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

VINCENT NELSON KELLERS DA SILVEIRA

X-Spread - Um mecanismo automático para propagação da evolução de esquemas para documentos XML

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

Prof. Dra. Renata de Matos Galante
Orientadora

Porto Alegre, setembro de 2007

CIP – CATALOGAÇÃO NA PUBLICAÇÃO

Silveira, Vincent Nelson Kellers da

X-Spread - Um mecanismo automático para propagação da evolução de esquemas para documentos XML / Vincent Nelson Kellers da Silveira. – Porto Alegre: PPGC da UFRGS, 2007.

107 f.: il.

Dissertação (mestrado) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação, Porto Alegre, BR–RS, 2007. Orientadora: Renata de Matos Galante.

1. XML. 2. Esquemas. 3. Evolução. I. Galante, Renata de Matos. II. Título.

“I may not have gone where I intended to go,
but I think I have ended up where I needed to be.”
— Douglas Adams

AGRADECIMENTOS

Estendo meus agradecimentos a todas as pessoas que demonstraram interesse neste trabalho. Agradeço também pelo envolvimento de Augusto Perini e Rodrigo Lumertz, responsáveis pela implementação de diferentes porções da especificação do X-Spread. Agradecimentos especiais a Renata Galante, pela orientação fundamental no desenvolvimento deste trabalho. As idéias e soluções desenvolvidas neste trabalho não seriam completas sem seus conselhos. Agradeço ao Instituto de Informática e à Universidade Federal do Rio Grande do Sul pela acolhida nos últimos noves anos.

Agradeço à equipe da ADP Labs pelo apoio durante o desenvolvimento deste trabalho: a Diego Banda, pelas dicas sobre Linux. A Júlio Gerchman, por outras dicas sobre Linux e pelos cafés. A Felipe Suslik, pela amizade e por permitir, com seu trabalho de qualidade, minha dedicação quase total na conclusão deste trabalho. A Fabiano Pereira, pelas oportunidades de crescimento oferecidas.

Aos meus amigos, Gian, Lilian e Enguer. Sou grato por poder dividir bons momentos com vocês nos últimos anos e por vocês compreenderem minhas ausências recentes.

A Carla, pelo apoio constante. Obrigado por estar sempre do meu lado, mesmo quando eu não sabia o rumo exato a seguir. Obrigado por sua constante presença: ela me leva a aproveitar as oportunidades oferecidas pela vida, e me torna uma pessoa melhor.

A Beatriz, por ser minha maior mestra na vida. Somente seu apoio e orientação me possibilitaram trilhar o caminho que me trouxe até este momento.

SUMÁRIO

LISTA DE ABREVIATURAS E SIGLAS

API	*Application Programming Interface*
DOM	*Document Object Model*
DTD	*Data Type Definition*
SAX	*Simple API for XML*
SGBD	Sistema de Gerenciamento de Banco de Dados
SGML	*Standard Generalized Markup Language*
XML	*Extensible Markup Language*
XSL	*Extendible Stylesheet Language*
W3C	*World Wide Web Consortium*

LISTA DE FIGURAS

LISTA DE TABELAS

RESUMO

Assim como as aplicações, as bases de dados evoluem ao longo do tempo. Esta evolução ocorre em função de alterações de cunho técnico ou por alterações na realidade modelada pela base de dados. Bases de dados semiestruturados, compostas por esquemas e documentos XML, são afetadas por esta evolução de uma maneira diversa daquela observada em bases de dados relacionais. Modificações em esquemas podem levar bases de dados semiestruturados a um estado inconsistente, pois as instâncias podem tornar-se incompatíveis com as definições mais recentes dos esquemas. Em bancos de dados relacionais, modificações que levem a base a um estado inválido são bloqueadas pelo sistema gerenciador. Em bases de dados semiestruturados sem um sistema gerenciador, modificações no esquema não podem ser bloqueadas em função das instâncias existentes, em função da ausência do gerenciador.

Trabalhos acadêmicos e comerciais na área de evolução de esquemas XML modelam diferentes aspectos deste processo, mas usualmente não abordam o efeito que a evolução do esquema possui sobre as instâncias existentes. As soluções propostas para este problema usualmente demandam intervenção do administrador do banco de dados na adaptação das instâncias ou a utilização de uma interface específica para edição do esquema.

Este trabalho especifica o X-Spread, um mecanismo automático para propagação de modificações em esquemas para documentos XML. O X-Spread monitora periodicamente esquemas XML, e ao identificar modificações em um esquema, inicia a revalidação dos documentos que referenciam o esquema modificado. A revalidação analisa somente as porções dos documentos correspondentes às porções modificadas no esquema. Documentos considerados inválidos no processo de revalidação serão submetidos ao processo de adaptação, a fim de tornarem-se novamente compatíveis com as definições do esquema.

A arquitetura proposta para o X-Spread permite o seu emprego em diferentes cenários de utilização de bases de dados semiestruturados. A mesma arquitetura proposta para revalidação e adaptação de documentos XML é aplicável a mensagens XML trocadas entre aplicações através de uma rede de dados. Ao administrador do banco de dados semiestruturado é oferecida a flexibilidade de utilização de qualquer ferramenta desejada para edição do esquema, bem como a possibilidade de restringir conforme suas necessidades o conjunto de operações de revalidação e adaptação executadas pelo X-Spread.

