutura do repositório integrado de dados e o conjunto de chaves. Isto é, uma chave $k = (Q_1, (Q_2, K))$ determina que sempre que dois nodos n_1 e n_2 são alcançáveis através de um caminho $p \in Q_1/Q_2$ e eles concordam nos valores dos seus K sub-elementos, então eles devem ser mapeados para o mesmo nodo n_D no repositório integrado de dados.

Assim, é definida a seguinte restrição para que um mapeamento seja *bem definido*: se $M = (source, R)$ é um mapeamento definido para o repositório integrado de dados $D = (T_D, K_D)$ e $k = (Q_1, (Q_2, K))$ é uma chave em K_D, então para cada caminho $p_D \in Paths(T_D)$ e $p_K \in K$ tal que $p_D \in Q_1/Q_2$, existe uma regra $p_D/P_K \leftarrow p'$ em R para algum p'. Intuitivamente, isto quer dizer que cada chave no repositório integrado de dados deve receber valores de elementos na fonte de dados. Para ilustrar, considere o mapeamento dado no Exemplo 8 e as chaves definidas no Exemplo 4. Esse mapeamento é bem definido uma vez que os valores da chave definidos para o repositório integrado de dados da Figura 1.2(c) definem que *modelo* e $fabricante$ são chaves de $produto$, e $loja$ é chave de $produto/cotação$. Além disso, todos os elementos que compõem as chaves contêm dados extraídos da *Fonte 1* de acordo com o mapeamento.

Apesar da simplicidade, a linguagem de mapeamento permite aninhamento e desaninhamento de dados, além de projeção, união, produto cartesiano e junção nos valores das chaves. A informação referente ao mapeamento entre caminhos do documento-fonte e o repositório integrado de dados deve ser fornecida pelo usuário no momento em que houver a inserção do documento no repositório integrado. Além disso, sempre que houver mudança na estrutura do documento que interfira na correlação entre caminhos no documento fonte e no repositório integrado, a alteração nos caminhos deve ser informada novamente.

A linguagem de mapeamento possui duas funções no modelo desta dissertação. A primeira é a que foi explicada nesta seção e a segunda é a utilizada no processo de reconstrução das fontes de dados armazenadas, onde o caminho do mapeamento é invertido. Na próxima seção a abordagem de reconstrução das fontes de dados que utiliza, basicamente, a linguagem de mapeamento, é apresentada.

3.4 Reconstrução das Fontes de Dados

O processo de reconstrução das fontes de dados tem por objetivo construir um documento XML composto pelos dados armazenados no repositório integrado de dados oriundos de uma determinada fonte e que possua a estrutura original da mesma. Com o documento reconstruído, torna-se possível efetuar comparações com uma versão atualizada da fonte e detectar quais foram as alterações ocorridas, e assim propagar para o repositório integrado de dados apenas essas atualizações.

Para que sejam identificados os dados que pertencem à fonte em questão, são utilizadas as informações de proveniência anotadas nos elementos armazenados. A proveniência dos dados é armazenada em todos os elementos folha do repositório integrado de dados, de forma que se alguma informação foi armazenada, ela garantidamente será encontrada para a reconstrução.

Outra informação anotada nos elementos do repositório integrado de dados é o caminho da fonte de dados a qual pertenciam os elementos. Com essa informação pode-se reconstruir os caminhos das fontes de dados sem que seja necessário que todos os elementos que formam um caminho estejam armazenados no repositório integrado.

Exemplo 9 Considere os nodos preço e loja na Figura 3.1(c), identificados na Ordem Dewey pelos valores 1.1.5.1 e 1.1.5.2, respectivamente. Os mapeamentos que geraram esses elementos foram:

$$/produto/cotação/loja \leftarrow /nome$$
$$/produto/cotação/preço \leftarrow /item/preço$$

O elemento loja recebeu as anotações $id = 1.1, path = /nome$ e o elemento cotação recebeu as anotações $id = 1.2.4, path = /item/preço$. O algoritmo de reconstrução infere do *id* dos elementos que eles pertencem à *Fonte 1*, pois os identificadores dos elementos são iniciados com o identificador da fonte de dados e reconstrói a sub-árvore mostrada na Figura 3.5.

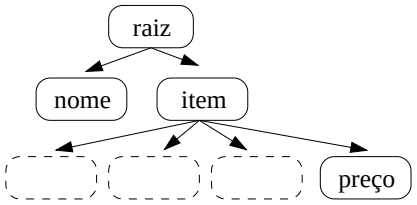


Figura 3.5: Árvore parcialmente reconstruída.

De fato, os elementos representados com linhas pontilhadas não são inseridos na árvore com apenas esses dois mapeamentos. Entretanto, quando os elementos que possuem como nodo pai um item e um $id = 1.2.d$, onde $d \in \{1, 2, 3\}$, tiverem suas sub-árvores reconstruídas, eles serão inseridos nas posições ocupadas pelos elementos com linhas pontilhadas.

□

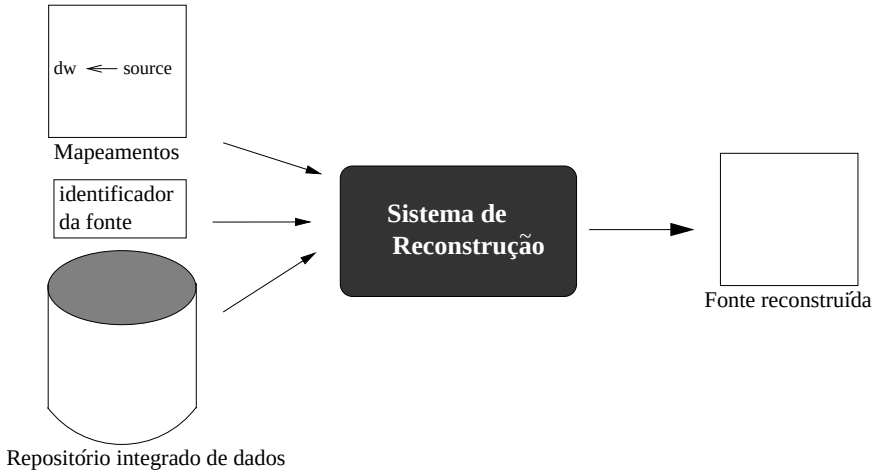


Figura 3.6: Estrutura do sistema de reconstrução das fontes de dados.

A Figura 3.6 apresenta a estrutura do sistema de reconstrução. O sistema de reconstrução utiliza o mapeamento definido para o processo de carga do repositório integrado de dados. No entanto, as regras de mapeamento são utilizadas de forma invertida, seguindo o formato $p_S \leftarrow p_D$. Além disso, o sistema precisa conhecer o identificador da fonte, que está armazenado nos elementos do repositório integrado de dados.

Agora que todas as principais características do modelo de dados foram definidas e detalhadas, são apresentados na próxima seção os algoritmos utilizados para a implementação desse modelo.

3.5 Algoritmos do Modelo

Nesta seção são apresentados os algoritmos do modelo proposto. A Seção 3.5.1 apresenta os algoritmos responsáveis por realizar a carga dos dados no repositório integrado de dados. A Seção 3.5.3 apresenta o algoritmo responsável pelo processo de reconstrução de uma fonte de dados armazenada no repositório integrado de dados.

3.5.1 Carga do repositório integrado de dados

O algoritmo para fazer a carga do repositório integrado de dados consiste de dois passos. Primeiro, são construídas sub-árvores estruturadas como o repositório integrado de dados e com valores extraídos da fonte de dados. Em seguida, essas árvores são unidas com dados existentes no repositório integrado de dados de acordo com o seus valores de chave. O pseudocódigo é apresentado no Algoritmo 1. As entradas desse algoritmo são: um repositório integrado de dados $D = (T_D, K_D)$, um mapeamento $M = (source, R)$ e uma árvore fonte T_S. As seguintes estruturas de dados são usadas pelo algoritmo:

- *sourcePath*: um vetor de caminhos $[p_0, p_1, \ldots, p_m]$, onde $p_0 = /$, e $p_i, i \in [1, n]$ são caminhos definidos na fonte de dados que são usados para povoar o repositório integrado de dados. Isto é, são caminhos do lado direito das regras de R. A lista está ordenada por prefixo: se $p_j = p_i/Q$ para algum caminho Q então $i < j$. O tamanho da lista é dado por $size(M)$, que é o número de regras em M.

- *dwPath*: um vetor de caminhos $[q_0, q_1, \ldots, q_m]$, onde $q_0 = /$ e q_i são caminhos definidos no repositório integrado de dados povoados como caminho $sourcePath[i]$; isto é, $q_i \leftarrow sourcePath[i]$ é uma regra no mapeamento.

- *sourceNode*: um vetor de nodos $[n_0, n_1, \ldots, n_m]$ na fonte de dados de forma que n_i é o último nodo em $[\![sourcePath[i]]\!]_{T_s}$ visitado pelo algoritmo.