A principal contribuição do X-Spread é a modelagem de todas as fases do processo de evolução de esquemas e dos processos de revalidação e adaptação de documentos XML. O processos definidos aplicam-se a documentos armazenados em um servidor de arquivos e a documentos transmitidos entre diferentes aplicações através de uma rede de dados.

Palavras-chave: XML, esquemas, evolução.

X-Spread - An automatic mechanism for propagation of schema evolution to XML documents

ABSTRACT

Like applications, databases evolve as time goes by. Evolution can occur due to technical changes or due to changes in the modeled reality. Semistructured databases, composed by schema and XML documents, are not affected by evolution as relational databases. Changes to semistructured schema can lead the database to an inconsistent state, since instances can become invalid with respect to the most recent schema definitions. Modifications that lead the database to an inconsistent state are blocked by the management system of relational databases. Semistructured databases with no management system can not have this kind of modification blocked due to the lack of management system.

Academic and commercial research on XML schema evolution models different evolution aspects, but usually the effect that evolution has on existing database instances is not investigaed. When this effect is subject of investigation, the proposed solutions usually require database administrator intervention in the instance adaptation process or usage of an specific schema update interface.

This work specifies X-Spread, an automatic mechanism for propagation of schema modification to XML documents. X-Spread periodically monitors XML schemata, and upon identification of changes, revalidation of documents with references to the changed schema is started. Revalidation takes into account only items in the document equivalent to items modified in the schema. Documents considered invalid during the revalidation process are subjected to the adaptation process, in order to become once again valid with respect to the schema.

The X-Spread architecture supports different scenarios of semistructured databases usage. This architecture can also be used for revalidation and adaptation of XML messages exchanged by applications over a network. To the database administrator is offered the flexibility of choice on schema edit tool, as well as the ability to restrict the set of operations executed by X-Spread during the revalidation and adaptation processes.

The main contribution of X-Spread is the modeling of all schema evolution phases as well as the XML document revalidation and adaptation processes modeling. These processes here defined are suitable to XML documents stored in a file server as well as XML messages exchanged by application over a network.

Keywords: XML, schemata, evolution.

1 INTRODUÇÃO

Sistemas de informação evoluem ao longo do tempo por diferentes razões. Esta evolução pode ser desencadeada em função de alterações de cunho legal, expansão de funcionalidades, modificação de requisitos técnicos ou manutenções. A evolução afeta diferentes aspectos da aplicação, podendo demandar alterações também no esquema da base de dados empregada pelo sistema.

Sistemas de informação distintos necessitam de tipos distintos de bases de dados, sendo que tais bases evoluem de maneira particular. Bases de dados semiestruturados apresentam um processo de evolução diferente daquele observado em bases relacionais. Compostas por esquemas e documentos XML, o processo de evolução de bases de dados semiestruturados difere em relação à evolução de bases relacionais devido à disjunção entre o esquema e o conjunto de documentos XML.

Em bases relacionais, as instâncias pertencentes a um esquema são determinadas através do sistema gerenciador do banco de dados. Em bases de dados semiestruturados apenas as instâncias, que podem encontrar-se em diferentes nodos de rede, referenciam o esquema e usualmente não há a figura de um sistema gerenciador. A evolução do esquema pode introduzir inconsistências na base de dados, posto que não há como impôr restrições na evolução em função das instâncias existentes, como ocorre em bases relacionais.

Um exemplo de evolução de esquema consiste na modificação de estruturas de dados complexas através da inclusão de novos elementos para armazenamento de informações adicionais da realidade modelada pelo esquema. Da mesma maneira, a simplificação de estruturas complexas, através da remoção de determinados componentes desta estrutura, consiste em uma evolução do esquema. Considerando-se um esquema fictício para um catálogo de livros, construído de maneira a armazenar informações básicas como nome do livro e seu autor, este passa por um processo de evolução quando da inclusão de estruturas para identificação das traduções existentes para os livros.

Diversas técnicas podem ser adotadas no sentido de modelar o processo de evolução de esquemas e o impacto que a evolução possui em documentos, de acordo com o contexto em que a aplicação é utilizada. Para esquemas cujo escopo de utilização é pequeno e bem conhecido, modificações nos esquemas seguidas de correções nos próprios documentos que referenciam tais esquemas podem ser aceitáveis, desde que o número de documentos a corrigir seja baixo. Já para esquemas cujas instâncias apresentam alto grau de distribuição, tal abordagem não pode ser adotada, pois normalmente a totalidade do conjunto de documentos XML que referenciam o esquema não é conhecida com antecedência. Logo, determinados documentos podem tornar-se inválidos, pois após a modificação no esquema, não foram alterados de maneira a respeitar a nova estrutura definida pelo esquema.

Um exemplo de inconsistêncina introduzida na base de dados pela evolução de esque-

mas é observado quando um elemento de cardinalidade obrigatória é inserido no esquema. Todas as instâncias da base de dados que não possuírem um elemento correspondente a este elemento serão consideradas inválidas. A fim de devolver documentos inválidos a um estado de conformidade em relação ao esquema, elementos correspondentes ao elemento obrigatório do esquema devem ser inseridos nos documentos.

O problema da evolução de esquemas e seu impacto nos documentos XML é abordado por diferentes propostas acadêmicas ((GUERRINI; MESITI; ROSSI, 2005), (AL-JADIR; EL-MOUKADDEM, 2004), (BERTINO et al., 2002)), através de restrições às modificações aplicáveis ao esquema ou mecanismos disponibilizados ao administrador do banco de dados para suporte à adaptação de documentos quando da evolução de esquemas XML.

Implementações de bancos de dados comerciais, como o Tamino (SOFTWAREAG, 2006) e o Oracle (ORACLE, 2004), abordam o impacto da evolução de esquemas XML sobre documentos XML através do suporte a diferentes versões de esquema e análise de modificações permitidas na base de dados em função das instâncias existentes.

Tanto a organização que define o formato XML e seus esquemas, a *World Wide Web Consortium* (W3C), quanto propostas acadêmicas e comerciais não definem uma abordagem amplamente aceita para os diferentes cenários deste problema, nem uma abordagem que modele todas as etapas da evolução de esquemas.

A ausência de definições amplamente aceitas para o problema da propagação da evolução de esquemas para documentos XML é a motivação para o desenvolvimento deste trabalho. Em diferentes cenários de uso de esquemas e documentos XML, tanto acadêmicos quanto comerciais, há a necessidade de propagação de modificações de esquemas para documentos, sendo que esta necessidade não é atendida completamente pelas soluções existentes.

Este trabalho especifica o X-Spread, um mecanismo automático para propagação da evolução de esquemas para documentos XML. O principal objetivo do X-Spread é reestabelecer a consistência de uma base de dados semiestruturados quando da evolução do esquema, através de adaptações nos documentos tornados inválidos pelas modificações realizadas no esquema. Os processos de revalidação e adaptação dos documentos são desencadeados a partir da detecção de modificações no esquema, o que é obtido através da monitoração periódica deste artefato. A evolução dos esquemas deve ocorrer de maneira independente do X-Spread, ou seja, o administrador do banco de dados não deve depender do X-Spread para realizar as modificações no esquema. A revalidação e adaptação dos documentos XML devem ocorrer sem intervenção do administrador do banco de dados, de maneira a permitir ao X-Spread a execução autônoma de suas tarefas.

A fim de atingir o objetivo de manter a consistência de bases de dados semiestruturados que passam por evolução, uma classificação de modificações realizadas em esquemas XML, bem como o impacto que estas modificações possuem nos documentos XML é apresentada. A partir desta análise, o processo de adaptação de documentos XML é modelado, com a especificação de quais alterações devem ser realizadas em documentos XML tornados incompatíveis com um esquema em função de modificações no esquema.

A partir destas definições, a arquitetura necessária para implementação deste mecanismo é definida. O protótipo correspondente à implementação desta arquitetura é apresentado, sendo seu desempenho comparado em relação a demais pacotes de revalidação de documentos XML. A partir da análise do desempenho dos processos de revalidação e adaptação do X-Spread, sua aplicabilidade quanto a diferentes cenários de documentos e esquemas XML é determinada.

A principal contribuição do X-Spread é baseada na modelagem de todas as fases do

processo de evolução de esquemas e nos processos de revalidação e adaptação, aplicáveis a documentos armazenados em um servidor de arquivos e a documentos XML transmitidos entre diferentes aplicações através de uma rede de dados. Também o conjunto de documentos XML e as portas de comunicação de um dado servidor sobre os quais o X-Spread deve operar são configuráveis pelo administrador do banco de dados.

O restante do texto encontra-se assim organizado: o capítulo 2, *Definições conceituais*, introduz conceitos e idéias utilizados no restante do trabalho. Os conceitos apresentados abordam dados e esquemas semiestruturados, bem como detalhes sobre o processo de evolução destes esquemas.

O capítulo 3, *Revisão bibliográfica*, apresenta as características de trabalhos de pesquisa e de implementações comerciais de bancos de dados na área de evolução de esquemas XML. A partir da definição de critérios de avaliação, estes trabalhos são comparados entre si e suas características analisadas.

O capítulo 4, *X-Spread*, define um mecanismo de suporte à evolução de esquemas e adaptação de documentos XML. A partir da definição de abstrações para documentos e esquemas XML, operações de modificação de esquema são definidas, bem como o impacto que estas operações possuem nos documentos. A partir destas definições, o processo de adaptação de documentos XML é apresentado.

O capítulo 5, *Arquitetura e implementação do X-Spread*, define a arquitetura necessária do X-Spread e apresenta as características do protótipo do mecanismo. Estas definições são seguidas pela apresentação de experimentos onde o desempenho do protótipo do X-Spread e avaliado em relação a demais pacotes de revalidação de documentos XML.

Por fim, o capítulo 6, *Conclusões e trabalhos futuros*, revisa as contribuições do trabalho realizado e apresenta possibilidades de trabalhos futuros.

2 DEFINIÇÕES CONCEITUAIS

Este capítulo apresenta conceitos pertinentes ao processo de evolução de esquemas XML e à propagação desta evolução para os documentos XML que referenciam estes esquemas.

As próximas seções definem documentos XML, esquemas XML, conceitos e aplicabilidade da evolução de esquemas XML, bem como técnicas relacionadas à evolução de esquemas e a adaptação de documentos, como revalidação de documentos, sistemas observadores e detecção de diferenças em esquemas XML.