A Função `populate` inicializa todos os nodos em *sourceNode* com *null*, e atribui a $sourceNode[0]$ a raiz da árvore fonte T_s. Ela então chama a função

Algoritmo 1 Algoritmo Carga do Repositório Integrado de Dados

1: **function** populate $(D = (T_D, K_D),\ M = (source, R),\ T_s)$
2: inicialize vetor *sourceNode* com valores *null*;
3: $sourceNode[0] \Leftarrow r(T_s)$;
4: return transformAndMerge $(T_D, 1, sourceNode)$;
5: **end function**

6: **function** transformAndMerge $(T_D, ind, sourceNode)$
7: **if** $ind > size(M)$ **then**
8: $T' \Leftarrow$createTree $(sourceNode)$;
9: return mergeTrees (T_D, T', K_D);
10: **else**
11: seja a o índice tal que $sourcePath[a]$ é o prefixo mais longo de $sourcePath[ind]$;
12: $anc \Leftarrow sourceNode[a]$;
13: **if** $anc = null$ **then**
14: $sourceNode[ind] \Leftarrow null$;
15: $T_D \Leftarrow$ transformAndMerge $(T_D, ind + 1, sourceNode)$;
16: **else**
17: let Q be the path s.t. $sourcePath[ind] = sourcePath[a]/Q$;
18: $V \Leftarrow anc[\![Q]\!]_{T_s}$;
19: **if** $V = \emptyset$ **then**
20: $sourceNode[ind] \Leftarrow null$;
21: $T_D \Leftarrow$ transformAndMerge $(T_D, ind + 1, sourceNode)$;
22: **else**
23: **for** each node v in V **do**
24: $sourceNode[ind] \Leftarrow v$;
25: $T_D \Leftarrow$ transformAndMerge $(T_D, ind + 1, sourceNode)$;
26: **end for**
27: **end if**
28: **end if**
29: **end if**
30: **end function**

31: **function** createTree $(sourceNode)$
32: crie nodo raiz r da árvore T';
33: **for** $i := 1$ to $size(M)$ **do**
34: **if** $sourceNode[i] \neq null$ and $dwPath[i] \neq null$ **then**
35: seja p o prefixo mais longo de $dwPath[i]$ tal que $[\![p]\!]_{T'}$ é não vazio.
36: seja anc o único nodo em $[[p]]'_T$
37: **for** $j := 1$ to m **do**
38: crie nodo n_j em T' como um filho de anc com rótulo l_j;
39: $anc \Leftarrow n_j$;
40: **end for**
41: **if** $\nexists p \in dwPath$ tal que $dwPath[i]$ é um prefixo de p **then**
42: $val(n_j) \Leftarrow val(sourceNode[i])$;
43: $n_j.@prov \Leftarrow \{(id(sourceNode[i]), sourcePath[i]\}$;
44: **end if**
45: **end if**
46: **end for**
47: return T';
48: **end function**

`transformAndMerge` para construir uma sub-árvore com nodos em *sourceNode* e combiná-la com o repositório integrado de dados T_D (Linhas 2 a 4).

Cada chamada à função `transformAndMerge` procura por um nodo alcançável através do caminho *sourcePath*[*ind*] na árvore fonte T_S, e o insere em *sourceNode*[*ind*]. Quando nodos são atribuídos para todos os elementos em *sourceNode* então uma sub-árvore T' é construída através da função `createTree` e T' é inserida no repositório integrado de dados pela função `mergeTrees`, apresentada no Algoritmo 2 (Linhas 6 a 9). Na busca por um nodo para o caminho *sourcePath*[*ind*], é preciso restringir o espaço de busca para a menor sub-árvore iniciada com o nodo *anc*, que já foi visitado por algum caminho em *sourceNode*. Como o vetor *sourceNode* é ordenado por prefixo e a função `transformAndMerge` considera caminhos na ordem ascendente, existe $j < ind$ tal que $sourceNode[j] = anc$.

Observe que a raiz *anc* desta menor sub-árvore é alcançável através de $p \in sourcePath$, o qual é o prefixo mais longo de *sourcePath*[*ind*]. Assim, se $p = sourcePath[a]$ então $anc = sourceNode[a]$ (Linhas 10 a 12). Então, cada nodo v na sub-árvore com raiz em *anc* alcançável através de *sourcePath*[*ind*] é inserida em *sourceNode*[*ind*] e a função `transformAndMerge` é chamada recursivamente para preencher o vetor *sourceNode* (Linhas 23 a 26).

A Função `createTree` constrói uma árvore T' com a estrutura do repositório integrado de dados como especificado pelo mapeamento e extrai valores dos nodos em *sourceNode* para povoar folhas em T'. A função inicia criando o nodo raiz r de T'. Então, para cada nodo v em *sourceNode*, o caminho $p_D = /l_1/\ldots/l_m$ no repositório integrado de dados povoado com v é obtido de *dwPath*. Se T' já contém um nodo n_a alcançável através do caminho $/l_1/\ldots/l_a$, $a < m$ então somente nodos para o caminho $l_{a+1}/\ldots/l_m$ são gerados como descendentes de n_a (Linhas 32 a 40).

Exemplo 10 Considere novamente o mapeamento dado no Exemplo 8 e a *Fonte 1* mostrada na Figura 1.2(a). O vetor *sourcePath* gerado por este mapeamento consiste dos seguintes valores:

[/, /*item*, /*item*/*modelo*, /*item*/*fabricante*, /*item*/*cor*, /*item*/*preço*, /*nome*]. Similarmente, o vetor *dwPath* contém os seguintes caminhos: [/, /*produto*,

/produto/modelo, */produto/fabricante*, */produt/cor*, */produto/cotação/preço*, */produto/cotação/loja*].

O conteúdo do vetor *sourceNode*, após o primeiro `item` da *Fonte 1* ser percorrido pela função `transformAndMerge`, é o seguinte: [1, 1.2, 1.2.2, 1.2.1, 1.2.3, 1.2.4, 1.1]. Baseado nestes nodos e na estrutura dada por *dwPath*, a função `createTree` constrói uma árvore que consiste dos seguintes nodos: 1, 1.1, 1.1.2, 1.1.2.1, 1.1.3, 1.1.3.1, 1.1.4, 1.1.4.1.1, 1.1.5, 1.1.5.1, 1.1.5.1.1, 1.1.5.2, 1.1.5.2.1. Esses são os identificadores dos nodos no repositório integrado de dados T_D mostrado na Figura 1.2(c). □

Complexidade. Na função `transformAndMerge`, para cada nodo, o vetor *sourcePath* é percorrido para procurar o elemento com o prefixo mais longo do caminho $sourcePath[ind]$ (Linha 11). Isto leva no máximo $O(|M|)$ tempo, onde $|M|$ é o tamanho dos caminhos no mapeamento M. A obtenção do conjunto de nodos $anc[\![Q]\!]_{T_s}$ é $O(|Q||T_s|)$ [Gottlob et al. 2003]. A função `createTree` é $O(|M|)$. Para ver isso, observe que a quantidade de nodos da árvore T' gerada é no máximo $|M|$ e cada caminho do repositório integrado de dados presente em M atinge um único nodo em T'. Assim, antes da criação de um nodo basta verificar se ele já foi criado anteriormente. As funções `transformAndMerge` e `createTree` são executadas no máximo uma vez para cada nodo em T_s. Portanto, a complexidade do processo de reestruturação da fonte de dados para a estrutura do repositório integrado de dados é $O(|T_s| * (|M| + |Q||T_s| + |M|))$. Como $|Q| < |M|$, a complexidade é $O(|T_s|^2|M|)$.

3.5.2 Integração de árvores XML

O algoritmo de integração recebe duas árvores XML: o repositório integrado de dados T_D e uma árvore fonte T_s que já foi convertida para o esquema do repositório integrado de dados. O resultado é a árvore T_D com elementos em T_s inseridos de acordo com as chaves XML $\mathcal{K}$ definidas sobre o repositório integrado de dados. Isto é, sempre que nodos em T_s e T_D coincidem em seus valores chave eles são combinados em um único nodo no repositório integrado de dados; caso contrário novos elementos são criados. Conflitos de valores são representados através da geração de dois sub-elementos distintos contendo

valores de ambas as fontes.

O pseudocódigo é apresentado no Algoritmo 2. A Função `MergeTrees` verifica se o repositório integrado de dados está vazio e, se for este o caso, o nodo raiz é criado (Linhas 3 e 4). A Função `MergeNodes` é então chamada para extrair os valores de cada filho do nodo raiz da fonte e inseri-los no repositório integrado de dados (Linhas 5 e 6).

A função `MergeNodes` recebe o repositório integrado de dados T_D, um nodo $n_D \in V(T_D)$, o qual é a raiz da sub-árvore no qual elementos da sub-árvore fonte com raiz em n_s em $V(T_s)$ serão inseridos de acordo com $\mathcal{K}$. A função verifica se o nodo fonte n_s é um nodo texto ou atributo. Neste caso, a função `MergeValues` é chamada (Linhas 10 e 11). Caso contrário, considerando que cada elemento no repositório integrado de dados deve possuir uma chave definida de acordo com $\mathcal{K}$, a função procura por um nodo no repositório integrado de dados que possua valores de chave que combinem com n_s. Se tal nodo for encontrado, eles são combinados no repositório integrado de dados e a função `MergeNodes` é chamada recursivamente para juntar seus filhos; caso contrário, a sub-árvore fonte é inserida no repositório integrado de dados (Linhas 13 a 19). A Função `MergeValues` é chamada tanto para criar novos nodos atributo ou texto ou para comparar seus valores com nodos existentes, gerando nodos que apontam conflitos de valores se eles existirem. Se um nodo n_D no repositório integrado de dados, sobre o qual um nodo atributo $@a$ ou nodo texto `S` está para ser criado não possua qualquer filho com esse rótulo, um novo nodo é criado (Linhas 25 a 26).