2.1 Documentos semiestruturados e artefatos relacionados

Definido pela W3C (*World Wide Web Consortium*) em 1998, o XML (*Extensible Markup Language*) atualmente encontra-se em sua quarta definição, publicada em 2006 (W3C, 2006). Derivado do SGML (*Standard Generalized Markup Language*), o formato XML originalmente destinava-se à publicação de dados na *web*, mas atualmente é utilizado tanto para armazenamento de dados quanto suporte para troca de mensagens entre aplicações heterogêneas.

Esquemas XML são estruturas de dados que visam restringir tanto o formato quanto os valores existentes em documentos XML. Uma vez que um documento XML possui referências aos esquemas por ele utilizados, ao contrário de bancos de dados relacionais, um esquema XML não possui conhecimento sobre o número total de documentos que o referenciam. Ainda, como os documentos XML podem referenciar esquemas disponibilizados em uma rede de dados, o conjunto formado por um esquema e um documento XML possui aspecto distribuído. Ao conjunto formado por um esquema XML e um conjunto de documentos com referência a este esquema, dá-se o nome de base de dados semiestruturados (COHEN et al., 1999).

Os dois principais formatos para representação de esquemas XML na atualidade, tanto em função do número de projetos de pesquisa quanto ao número de projetos comerciais que os utilizam (BEX; NEVEN; BUSSCHE, 2004), são DTD (*Data Type Definition*) e XML Schema.

O DTD (W3C, 1998) caracteriza-se por definir hierarquicamente a estrutura de um documento XML. Ao permitir a definição dos valores contidos nos atributos de elementos de um documento, o DTD permite uma estruturação básica da informação. Uma característica importante do formato DTD reside no fato de que sua representação não é feita em formato XML. Ou seja, um esquema DTD não é considerado um documento XML.

O XML Schema (W3C, 2004) caracteriza-se como uma evolução do DTD. Definido pela W3C, o XML Schema é representado em formato XML e permite a estruturação hierárquica de documentos XML, com poder de expressividade maior quando comparado

ao DTD. Somando-se a este fato, XML Schema ainda permite a validação de valores de elementos e atributos de documentos XML através da especificação de expressões regulares.

As estruturas de dados representadas em formatos DTD e XML Schema, bem como os dados representados em formato XML são passíveis de processos de evolução devido a mudanças na realidade modelada ou alterações em requisitos técnicos, por exemplo. O restante deste capítulo define com maiores detalhes técnicos e conceituais o processo de evolução destes artefatos, as conseqüências da evolução e as abordagens adotadas para tratamento destas conseqüências.

2.2 Evolução de esquemas

A natureza de sistemas de informações é dinâmica. Uma aplicação evolui ao longo do tempo por diferentes razões, como modificações ou refinamentos nos seus requisitos em função de alterações na realidade modelada, definição de novos requisitos ou mesmo em função de correções de problemas de codificação detectados na aplicação.

De forma similar, as bases de dados utilizadas por estas aplicações encontram-se sujeitas a este processo de evolução. No caso de bases de dados, tanto os esquemas quanto os próprios dados podem evoluir.

Conforme definido em (JENSEN; et al., 1998), a evolução de esquema constitui um processo em que modificações no esquema são permitidas sem que ocorra perda dos dados existentes na base de dados e sem suporte à coexistência de esquemas criados anteriormente. Ainda, conforme definido em (CAMOLESI; TRAINA, 1996), a evolução de esquemas ocorre em diferentes níveis:

- projeto - aborda o conceito de modelo de dados, ou seja, os conceitos sobre esquemas de bancos de dados e a metodologia utilizada para produção de tais esquemas. Como modificações neste nível afetam a própria estrutura da base de dados, um ciclo de redesenho dos diferentes esquemas de dados e das próprias aplicações que operam sobre estes esquemas pode ser iniciado;

- físico - aborda a evolução do esquema físico da base de dados e suas conseqüências, como reestruturação e reformatação das instâncias;

- lógico - aborda a evolução do esquema físico e conceitual, além da evolução do esquema externo, ou seja, as visões do usuário. Refere-se às operações de alteração do esquema lógico, às técnicas de evolução empregadas e às estratégias de propagação das atualizações às instâncias.

A evolução do esquema ainda pode ser classificada em diferentes níveis a fim de preservar a consistência da base de dados:

- representação de esquema - identificação dos aspectos do esquema que devem sofrer modificações;

- operações de modificação de esquema - definição das operações de modificação permitidas para o esquema;

- consistência na modificação de esquema - refere-se à consistência interna do esquema em relação às modificações neste realizadas;

- integridade do banco de dados - identificação da forma como as instâncias vigentes no banco de dados são afetadas frente à modificação do esquema;

- disponibilidade do banco de dados - especificação da capacidade da base de dados de operar durante a execução do processo de evolução.

Conforme descrito em (GALANTE, 2003), a evolução de uma base de dados ainda pode ser classificada de acordo com as diferentes estruturas do banco de dados abordadas pela evolução. As diferentes estruturas abordadas por esta classificação dizem respeito à evolução da estrutura e à evolução do comportamento da base de dados:

- evolução estrutural - as modificações são realizadas nas estruturas de dados representadas no esquema. Neste caso, modificações são realizadas no conjunto de atributos modelados por uma determinada estrutura ou ainda, afetam o relacionamento de composição, referência ou extensão estabelecido entre as diferentes estruturas descritas em um esquema;

- evolução comportamental - as modificações são realizadas no conjunto de métodos definidos em estruturas de dados contidas em bancos de dados objeto-relacionais ou orientados a objetos.