Caso contrário, é verificado se o novo valor é igual a de algum existente no repositório integrado de dados. Se este é o caso, a informação de proveniência é inserida em T_D (Linhas 28 a 29). Se este não é o caso, então um conflito de valor foi detectado e um novo nodo com o rótulo `source` é criado para identificar o conflito (Linhas 31 a 35).

Exemplo 11 Considere as árvores XML na Figura 3.1 e as chaves XML definidas no Exemplo 4. Suponha que o repositório integrado de dados T_D já tenha sido carregado com todos os elementos da *Fonte 1*, e a *Fonte 2* está agora sendo considerada. Lembre-se que antes de chamar a função `MergeTrees`, a *Fonte 2* já foi reestruturada pelo função `Populate`, tornando a estrutura dessa fonte idêntica à estrutura do repositório integrado

Algoritmo 2 Algoritmo de integração

1: **function** MergeTrees $(T_D, T_s, \mathcal{K})$
2: $r_S \Leftarrow r(T_s)$;
3: **if** $V(T_D) = \{\}$ /* o repositório está vazio */ **then**
4: crie nodo raiz r_D de T_D; **else** $r_D \Leftarrow r(T_D)$;
5: **end if**
6: **for** $n \in children(r_s)$ **do**
7: $T_D \Leftarrow \text{MergeNodes}(T_D, r_D, T_s, n, \mathcal{K})$;
8: **end for**
9: return T_D;
10: **end function**

11: **function** MergeNodes$(T_D, n_D, T_s, n_s, \mathcal{K})$
12: $path_s \Leftarrow$ caminho da raiz de T_s até n_s;
13: **if** n_s é um nodo texto ou atributo **then**
14: $T_D \Leftarrow \text{MergeValues}(T_D, n_D, T_s, n_s, path_s)$;
15: **else**
16: $keyPaths \Leftarrow \{p \mid \mathcal{K} \models (Q_1, (Q_2, K)), path_s \in Q_1/Q_2, p \in K\}$;
17: **if** existe n na sub-árvore com raiz em n_D tal que para todo $k \in keyPaths$ $val(n.k) = val(n_s.k)$ **then**
18: **for** cada nodo $n' \in children(n_s)$ **do**
19: $T_D \Leftarrow \text{MergeNodes}(T_D, n, T_s, n', \mathcal{K})$;
20: **end for**
21: **else**
22: copie a sub-árvore com raiz em n_s para T_D;
23: insira n_s em $children(n_d)$;
24: **end if**
25: **end if**
26: return T_D;
27: **end function**

28: **function** MergeValues(T_D, n_D, T_s, n_s, p)
29: **if** n_s é um nodo atributo tal que $p = p'/@a$ **then**
30: $p_s \Leftarrow @a$; **else** $p_s \Leftarrow \texttt{S}$;
31: **end if**
32: $nodes_D \Leftarrow n_D[\![p_s]\!] \cup n_D[\![\texttt{source}/p_s]\!]$;
33: **if** $nodes_D = \{\}$ /* nenhum elemento atributo ou valor */ **then**
34: copie n_s para T_D e o insira como um filho de n_D;
35: **else**
36: **if** existe um nodo $n \in nodes_D$ tal que $val(n) = val(n_s)$ **then**
37: $val(n.@prov) \Leftarrow val(n.@prov) \cup val(n_s.@prov)$;
38: **else**
39: crie um nodo com rótulo `source` n'_s como um filho de n_D;
40: copie n_s para T_D como um filho de n'_s;
41: **if** $n_D[\![p_s]\!]_= \{n\}$ /* este é o primeiro conflito de valor */ **then**
42: crie um nodo n'_D com rótulo `source` como um filho de n_D;
43: mova n de $children(n_D)$ para $children(n'_D)$;
44: **end if**
45: **end if**
46: **end if**
47: return T_D;
48: **end function**

de dados. Quando o nodo produto (2.2.2) é processado, é conferida à sua chave dada por $K_1 : (\epsilon, (produto, \{fabricante, modelo\}))$, e verificado se algum outro nodo possui os mesmos valores de chave no repositório integrado de dados. Chamadas recursivas à função `MergeNodes` anotam as informações de proveniência nos nodos folha fabricante e modelo. Quando o nodo cor (2.2.2.2) é considerado, T_D já foi povoado com um nodo cor. Sabendo que a chave $K_3 : (//produto, (cor, \{\}))$ define que cada produto contém um único sub-elemento cor, quando a função `MergeValues` é chamada, o valor na fonte não confere com o valor que já está em T_D e, como consequência sub-elementos source são criados. Observe que se a chave K_3 não fosse definida, não existiria restrição no número de cores que um produto pode ter. Assim, o algoritmo não criaria um nodo inconsistente, mas dois sub-elementos cor, cada um originado de uma fonte. □

Complexidade. Na função `MergeNodes`, a busca pelos caminhos que fazem parte da chave de um nodo n alcançável pelo caminho $path_s$ (Linha 16) envolve o teste $paths_s \in Q_1/Q_2$, onde Q_1/Q_2 são os caminhos de contexto e alvo de uma chave em $\mathcal{K}$. Isto leva tempo $O(|path_s||Q_1/Q_2|)$ [Buneman et al. 2002]. Como $|path_s| < |T_s|$, para todas chaves em $\mathcal{K}$ este teste é $O(|T_s||\mathcal{K}|)$. Para procurar um nodo no repositório integrado de dados que contenha os mesmos valores de chave que n, o repositório integrado de dados de tamanho $|T_D|$ tem que ser percorrido. Assim, a complexidade da função `MergeNodes` é $O(|T_s||\mathcal{K}| + |T_D|)$. A função `MergeValues` tem complexidade $O(|T_D|)$. As funções `MergeNodes` e `MergeValues` são chamadas no máximo $|T_s|$ vezes. Assim, a complexidade do algoritmo de integração de dados é $O(|T_s| * ((|T_s||\mathcal{K}| + |T_D|) + T_D))$ ou $O(|T_s|^2|D|)$, onde $|D|$ é a soma do tamanho do repositório integrado de dados T_D e suas chaves $\mathcal{K}$. Com isso podemos concluir que o tempo para a carga de uma fonte de dados T_s no repositório integrado de dados D através de um mapeamento M é $O(|T_s|^2(|D| + |M|))$.

3.5.3 Reconstrução das fontes de dados

O algoritmo de reconstrução tem como objetivo extrair do repositório integrado de dados os dados provenientes de uma determinada fonte e reconstruir o documento original. Isto é feito utilizando as anotações armazenadas no repositório integrado de dados. O

Algoritmo 3 Reconstrução do documento fonte

1: **function** buildSource $(D = (T_D, K_D), source)$
2: $V(T_s) \Leftarrow \{r\}$;
3: $Leaves \Leftarrow \{n \in V(T_D) \mid n$ é um nodo folha$\}$;
4: **for** cada nodo n_D em $Leaves$ **do**
5: **if** $(id_s, p_s) \in n_D.@prov$ e $origin(id_s) = source$ **then**
6: $T_s \Leftarrow$ createPath(T_s, id_s, p_s, n_D);
7: **end if**
8: **end for**
9: return T_s;
10: **end function**

11: **function** createPath $(T, id_s, path_s, node)$
12: seja anc o nodo em T tal que. $id(anc)$ é o prefixo mais longo de id_s entre os identificadores dos nodos em T;
13: seja $id(anc)$ $j_0.j_1.\ldots.j_a$;
14: seja id_s $j_0.\ldots.j_a.j_{a+1}.\ldots.j_{a+m}$;
15: seja $path_s$ $/l_1/l_2/\ldots/l_{a+m}$;
16: **for** $i := 1$ até m **do**
17: crie nodo n_i em T como um filho de anc com rótulo l_{a+i};
18: $id(n_i) \Leftarrow j_0.\ldots.j_a.\ldots.j_{a+i}$;
19: $anc \Leftarrow n_i$;
20: **end for**
21: $val(n_m) \Leftarrow val(node)$;
22: **if** n_1 é um nodo elemento **then**
23: ordene a lista $ele(anc)$ de acordo com os identificadores dos nodos;
24: **end if**
25: return T;
26: **end function**

pseudocódigo é apresentado no Algoritmo 3.

A função `buildSource` recebe como parâmetros de entrada o repositório integrado de dados D e um identificador de fonte *source*. Inicialmente, o algoritmo cria o nodo raiz do documento fonte (Linha 2) e então determina o conjunto (*Leaves*) com todos os nodos folhas no repositório integrado de dados. Para cada nodo neste conjunto, é verificado se a anotação @*prov* contém um par (id_s, p_s), tal que id_s é um identificador de um nodo em *source* (Linhas 4 e 5). Se é este o caso, a função `createPath` é chamada para gerar um novo nodo em T_s alcançável através do caminho p_s.

A função `createPath` recebe como entrada: uma árvore parcialmente construída T representando a fonte de dados; um identificador id_s na forma $j_0.j_1 \ldots j_b$; um caminho $path_s$ no documento fonte na forma $/l_1/\ldots/l_b$, e um nodo folha *node* no repositório integrado de dados povoado com um nodo de *source*. Seja n_S esse nodo da fonte. Então $id(n_s) = id_s$ e, no documento original T_s, $n_s \in [\![path_s]\!]_{T_s}$.

Observe que a partir de id_s e $path_s$ é possível construir todo o caminho da fonte do nodo raiz levando a n_s. Isto é, este caminho percorre os nodos $n_1, \ldots n_b$, tal que para cada n_i, $id(n_i) = j_0 \ldots j_i$ e $label(n_i) = l_i$. Antes de criar estes nodos é necessário verificar se algum deles já foi criado (Linhas 9 a 12), e então gerar os nodos remanescentes (Linhas 13 a 16). O valor do último nodo no caminho é extraído do nodo *node* no repositório integrado de dados. Para manter a ordem relativa dos elementos no documento fonte, a lista de elementos filhos do pai do nodo que acabou de ser criado é ordenada de acordo com os identificadores Dewey (Linhas 18 a 19).

Exemplo 12 Considere a reconstrução de *Fonte 2* a partir do repositório integrado de dados T_D mostrado na Figura 1.2(c). Suponha que o primeiro nodo folha considerado em T_D é o nodo 1.1.3.1 em $[\![/produto/modelo/S]\!]_{T_D}$. Observe que 1.1.3.1.@$prov$ = $\{(1.2.2.1, /item/modelo/S), (2.2.2.1.1, /categoria/produto/modelo/S)\}$. Dado que existe um identificador do nodo que inicia com o identificador 2 da *Fonte 2*, são criados novos nodos 2.2 (um nodo *categoria*), 2.2.2 (um nodo *produto*), 2.2.2.1 (um nodo *modelo*) e 2.2.2.1.1 (um nodo *S*), e é atribuído o valor "0605041" ao último nodo. Considere que o próximo nodo folha no repositório integrado de dados a ser considerado é o nodo 1.1.1.1

em $[\![/tipo/S]\!]_{T_D}$. Observe que $1.1.1.1.@prov = (2.2.1, /categoria/@nome)$. Dado que o nodo 2.2 com rótulo *categoria* já foi criado, o algoritmo cria somente um novo filho *@nome* para este nodo. □

Complexidade. A complexidade do algoritmo. de reconstrução é $O(|T_D||M|)$. Para ver isso, basta observar que a obtenção dos elementos folhas do repositório integrado de dados provenientes de uma determinada fonte leva $O(|T_D|)$ tempo e que para cada nodo obtido, um caminho de tamanho $O(|M|)$ é construído.

3.6 Resumo

Neste capítulo foi apresentado o modelo de dados do sistema de integração proposto nesta dissertação. O modelo de dados é caracterizado pelas funções previstas em um repositório integrado de dados que armazena documentos XML. O esquema do repositório integrado de dados é fixo e pré-determinado, podendo ser representado como uma árvore XML. O esquema das fontes de dados pode ser distinto do esquema do repositório integrado de dados. No entanto, um conjunto de regras de mapeamento deve ter sido definido, como apresentado na Seção 3.3. Esse mapeamento determina as regras de conversão do esquema das fontes de dados para o esquema do repositório integrado de dados. Quando múltiplas fontes têm seus dados integrados, é necessário definir uma técnica de identificação de entidades. Nesta dissertação é utilizada a técnica baseada em chaves XML. Além disso, podem existir entidades que são identificadas unicamente através das chaves XML, mas que possuem valores conflitantes entre as diversas fontes. Para tratar essa situação, nesta dissertação foi utilizada uma técnica de representação explícita de inconsistências, definida na Seção 3.2. Outra característica importante do modelo é a capacidade de poder reconstruir as fontes de dados armazenadas a partir das anotações feitas nos elementos oriundos dessas fontes e armazenados no repositório integrado de dados. O processo de reconstrução das fontes foi descrito na Seção 3.4. Finalmente, na Seção 3.5 foram apresentados os algoritmos que determinam como as funcionalidades desse modelo de dados podem ser implementadas.

O próximo capítulo apresenta brevemente uma implementação dos algoritmos introdu-

zidos aqui e um estudo experimental que quantifica as principais vantagens da utilização desse modelo de dados num cenário real.

CAPÍTULO 4

ESTUDO EXPERIMENTAL

Neste capítulo, a implementação dos algoritmos propostos é brevemente apresentada na Seção 4.1. Na Seção 4.2 são apresentados os resultados de alguns estudos sobre o funcionamento dos algoritmos desenvolvidos.

4.1 Implementação dos Algoritmos

Os algoritmos foram implementados com a linguagem Java, utilizando a ferramenta JDom [Hunter 2000], que utiliza o SAX [Murray-Rust 1997] como *parser* XML.

Na próxima seção são apresentados alguns experimentos realizados com a ferramenta implementada.

4.2 Experimentos

Uma característica interessante da utilização do modelo de dados desta dissertação é a possibilidade de serem definidos níveis de reestruturação dos documentos XML antes de eles serem armazenados no repositório integrado de dados. Em linhas gerais, o processo de reestruturação dos documentos das fontes de dados retira elementos de um nível da árvore da fonte de dados e os insere em outro. Uma reestruturação interessante, e que é o objeto de estudo dos experimentos desta seção, é aquela em que elementos que se repetem em larga escala nas fontes de dados sejam colocados em um nível acima, agrupando os elementos a que eles anteriormente pertenciam. A Figura 4.1 ilustra esse processo, no qual os elementos destacados foram identificados como elementos com muitas repetições (Figura 4.1(a)), retirados do nível em que estão, e agrupados num nível superior (Figura 4.1(b)). Foi constatado que esse processo reduz o espaço de armazenamento utilizado pela base de dados integrada, e, também diminui substancialmente o tempo necessário para a

criação dessa base.

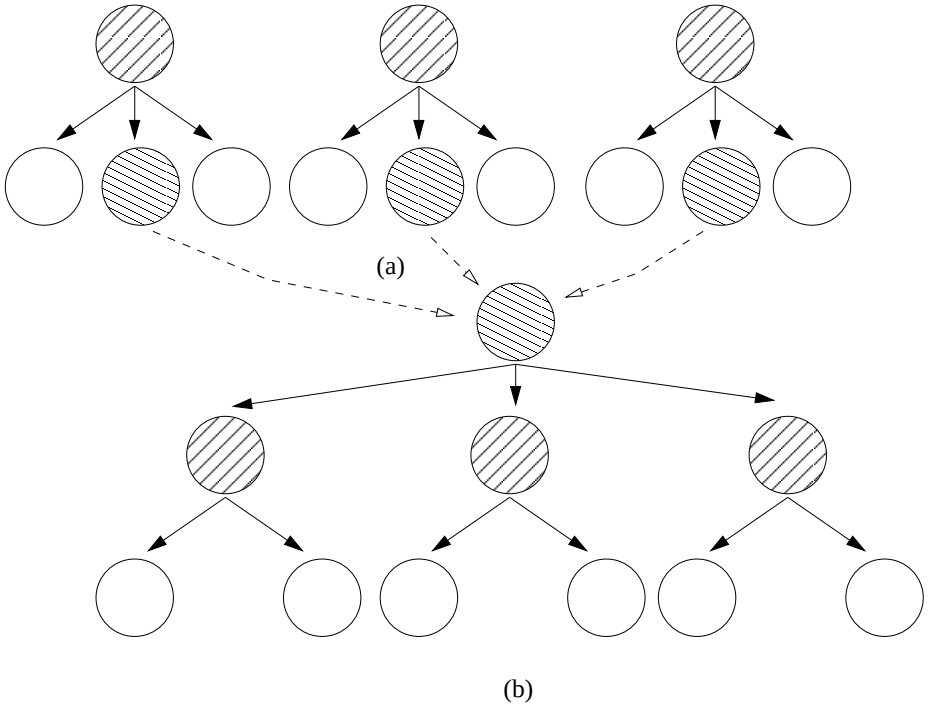


Figura 4.1: Processo de reestruturação e agrupamento de dados de documentos XML

Nas próximas seções são analisados os experimentos realizados para detectar o espaço de armazenamento que o modelo proposto consegue economizar (Seção 4.2.1), o tempo de criação do repositório integrado de dados na presença de reestruturação das fontes com a utilização de chaves XML (Seção 4.2.2) e o tempo de reconstrução das fontes a partir dos repositórios integrados de dados gerados (Seção 4.2.3). Todos os experimentos foram realizados numa máquina com processador Pentium 4 de 3.0GHz, com 1GB de memória RAM, utilizada exclusivamente para esses testes.

4.2.1 Espaço de armazenamento

A capacidade do sistema de reconstruir a porção do documento fonte armazenada no repositório integrado de dados dispensa o armazenamento de todo o documento fonte. Entretanto, as anotações nas quais o algoritmo de reconstrução é baseado acarretam em um aumento do custo de armazenamento. O objetivo deste experimento é quantificar esse custo adicional.