A evolução estrutural ou comportamental de um esquema pode afetar a integridade de uma base de dados e a integridade no relacionamento entre uma base de dados e aplicações que a acessam. A evolução de determinadas estruturas em um esquema pode torná-lo internamente inconsistente por diferentes razões, como por exemplo, conflitos na nomeação de estruturas ou referência a estruturas ou métodos não existentes.

Uma base de dados pode tornar-se inconsistente em função da evolução do esquema quando modificações em estruturas do esquema são incompatíveis com os dados já existentes. Exemplos deste tipo de inconsistência ocorrem quando estruturas do esquema têm sua cardinalidade alterada ou quando os tipos de dados das propriedades de uma estrutura são alterados para tipos incompatíveis com os valores associados a estas propriedades nas instâncias existentes na base de dados.

Já a relação de integridade estabelecida entre um esquema de dados e aplicações pode ser afetada quando estruturas de dados ou métodos definidos no esquema tornam-se incompatíveis com a representação interna das aplicações para estes artefatos. Esta inconsistência pode ser introduzida pelas mesmas razões que causam a inconsistência em uma base de dados, como alterações na estrutura interna de tipos definidos em um esquema, ou mesmo pela completa exclusão de estruturas ou métodos definidos no esquema e referenciados pela aplicação.

Conforme ilustrado na Figura 2.1, o processo usual de evolução de bases de dados é composto por modificações no esquema que podem ser seguidas por propagação destas mudanças aos dados e adequação das aplicações a estas modificações.

Usualmente as modificações em esquemas, tanto estruturais quanto comportamentais, são realizadas pelo administrador da base de dados através de interfaces de administração do bancos de dados. Estas interfaces oferecem ao administrador da base de dados um conjunto de ferramentas a fim de que modificações nesta base sejam realizadas de maneira simples, rápida e principalmente, transparente ao usuário final.

O segundo processo envolvido na evolução de bancos de dados, a propagação de modificações nos esquemas aos dados a fim de que se evitem inconsistências na base de dados, pode ser executado também pelo administrador da base de dados. Este processo

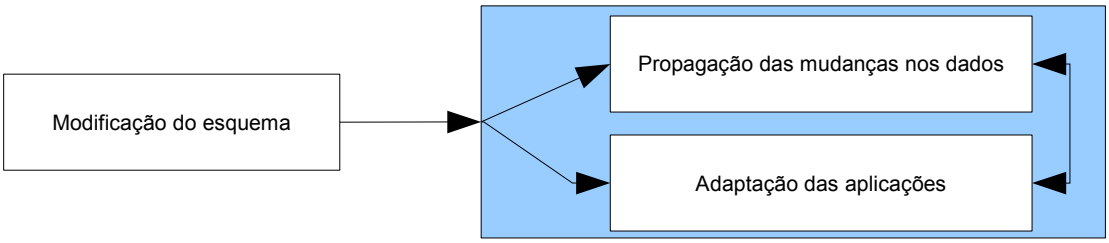

Figura 2.1: Passos envolvidos no processo de evolução de um esquema de dados

também é realizado através de interfaces de administração, com a execução de séries de comandos que atualizam os dados e os tornam compatíveis com o novo formato do esquema. Estes comandos realizam a alteração do formato e do tipo dos dados existentes ou a especificação de valores anteriormente inexistentes em determinados registros.

A propagação de modificações nos esquemas aos dados pode ocorrer em dois momentos distintos: imediatamente após a modificação do esquema, onde a propagação de modificações é parte da transação de modificação dos esquemas ou em momento posterior. Quando realizada imediatamente após a modificação no esquema, tem-se que quaisquer modificações realizadas no esquema implicarão pelo menos em tentativas de adaptações das instâncias. Quando a propagação das modificações ocorre em momento posterior, tem-se que seqüências de modificações nos esquemas são refletidas nas instâncias. A propagação das modificações aos dados pode ser realizada tanto pelo administrador da base de dados, através da execução de comandos em uma interface de administração ou pelo próprio usuário final da base de dados, caso as aplicações disponibilizadas ao usuário estejam consistentes em relação às modificações realizadas no esquema.

Também pode-se caracterizar a disponibilidade das aplicações durante o processo de propagação das modificações nos esquemas. A propagação pode ser *off-line*, onde todas as aplicações de usuário tem o acesso à base de dados bloqueado. Neste caso, somente o processo de propagação das modificações tem acesso às instâncias. A alternativa à propagação *off-line* é a propagação *on-line*, onde a base de dados pode ser acessada pelas aplicações de usuário durante o processo de propagação de modificações. No caso de propagação de modificações *on-line*, é necessária a implementação de um controle de concorrência, a fim de evitar conflito de edições realizadas entre aplicações de usuário e o processo responsável pela propagação de modificações realizadas no esquema.

O terceiro processo relacionado a modificações em esquemas de dados, a adequação das aplicações disponibilizadas ao usuário final às modificações na base de dados, é necessariamente realizada pelo desenvolvedor da aplicação. As modificações realizadas pelos desenvolvedores da aplicação podem permitir ao usuário a conclusão do processo de adaptação de dados existentes, que pode não ter sido finalizado pelo administrador da base de dados uma vez que este usualmente não possui conhecimento aprofundado sobre o domínio e regras específicas da aplicação.