Observe que no modelo proposto no Capítulo 3, o tamanho do repositório integrado

de dados que possui dados importados de duas ou mais fontes depende da quantidade de dados sobrepostos que ele contém. Para que essa característica não afete os resultados, neste experimento é gerado um repositório integrado de dados a partir de uma única fonte de dados. Foi utilizado o repositório DBLP (*Digital Bibliography Library Project*) [Ley 1997]. O DBLP é uma fonte de informações bibliográficas envolvendo periódicos e eventos da ciência da computação. O experimento foi executado com seis documentos extraídos do DBLP com diferentes tamanhos. O repositório integrado de dados foi criado com mapeamentos "completos", isto é, cada valor texto na fonte é extraído e inserido no repositório integrado de dados. Como resultado, o tamanho do repositório integrado de dados reportado no experimento é o cenário de pior caso, no qual anotações são criadas para cada elemento extraído da fonte e nenhuma combinação de dados é realizada.

```
<dblp O="1">
        <article>
                <author>Frank Manola</author>
                <title>Object Data Model Facilities for Multimedia Data Types.</title>
                <journal>GTE Laboratories Incorporated</journal>
                <year>1990</year>
        </article>
        <article>
                <author>Farshad Nayeri</author>
                <title>Experiments with Dispatching in a Distributed Object.</title>
                <journal>GTE Laboratories Incorporated</journal>
                <year>1993</year>
        </article>
             .
             .
             .
```

(a) Repositório DBLP

```
<publs>
  <article>
    <author S="1.1.1" P="2">Frank Manola</author>
    <title S="1.1.2" P="3">Object Data Model Facilities for Multimedia Data Types.</title>
    <journal S="1.1.3" P="4">GTE Laboratories Incorporated</journal>
    <year S="1.1.4" P="5">1990</year>
  </article>
  <article>
    <author S="1.2.1" P="2">Farshad Nayeri</author>
    <title S="1.2.2" P="3">Experiments with Dispatching in a Distributed Object.</title>
    <journal S="1.2.3" P="4">GTE Laboratories Incorporated</journal>
    <year S="1.2.4" P="5">1993</year>
  </article>
      .
      .
      .
```

(b) Repositório integrado com anotações

Figura 4.2: (a) Pequena parte do repositório DBLP e (b) uma porção dos dados com anotações.

A Figura 4.2(a) mostra uma pequena porção dos dados do repositório DBLP. A Figura 4.2(b) apresenta os dados XML após a conversão num esquema semelhante, com as anotações de proveniência, que são usadas para a reconstrução da fonte de dados. Note

que na porção dos dados DBLP, o nodo raiz possui o atributo O (de *Origin*) que identifica a fonte de dados (na Figura, a fonte de dados é identificada pelo número 1). Já nos dados com anotações são utilizados dois atributos, que são *id* (de *source id*), representado por um S na Figura, que tem como valor a ordem Dewey do nodo na fonte, e P (de *path*) que identifica o caminho ao qual esse nodo pertence na fonte de dados. O atributo P é um identificador de caminho no conjunto de regras do mapeamento. Foi escolhido utilizar apenas um identificador e armazenar separadamente o conjunto de caminhos pois muitos elementos no repositório integrado de dados são originários dos mesmos caminhos. Assim a implementação do modelo de dados não sobrecarrega o repositório integrado de dados com informações repetitivas.

A Tabela 4.1 apresenta os resultados do experimento. A coluna *Nodos* mostra o número de elementos na fonte de dados, enquanto a coluna *Nodos Folha* contém a quantidade de nodos folha na fonte. A coluna *DW e Fonte* mostra o tamanho aproximado da fonte de dados combinada com o repositório integrado de dados *sem anotações*. A coluna *DW Anotado* apresenta o tamanho aproximado do repositório integrado de dados com as fontes armazenadas de acordo com o modelo proposto, com as folhas anotadas com informação de proveniência. A coluna *% Salvo* mostra que, em média, *DW Anotado* é 26% menor que *DW e Fonte*.

Tabela 4.1: Tamanho da fonte de dados e do repositório integrado de dados.

Nodos	Nodos Folha	(1) DW e Fonte	(2) DW Anotado	% Salvo
2.401	1.920	204KB	147KB	28%
4.801	3.840	406KB	293KB	28%
9.601	7.680	812KB	590KB	27%
14.401	11.520	1,2MB	887KB	26%
19.201	15.360	1,6MB	1,2MB	25%
24.001	19.200	2MB	1,5MB	25%

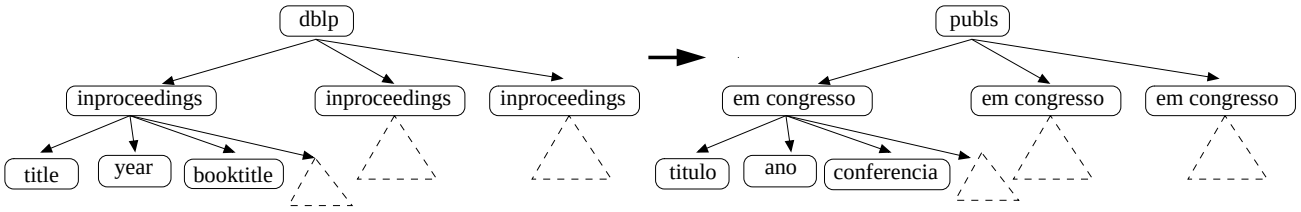
É importante notar que em todos os conjuntos de dados aproximadamente 80% dos nodos são folhas. Considerando que no modelo proposto somente nodos folha são anotados, essa característica traz um impacto maior no tamanho de *DW Anotado*. Apesar de a proposta prover uma redução significativa no custo de armazenamento, é possível reduzi-lo ainda mais aplicando-se um algoritmo de compressão nos identificadores dos

nodos. Um mecanismo de compressão para uma codificação similar à Ordem Dewey foi proposta em [O'Neil et al. 2004], a qual pode ser usada neste modelo.

O experimento mostra que este modelo reduz o custo de armazenamento de um repositório integrado de dados em no mínimo 26% quando as fontes de dados originais precisam estar disponíveis para futuras comparações. Este é o cenário do pior caso, pois, em repositórios integrados de dados reais, dados podem ser filtrados e fontes podem conter dados sobrepostos. Esses processos podem melhorar o percentual de espaço de armazenamento salvo. Na próxima seção será analisado o tempo de criação do repositório integrado de dados na presença de operações de reestruturação.

4.2.2 Tempo de criação do repositório com reestruturação

Este experimento tem como objetivo mensurar o tempo que o algoritmo leva para reestruturar e agrupar os dados de um repositório integrado de dados, através da definição de chaves e mapeamentos diferenciados. Foi utilizado novamente o repositório DBLP como fonte de dados. A Figura 4.3 apresenta a estrutura original do repositório DBLP (4.3(a)) e uma conversão feita sem reestruturação dos dados (4.3(b)). Para tanto, foi definida uma única chave para esse mapeamento, que é a chave que identifica uma publicação: `(emcongresso,{título})`



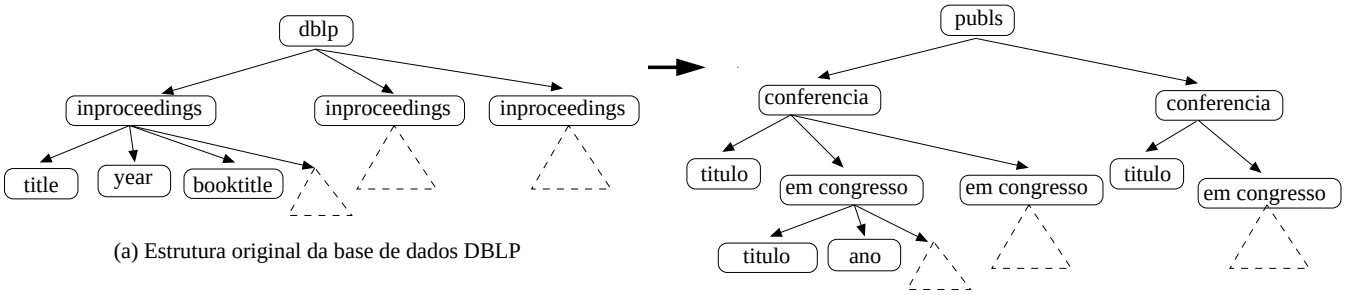
(a) Estrutura original da base de dados DBLP

(b) Estrutura convertida sem reestruturacao

Figura 4.3: Conversão dos dados da base dados DBLP para uma estrutura semelhante.

No teste seguinte, foi inserido mais um nível na estrutura convertida. Para tanto, foram definidos os mapeamentos adicionais $/conferencia/titulo \leftarrow /inproceedings/booktitle$ e $/conferencia/emcongresso \leftarrow /inproceedings$, foi criada a chave *(conferencia,{título})* e alterada a chave que identifica uma publicação para *(conferencia,(em congresso,{título}))*. A Figura 4.4 ilustra a conversão do repositório DBLP. A idéia dessa reestruturação é

agrupar todas as publicações que possuam a mesma `conferência` numa única sub-árvore. Por exemplo, se em todas as publicações houver 10 conferências diferentes, a raiz `publs` contém 10 sub-árvores `conferência`.