A seção 2.2.1 define características de bancos de dados semiestruturados, refina os conceitos apresentados sobre evolução de esquemas, agora no contexto semiestruturado e introduz definições e características próprias da evolução de esquemas neste contexto.

2.2.1 Evolução de esquemas XML

Um outro tipo de dado, dito semiestruturado, vem sendo crescentemente utilizado nos sistemas de informações desenvolvidos recentemente, seja para armazenamento de dados

em si, seja para modelagem de mensagens trocadas entre aplicações potencialmente heterogêneas. Embora também sujeito a processos de evolução, a natureza de base de dados semiestruturados difere das características observadas em bases de dados relacionais em função de sua distribuição e administração.

No que refere-se à distribuição de bases de dados semiestruturados, as instâncias podem encontrar-se espalhadas por diferentes nodos de uma rede de dados. E ao contrário do que ocorre em bases de dados tradicionais, são as instâncias que possuem as referências aos esquemas que definem suas estruturas internas. Decorre desta última característica o fato de que esquemas semiestruturados não possuem conhecimento sobre quantas instâncias o referenciam, pois por definição os esquemas semiestruturados não contêm referências a suas instâncias.

A segunda característica importante destas bases de dados decorre da ausência de um sistema gerenciador da base de dados: quando uma estrutura de dados é modificada, é necessário propagar esta modificação aos dados existentes a fim de garantir a consistência da base de dados. Em bancos de dados relacionais, modificações que levam a base de dados a um estado inconsistente são bloqueadas pelos sistemas gerenciadores. Considerando-se que bancos semiestruturados usualmente não possuem gerenciadores, quaisquer modificações no esquema são permitidas, tornem elas a base de dados inconsistentes ou não. Fica a cargo do próprio usuário a propagação das modificações realizadas em uma estrutura de dados a todos os registros que dela fazem uso, seja com o auxílio de aplicações próprias para este fim ou manualmente. Há de se observar que a propagação destas modificações pode ocorrer em um número não determinado de instâncias, dada a natureza disjunta destas bases de dados.

Por fim, esquemas XML não associam métodos a estruturas de dados definidas em esquemas, tal como ocorre em bases de dados objeto-relacionais e orientadas a objetos. Desta maneira, a evolução comportamental de esquemas XML diz respeito não a métodos, mas sim a conjuntos de aplicações que operam com estruturas de dados definidas nestes esquemas. Como aplicações que operam com bases de dados semiestruturados são disjuntas em relação à base de dados, elas precisam de alguma maneira ser sinalizadas sobre modificações na estrutura dos dados que podem levar à inconsistência das aplicações.

Considerando-se as características apresentadas, tem-se que o processo de evolução de estruturas modeladas e armazenadas por bancos de dados semiestruturados difere daquele observado em bancos de dados tradicionais, uma vez que este pode ser seguido de um processo de adaptação das aplicações que operam com estas bases e também dos dados que as compõem. Em bases de dados relacionais, usualmente a propagação de modificações para dados não existiria uma vez que modificações que levassem a base a um estado inconsistente seriam bloqueadas pelo gerenciador da base de dados.

No que concerne à dimensão de projeto da base de dados semiestruturados, além de modificações em esquemas XML definidos de acordo com as especificações usuais deste tipo de artefato, conceitos de tempo de validade ou de transação, ou ainda, ambos os conceitos podem ser definidos em esquemas XML. Nestes casos, a evolução dos esquemas pode ocorrer de diferentes maneiras: através do armazenamento do aspecto temporal da evolução de um esquema ou através do versionamento destes artefatos.

O aspecto temporal da evolução do esquema diz respeito ao armazenamento do histórico de modificações realizadas no esquema. Conforme definido em (JENSEN; et al., 1998), a história de um elemento do banco de dados, no caso o seu esquema, diz respeito a uma seqüência de operações de modificação associadas a este esquema. Tal histórico poderia ser armazenado junto à própria definição do esquema, utilizando-se uma repre-

sentação baseada no modelo paramétrico de dados (NOH; GADIA, 2005) a fim de capturar os conceitos de tempo de validade ou de transação, ou ainda, ambos os conceitos de tempo (SNODGRASS, 2000). O armazenamento de marcas de tempo de transação e/ou validade permite a identificação do processo de evolução seguido por um esquema, bem como o planejamento de futuras modificações na estrutura de dados.

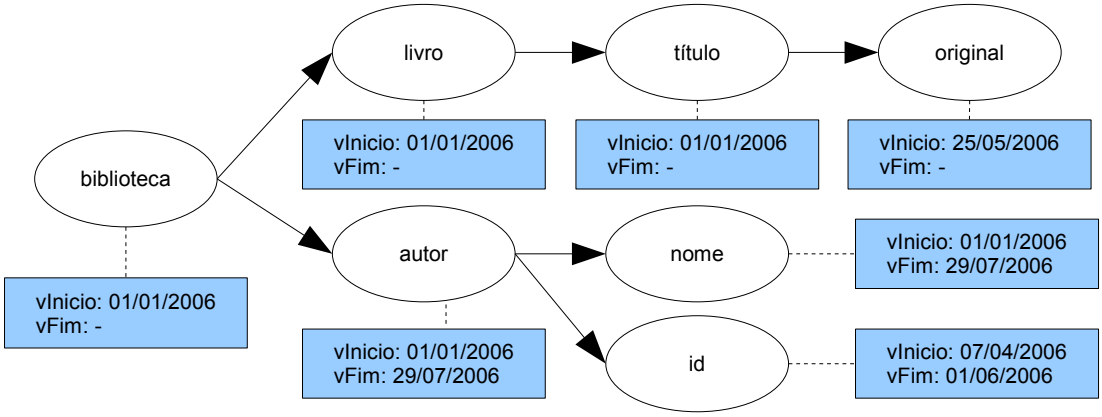

Figura 2.2: Esquema com suporte a tempo de validade

A Figura 2.2 ilustra um esquema XML, independente de representação em formato DTD ou XML Schema, onde cada elemento do esquema encontra-se associado a atributos *vInicio* e *vFim* que identificam o início e o fim do tempo de validade destes elementos na realidade modelada. No exemplo, os elementos obedecem a uma restrição de temporalidade simples, onde os intervalos de tempos de validade dos elementos devem estar contidos nos intervalos de tempos de validade de elementos ancestrais.

Uma segunda alternativa para a abordagem da evolução de projeto de um esquema XML consiste no armazenamento de versões do esquema. Conforme definido em (JENSEN; et al., 1998), o conceito de versões permite ao usuário do banco de dados a consulta de dados através de versões por ele definidas ou definidas automaticamente a cada modificação no esquema. A introdução do conceito de versões na definição de esquemas XML permitiria que múltiplas encarnações de uma mesma estrutura de dados coexistissem, permitindo por exemplo, o trabalho cooperativo de diferentes equipes na definição das estruturas de dados.

A Figura 2.3 ilustra um esquema XML versionado. Neste exemplo, toda a estrutura do esquema é versionada integralmente, sendo cada versão associada a atributos *idVersao* e *versaoCorrente*, que especificam o identificador da versão e se uma determinada versão do esquema é considerada a versão corrente.

Entretanto, tanto a abordagem temporal quanto versionada para evolução de esquemas possuem aplicabilidade restrita por utilizarem formatos de esquema diferentes do formato padrão já utilizado por um grande número de aplicações e projetos de pesquisa. Estas diferenças devem-se à introdução dos conceitos de tempo e versão nos elementos de um esquema XML, ausentes na definição inicial deste tipo de artefato.

Ambas as abordagens podem ter sua aplicabilidade aumentada através da utilização de filtros que interceptem requisições a estes esquemas. Estes filtros processariam o esquema a fim de identificar a versão correta, considerando parâmetros de versão corrente ou tempo de validade, e forneceriam como resposta o esquema processado, que encontraria-se no formato padrão de esquemas XML. O fluxo de tarefas e os parâmetros requisitados pelo

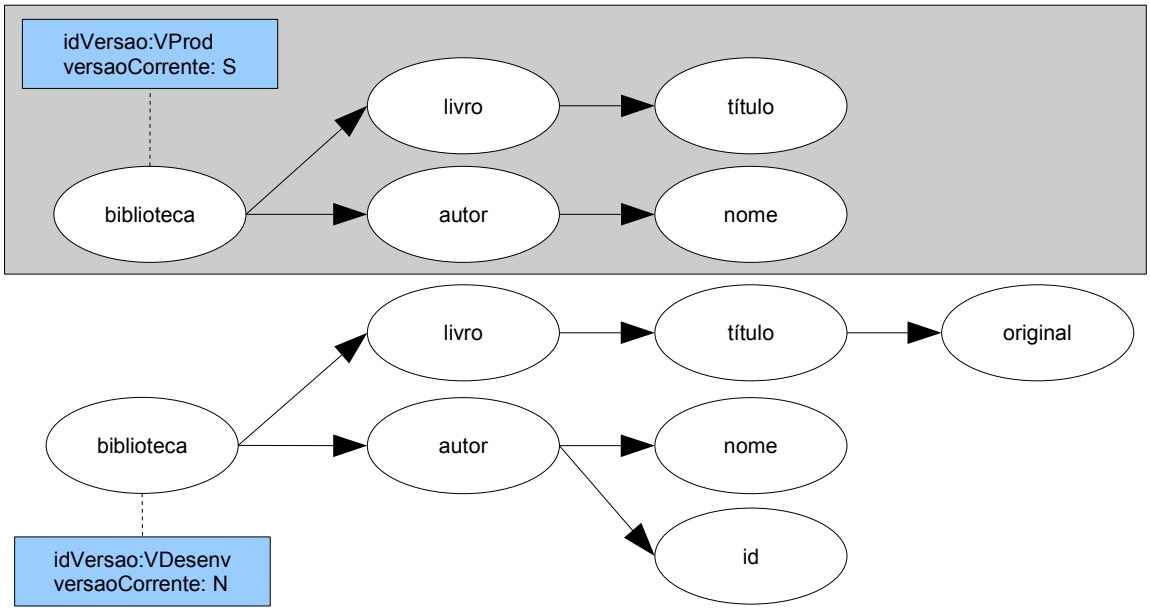

Figura 2.3: Esquema versionado

processo de filtragem de esquemas encontra-se ilustrado na Figura 2.4.