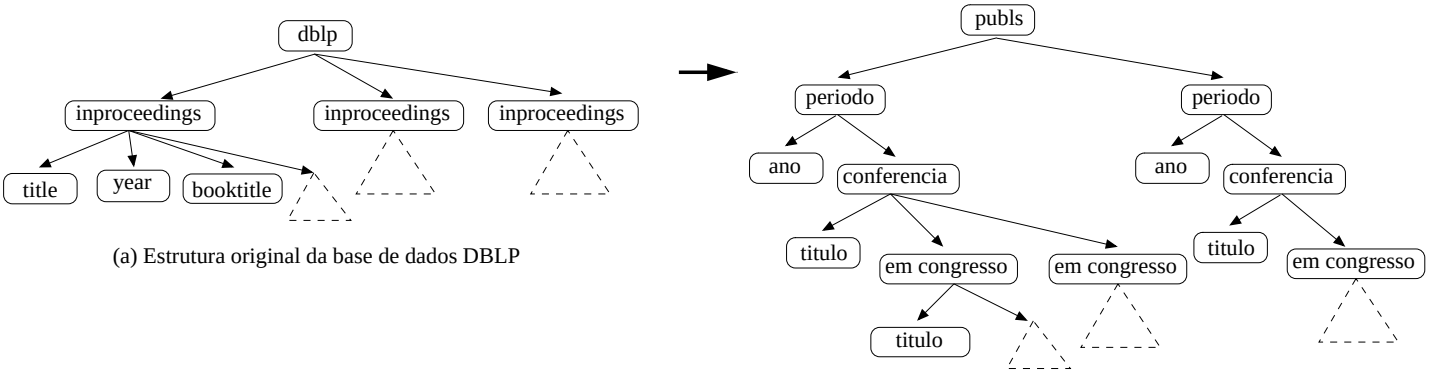


(a) Estrutura original da base de dados DBLP

(b) Publicacoes agrupadas por livro

Figura 4.4: Reestruturação e agrupamento em 1 nível do repositório DBLP.

E, finalmente mais um nível foi inserido na estrutura convertida. Mais alguns mapeamentos foram definidos, além dos definidos no exemplo anterior, que são $/periodo/ano \leftarrow /inproceedings/year$ e $/periodo/conferencia/emcongresso \leftarrow /inproceedings$, foi criada a nova chave *(periodo,*$\{ano\}$*)* e a chave que identifica um conferência foi alterada para *(periodo,(conferência,*$\{título\}$*)).* A conversão do repositório DBLP ficou conforme apresenta a Figura 4.5. Essa reestruturação agrupa todos as conferências dentro do ano em que suas `publicações` foram publicadas. Agora, a raiz `publs` possui sub-árvores de `período`, que, por sua vez, possui sub-árvores `conferência`.



(a) Estrutura original da base de dados DBLP

(b) Publicacoes agrupadas por ano e livro

Figura 4.5: Reestruturação e agrupamento em 2 níveis da base de dados DBLP.

Após a definição dessas conversões e reestruturações, foi mensurado o tempo que o algoritmo leva para realizar esses processos, conforme o número de nodos da fonte de

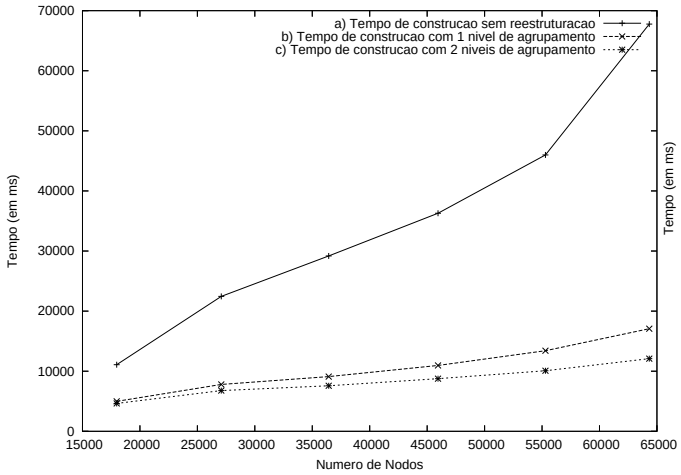


Figura 4.6: Tempo de carga do repositório integrado de dados com reestruturações.

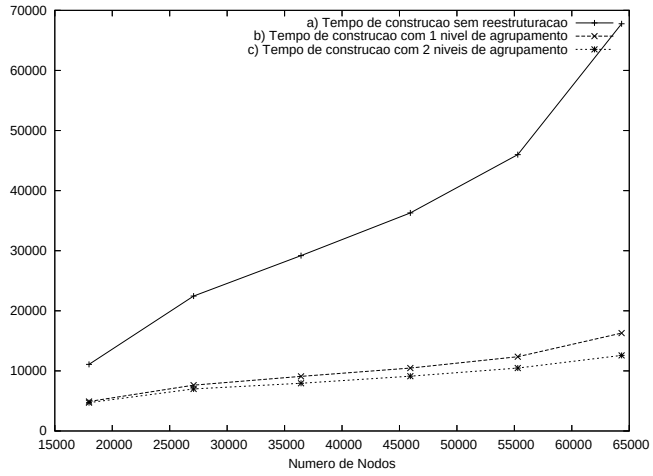


Figura 4.7: Tempo de carga do repositório integrado de dados com reestruturações invertidas.

dados aumentavam. A Figura 4.6 mostra a evolução do tempo para um número crescente de nodos.

Note que a primeira conversão, em que a estrutura não sofre alteração, leva um tempo muito maior para ser executada do que as conversões onde ocorrem reestruturações. Quando ocorre a reestruturação de 1 nível, o tempo para efetuar o agrupamento do mesmo número de nodos pode chegar a aproximadamente 25% do tempo que o algoritmo leva para simplesmente traduzir a estrutura da base de dados. Já no caso em que é realizada a reestruturação de 2 níveis, o tempo que o algoritmo leva pode chegar a 20% do tempo que o algoritmo leva para traduzir a estrutura sem agrupamento.

Esses testes apresentaram resultados bons devido à qualidade do agrupamento. Isso se deve ao fato de as publicações disponíveis no DBLP serem de anos muito próximos e, também, de relativamente poucas conferências. Então, quando são realizados agrupamentos por `conferência` ou `ano`, são criadas poucas sub-árvores com essas raízes e o espaço de pesquisa das chaves é drasticamente diminuído. Para verificar qual seria o agrupamento melhor, se por `conferência` ou `ano`, foram invertidos os mapeamentos e realizados os testes de tempo mostrados na Figura 4.7. Nesse caso, ao invés de primeiro agrupar os dados por `conferência`, os dados foram agrupados por `ano`. E, posteriormete, os dados foram agrupados num nível abaixo, por `conferência`. Desta forma, o agrupamento de dois níveis colocou todas as publicações abaixo de `conferências` e agrupou

as `conferências` pelo ano de publicação das `publicações`. Note que os resultados são muito semelhantes, sendo que o agrupamento de 1 nível por `ano` leva ligeira vantagem sobre o agrupamento de 1 nível por `conferência`. Já no agrupamento de 2 níveis, ambos os resultados são bastante parecidos.

Nessa base de dados, entre todas as informações disponíveis, os agrupamentos por `conferência` e `ano` são os melhores possíveis. Outros agrupamentos apenas fariam a reestruturação criando muito mais nós do que os existentes na base original. Para efeitos de comparação, foram realizados testes com os mesmos dados agrupando-os por uma informação diferente: o nome do autor. Foi constatado que há autores que possuem mais de uma publicação. Entretanto, este número não se compara ao número de publicações dentro de um ano específico, por exemplo. Assim, o agrupamento foi efetuado em um nível bem menor e o tempo gasto, bem maior. Em média, para agrupar a base de dados pelo nome do autor, o tempo gasto foi 65% do tempo gasto para simplesmente traduzir a estrutura da fonte de dados.

Assim, foi mostrado que quanto maior o nível de reestruturação, com a criação de grandes agrupamentos, menor é o tempo que o algoritmo proposto leva para criar a estrutura do repositório integrado de dados. Isso se deve a utilização das chaves XML, que propiciam um tempo de busca reduzido nos diversos níveis da árvore. Na próxima seção é analisado o tempo que o algoritmo de reconstrução leva para construir as fontes de dados originais a partir dos dados gravados no repositório integrado de dados.

4.2.3 Tempo de reconstrução das fontes de dados

Neste experimento, foram utilizados os repositórios integrados de dados gerados a partir das fontes de dados utilizadas na Seção 4.2.2. O processo de reconstrução toma por base o repositório integrado de dados e os mapeamentos que converteram as fontes no esquema do repositório integrado de dados. No entanto, os mapeamentos são utilizados no formato inverso: $p_S \leftarrow p_D$. A Figura 4.8 apresenta a evolução do tempo do processo de reconstrução das fontes de dados armazenadas no repositório integrado de dados conforme o número de nodos dessas fontes aumentam.

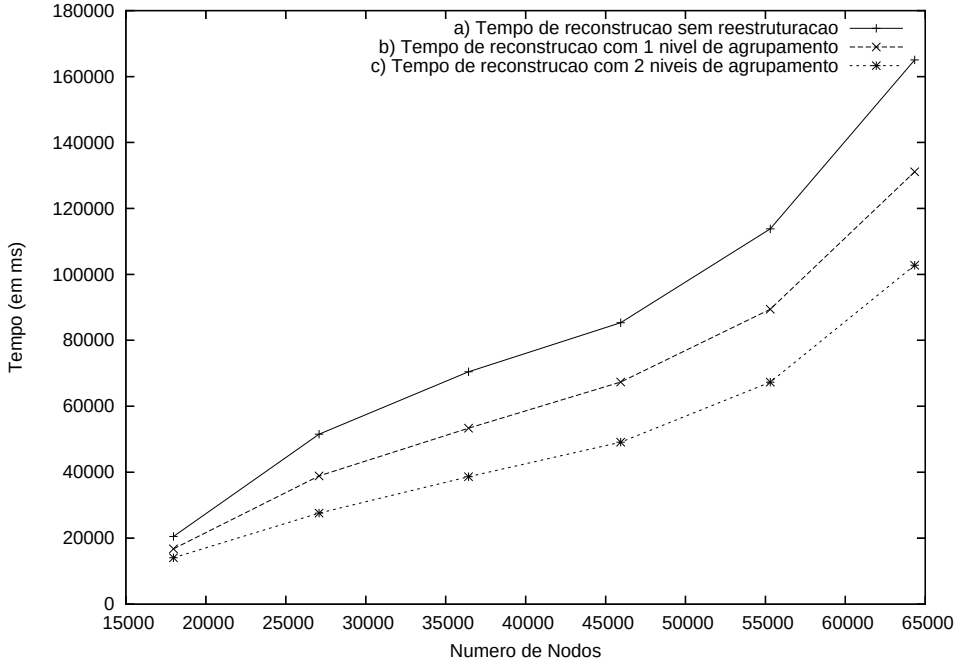


Figura 4.8: Tempo de reconstrução de fontes de dados conforme o número de nodos aumenta.

Na linha (a) do gráfico as fontes não foram reestruturadas para serem gravadas no repositório integrado de dados. Na linha (b) do gráfico as fontes foram agrupadas em 1 nível e na linha (c) do gráfico as fontes sofreram agrupamento de 2 níveis.

Note que a evolução dos tempos segue uma função semelhante. O processo de reconstrução do repositório integrado de dados com 1 nível de agrupamento é, em média, 20% mais rápido que o processo de reconstrução do repositório integrado de dados criado sem agrupamento. Os resultados também mostraram que no processo de 2 níveis de agrupamento o algoritmo é também, em média, 20% mais rápido para reconstruir a fonte de dados do que no processo com 1 nível de agrupamento e, em média, 40% mais rápido que o processo em que não houve reestruturação.

Observando a maneira como o algoritmo foi implementado, e também o resultado dos experimentos realizados, conclui-se que o tempo que o algoritmo leva para realizar a reconstrução está diretamente ligado ao número de elementos que o sistema deve gerenciar em memória, visto que não são utilizadas chaves para a localização dos elementos. O número de elementos no repositório integrado de dados que sofreu reestruturação é menor, pois, o processo de reestruturação retirou de cada elemento `publicação` o elemento

`conferência` na primeira reestruturação, e o elemento `ano` na segunda reestruturação. Conforme o número de elementos cresce, essa diferença torna-se mais visível, ou seja, o número de elementos em memória é menor.

4.3 Resumo

Neste capítulo foi apresentada uma descrição da implementação dos algoritmos propostos no Capítulo 3. Foram apresentados estudos experimentais para mensurar a economia de espaço que o modelo proposto pode obter através da anotação de dados de proveniência e da ordenação dentro do documento XML, ao invés de armazenar todo o documento XML original. O ganho de espaço de armazenamento, no pior caso, onde não há um processo de reestruturação das fontes, é de 26%. Também foram estudados os tempos de carga de um repositório integrado de dados e o tempo de reconstrução de fontes de dados armazenadas. Quando há um processo de reestruturação das fontes, o tempo de carga do repositório integrado de dados chega a ser 20% do tempo de carga quando não há reestruturação. O tempo de reconstrução das fontes está fortemente ligado ao número de elementos que o algoritmo precisa controlar, sendo que, quando os dados das fontes passaram por reestruturação, o número de elementos armazenados tende a diminuir em relação a uma inclusão das fontes sem reestruturação. No caso da reestruturação do documento fonte, o tempo de reconstrução é aproximadamente 20% menor para cada nível de agrupamento pelo qual a fonte passou.

CAPÍTULO 5

CONCLUSÃO

Neste capítulo é feita a conclusão deste trabalho. Na Seção 5.1 é feita uma revisão do trabalho apresentado nesta dissertação. Na Seção 5.2 são sugeridos alguns trabalhos futuros que podem complementar a proposta apresentada.

5.1 Revisão do Trabalho

Neste trabalho foi apresentado um modelo de integração de documentos XML que utiliza abordagens existentes na literatura para alcançar o objetivo de integração de documentos XML. Foram estudadas as abordagens existentes na literatura referentes a integração de instâncias, identificação de entidades, proveniência de dados, anotações em elementos e limpeza de dados.

O modelo proposto utiliza como ferramenta de identificação de elementos a proposta de chaves XML definida em [Buneman et al. 2001]. Através dessa abordagem é possível identificar unicamente nodos em documentos XML. Da mesma forma, chaves XML definidas na base de dados integrada determinam quando dados de diversas fontes que possuem informações sobre a mesma entidade do mundo real devem ser integrados.

A abordagem de integração utilizada é semelhante à abordagem de integração de dois níveis, definida em [la Fontaine 2002]. Foi definida uma linguagem de mapeamento que, juntamente com algoritmos de conversão, gera um documento XML com a estrutura do repositório integrado de dados carregado com os dados da fonte de dados. Os dados presentes nessa estrutura podem então ser comparados com os dados presentes no repositório integrado de dados. Nesse passo, através das chaves definidas para a estrutura do repositório integrado de dados, o sistema consegue determinar se elementos podem compartilhar dados de diversas fontes ou se novos elementos devem ser criados.

Quando novas versões de uma fonte já armazenada são disponibilizadas, o sistema

provê o processo de reconstrução para que a fonte atualizada seja comparada com os dados já armazenados. Esse processo de reconstrução utiliza a linguagem de mapeamento definida na Seção 3.3 e anotações feitas nos elementos armazenados. Essas anotações seguem o padrão de Ordenação *Dewey* [Tatarinov et al. 2002], que é um padrão de anotação que define a ordem dos elementos dentro da árvore XML. Através desse sistema de anotações, é possível reconstruir o documento fonte com a mesma ordem que ele possuía originalmente. Isto facilita o processo de comparação entre os dados da fonte atualizada e os dados da fonte armazenada.

Finalmente, foram realizados testes de desempenho dos processos de inserção de elementos no repositório integrado de dados e o processo de reconstrução das fontes de dados. Os experimentos mostraram que quanto maior o nível de reestruturação que as fontes passam antes de ser inseridas no repositório integrado de dados, menor é o tempo necessário para que tais dados sejam inseridos. Já os experimentos do processo de reconstrução das fontes mostraram que quando há reestruturação das fontes de dados armazenadas no repositório integrado de dados, o tempo de reconstrução dessas fontes é menor se comparado ao tempo de reconstrução das fontes sem reestruturação.

A próxima seção apresenta algumas sugestões de trabalhos futuros que podem complementar o modelo de dados proposto nessa dissertação.

5.2 Trabalhos Futuros

Diversos trabalhos que podem complementar a proposta apresentada nesta dissertação podem ser citados. Dentre eles, destacam-se:

- Uma ferramenta que auxilie o processo de remoção de inconsistências de dados: Como a representação das inconsistências no modelo proposto é explícita e de forma padronizada, uma ferramenta de remoção de inconsistências pode ser implementada utilizando, por exemplo, uma abordagem de atribuição de níveis de confiança a cada fonte que atualiza a base de dados integrada [Menestrina et al. 2006] e ontologias [Wache et al. 2001].

- Revisão do algoritmo de reconstrução: O processo de remoção de inconsistências provocará atualizações no repositório integrado de dados. Neste caso, o algoritmo de reconstrução das fontes de dados deve ser revisado para levar em consideração tais alterações.

- Criação de um *log* de alterações: O armazenamento das alterações comumente realizadas para serem utilizadas em outras situações é outro trabalho a ser investigado.

- Tradução da linguagem de mapeamento para *XQuery* [Chamberlin 2003]: Os algoritmos propostos e as operações de mapeamentos podem ser traduzidos para a linguagem *XQuery*, viabilizando a utilização desse modelo em diversos outros ambientes.

- Ferramenta ETL: É desejável o desenvolvimento de um sistema semi-automatizado que contemple todo o processo de extração e atualização do repositório integrado de dados com dados provenientes de diversas fontes.

REFERÊNCIAS BIBLIOGRÁFICAS

[Abiteboul 1999] Abiteboul, S. (1999). On views and XML. In *Proceedings of the eighteenth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS)*, páginas 1–9.

[Batini et al. 1986] Batini, C., Lenzerini, M., e Navathe, S. B. (1986). A comparative analysis of methodologies for database schema integration. *ACM Computing Surveys*, 18(4):323–364.

[Bernstein e Bergstraesser 1999] Bernstein, P. A. e Bergstraesser, T. (1999). Meta-data support for data transformations using Microsoft repository. *IEEE Data Engineering Bulletin*, 22(1):9–14.

[Bitton e DeWitt 1983] Bitton, D. e DeWitt, D. J. (1983). Duplicate record elimination in large data files. *ACM Transactions on Database Systems (TODS)*, 8(2):255–265.

[Bray et al. 1998] Bray, T., Paoli, J., Sperberg-McQueen, C. M., Maler, E., e Yergeau, F. (1998). *Extensible Markup Language (XML) 1.0*. World Wide Web Consortium (W3C). `http://www.w3.org/TR/REC-xml`.

[Buneman et al. 2006] Buneman, P., Chapman, A., e Cheney, J. (2006). Provenance management in curated databases. In *Proceedings of the 2006 ACM SIGMOD International Conference on Management of Data (SIGMOD)*, páginas 539–550.

[Buneman et al. 2001] Buneman, P., Davidson, S., Wenfei, F., Hara, C., e Tan, W. (2001). Keys for XML. *Proceedings of the International World Wide Web Conference (WWW)*, páginas 201–210.

[Buneman et al. 2002] Buneman, P., Davidson, S. B., Fan, W., Hara, C. S., e Tan, W. C. (2002). Reasoning about keys for XML. In *Revised Papers from the 8th International Workshop on Database Programming Languages (DBPL)*, páginas 133–148.

[Buneman et al. 2004] Buneman, P., Khanna, S., Tajima, K., e Tan, W. (2004). Archiving scientific data. *ACM Transactions on Database Systems (TODS)*, 29(1):2–42.

[Chamberlin 2003] Chamberlin, D. D. (2003). Xquery: A query language for XML. In *Proceedings of ACM SIGMOD International Conference on Management of Data (SIGMOD)*, página 682.

[Chan e Mitchell 2003] Chan, L. M. e Mitchell, J. S. (2003). *Introduction to the Dewey Decimal Classification*. `http://www.oclc.org/oclc/fp/about/about_the_ddc.htm`.

[Clark 1999] Clark, J. (1999). XSL transformations (XSLT). `http://www.w3.org/TR/xslt`.

[Clark e DeRose 1999] Clark, J. e DeRose, S. (1999). XML Path Language (XPath). World Wide Web Consortium (W3C). `http://www.w3.org/TR/xpath`.

[Cui e Widom 2000] Cui, Y. e Widom, J. (2000). Practical lineage tracing in data warehouses. *Proceedings of the 16th International Conference on Data Engineering (ICDE)*, página 367.

[Davidson e Liefke 2001] Davidson, S. B. e Liefke, H. (2001). Creating and maintaining curated view databases. *Knowledge Discovery and Data Mining in Biological Databases*.

[Draper et al. 2001] Draper, D., HaLevy, A. Y., e Weld, D. S. (2001). The Nimble XML data integration system. In *Proceedings of the International International Conference on Data Engineering (ICDE)*, páginas 155–160.

[Etzold e Argos 1993] Etzold, T. e Argos, P. (1993). SRS: An indexing and retrieval tool for flat file data libraries. *Computer Applications of Biosciences (CABIOS)*, páginas 49–57.

[Fallside 2000] Fallside, D. C. (2000). *XML Schema Part 0: Primer*. World Wide Web Consortium (W3C). `http://www.w3.org/TR/xmlschema-0/`.

[Galhardas et al. 2000] Galhardas, H., Florescu, D., Shasha, D., e Simon, E. (2000). E.: Declaratively cleaning your data using AJAX. In *Journées Bases de Données Avancées (BDA)*.

[Garcia-Molina et al. 1997] Garcia-Molina, H., Papakonstantinou, Y., Quass, D., Rajaraman, A., Sagiv, Y., Ullman, J., e Widom, J. (1997). The TSIMMIS approach to mediation: Data models and languages. *Journal of Intelligent Information Systems*, 8(2):117–132.

[Gottlob et al. 2003] Gottlob, G., Koch, C., e Pichler, R. (2003). The complexity of XPath query evaluation. In *Proceedings of the twenty-second ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS)*, páginas 179–190.

[Haas 1990] Haas, L. e. a. (1990). Starburst mid-flight: As the dust clears. *IEEE Transactions on Knowledge and Data Engineering*, 2(1):143–160.

[Hernandez et al. 1998] Hernandez, M. A., Stolfo, S. J., e Fayyad, U. (1998). Real-world data is dirty: Data cleansing and the merge/purge problem. *Data Mining and Knowledge Discovery*, 2(1):9–37.

[Hopcroft e Ullman 1979] Hopcroft, J. E. e Ullman, J. D. (1979). *Introduction to Automata Theory, Languages and Computation*. Addison Wesley.

[Hunter 2000] Hunter, J. (2000). *JDOM*. `http://www.jdom.org`.

[ISO 2004] ISO (2004). Standard generalized markup language (sgml). `http://www.iso.ch/cate/d16387.html`.

[la Fontaine 2002] la Fontaine, R. (2002). Merging XML files: a new approach providing intelligent merge of XML data sets. In *Proceedings of XML Europe*.

[Lee et al. 1998] Lee, T., Bressan, S., e Madnick, S. (1998). Source attribution for querying against semi-structured documents. *Workshop on Web Information and Data Management, CIKM*, páginas 33–39.

[Lenzerini 2002] Lenzerini, M. (2002). Data integration: a theoretical perspective. In *Proceedings of ACM Symposium on Principles of Database Systems (PODS)*, páginas 233–246.

[Ley 1997] Ley, M. (1997). *XML Schema Part 0: Primer*. Universitat Trier. `http://dblp.uni-trier.de`.

[Lim et al. 1996] Lim, E.-P., Srivastava, J., Prabhakar, S., e Richardson, J. (1996). Entity identification in database integration. *Information Sciences: an International Journal*, 89(1-2):1–38.

[Menestrina et al. 2006] Menestrina, D., Benjelloun, O., e Garcia-Molina, H. (2006). Generic entity resolution with data confidences. In *Proceedings of the International VLDB Workshop on Clean Databases*, páginas 25–32.

[Murray-Rust 1997] Murray-Rust, P. (1997). *The SAX Project*. `http://www.saxproject.org`.

[Nascimento e Hara 2008] Nascimento, A. M. e Hara, C. (2008). A model for XML instance level integration. *Anais do XXIII Simposio Brasileiro de Banco de Dados (SBBD)*, páginas 46–60.

[O'Neil et al. 2004] O'Neil, P., O'Neil, E., Pal, S., Cseri, I., Schaller, G., e Westbury, N. (2004). ORDPATHs: Insert-friendly XML node labels. In *Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data (SIGMOD)*, páginas 903–908.

[Papakonstantinou et al. 1995] Papakonstantinou, Y., Garcia-Molina, H., e Widom, J. (1995). Object exchange across heterogeneous information sources. In *Proceedings of the Eleventh International Conference on Data Engineering (ICDE)*, páginas 251–260.

[Pemberton et al. 2002] Pemberton, S., Austin, D., Axelsson, J., Çelik, T., Dominiak, D., Elenbaas, H., Epperson, B., Ishikawa, M., Matsui, S., McCarron, S., Navarro, A., Peruvemba, S., Relyea, R., Schnitzenbaumer, S., e Stark, P. (2002). *Document Type*

Definitions. World Wide Web Consortium (W3C). `http://www.w3.org/TR/xhtml1/dtds.html`.

[Poggi e Abiteboul 2005] Poggi, A. e Abiteboul, S. (2005). XML data integration with identification. In *Proceedings of International Workshop on Database Programming Languages (DBPL)*, páginas 106–121.

[Pokorný 2002] Pokorný, J. (2002). XML data warehouse: Modelling and querying. In *Proceedings of the Baltic Conference (BalticDB&IS)*, páginas 267 – 280.

[Pottinger e Bernstein 2002] Pottinger, R. A. e Bernstein, P. A. (2002). Creating a mediated schema based on initial correspondences. In *Proceedings of the International International Conference on Data Engineering (ICDE)*, páginas 155–160.

[Prabhakar et al. 1993] Prabhakar, S., Richardson, J., Srivastava, J., e Lim, E.-P. (1993). Instance-level integration in federated autonomous databases. páginas 62–69.

[Ragget et al. 1999] Ragget, D., Hors, A. L., e Jacobs, I. (1999). Hypertext markup language 4.01 specification. `http://www.w3.org/TR/html4/`.

[Rahm e Do 2000] Rahm, E. e Do, H. H. (2000). Data cleaning: Problems and current approaches. *IEEE Data Engineering Bulletin*, 23(4):3–13.

[Sawires et al. 2005] Sawires, A., Tatemura, J., Po, O., Divyakant, A., e Candan, K. (2005). Incremental maintenance of path-expression views. In *Proceedings of the 2005 ACM SIGMOD International Conference on Management of Data (SIGMOD)*, páginas 443–454.

[Suciu 1999] Suciu, D. (1999). Managing web data. In *Proceedings of the 1999 ACM SIGMOD International Conference on Management of Data (SIGMOD)*, página 510.

[Tan 2007] Tan, W.-C. (2007). Provenance in databases: Past, current, and future. *IEEE Data Engineering Bulletin*, 30(4):3–12.

[Tatarinov et al. 2002] Tatarinov, I., Viglas, S. D., Beyer, K., Shanmugasundaram, J., Shekita, E., e Zhang, C. (2002). Storing and querying ordered XML using a relational database system. In *Proceedings of the 2002 ACM SIGMOD International Conference on Management of Data (SIGMOD)*, páginas 204–215.

[W3C 1994] W3C (1994). World wide web consortium. `http://www.w3.org/`.

[Wache et al. 2001] Wache, H., Vögele, T., Visser, U., Stuckenschmidt, H., Schuster, G., Neumann, H., e Hübner, S. (2001). Ontology-based integration of information - a survey of existing approaches. In *Proceedings of the IJCAI-01 workshop*, páginas 108–117.

[Wong 2000] Wong, L. (2000). Kleisli, a functional query system. *Journal of Functional Programming*, 10(1):19–56.

[Woodruff e Stonebraker 1997] Woodruff, A. e Stonebraker, M. (1997). Supporting fine-grained data lineage in a database visualization environment. *Proceedings of the International International Conference on Data Engineering (ICDE)*, páginas 91–102.