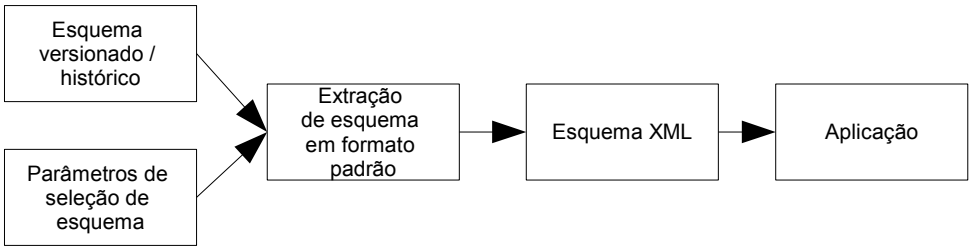

Figura 2.4: Filtro de esquemas versionados ou com histórico

Embora a inclusão de conceitos de tempo e de versão permita um processo de evolução com grande visibilidade sobre as modificações realizadas em esquemas, o fato de utilizar um formato apenas baseado em padrões para representação destas estruturas de dados é um impeditivo à adoção destas abordagens. Mesmo a utilização de filtros que atuem sobre estes esquemas não viabilizaria sua adoção, na medida em que estes filtros necessariamente teriam de ser utilizados junto a estes esquemas, o que pode não ser possível em todos os cenários de utilização de esquemas semiestruturados. Uma discussão mais aprofundada sobre este tópico pode ser encontrada em (SILVEIRA, 2006).

Independentemente da abordagem adotada para implementação do processo de evolução de esquemas do ponto de vista do projeto, no que diz respeito à evolução da base de dados semiestruturados no nível lógico, o conjunto de aspectos das instâncias passíveis de controle através de esquemas XML é consideravelmente grande, logo, também grande é o conjunto de modificações que podem ser realizadas em esquemas XML:

- inclusão de novo elemento;
- alteração de nome de elemento;
- alteração de tipo de dado de elemento;

- exclusão de elemento;
- movimentação de elemento na hierarquia de dados;
- cópia de um elemento para outro ponto na hierarquia de dados;
- definição da classe de valores aceitos por um elemento;
- alteração da cardinalidade de elemento;
- alteração do agrupamento de elementos;
- inclusão de novo atributo;
- exclusão de atributo;
- alteração de nome de atributo;
- alteração de tipo de dado de atributo.

Considerando-se as operações de modificação listadas, tem-se que todas elas, quando aplicadas a um esquema válido, geram um esquema válido, desde que as referências a componentes existentes no esquema sejam respeitadas. A utilização de referências incorretas pelas operações de modificação pode levar o esquema a um estado de inconsistência que será detectado quando um documento for validado contra o esquema ou quando ferramentas específicas de validação de esquemas XML analisarem o esquema em questão. Sabendo-se ainda que qualquer esquema válido pode ser construído a partir da execução de seqüências de operações de modificações, tem-se que tais operações garantem a integridade interna de um esquema. Quando apenas a integridade interna do esquema é analisada, a relação de integridade entre esquema e instâncias é desconsiderada.

Ainda analisando-se as operações de modificação de esquemas XML e sabendo-se que a classe de esquemas representáveis pelo formato DTD é contida pela classe de documentos representáveis pelo formato XML Schema (RAGHAVACHARI; SHMUELI, 2004), pode-se assumir o formato XML Schema como um formato mais genérico, com maior poder de expressividade. Tem-se então que todas as operações de modificação de esquemas em formato DTD necessariamente aplicam-se ao formato XML Schema.

De uma maneira geral, dada a ausência de um sistema gerenciador de bases semiestrutradas, pode-se dizer que o conjunto de modificações possíveis nestes esquemas pode ser restrito apenas por um modelo de suporte à evolução destes esquemas. A definição e classificação destes conjuntos de operações permitem que a consistência estrutural do esquema seja mantida após a execução destas modificações.

A manutenção da consistência estrutural de um esquema semiestruturado pode ser atingida através de diferentes maneiras. Uma possibilidade refere-se ao gerenciamento do esquema por um sistema específico, tal qual um sistema gerenciador de modificações, responsável pelo processamento e validação de cláusulas de alteração do esquema. Assim sendo, quando uma modificação levasse o esquema a um estado inconsistente, esta seria rejeitada, garantindo a consistência estrutural do esquema.

Uma alternativa a esta possibilidade refere-se à execução de processos de validação do esquema, que identifiquem inconsistências em esquemas XML e falhas específicas referentes a um dado formato de representação de esquema. Estes processos de validação podem ser executados manualmente pelo usuário do esquema ou por aplicações que

monitorem constantemente o estado de um esquema. Como resultado da identificação de uma inconsistência no esquema, tanto o esquema poderia ser restaurado a uma versão consistente quando os usuários do esquema sinalizados sobre o ocorrido.

Já a manutenção da consistência comportamental de um esquema semiestruturado é obtida através de modificações nas aplicações que referenciam os esquemas que passaram por modificações. Segundo o processo usual de desenvolvimento de aplicações, a pessoa ou a equipe responsável por uma determinada aplicação deve ter ciência da alteração realizada em um determinado esquema, para que a aplicação sob sua responsabilidade seja alterada a fim de manter seu correto funcionamento.

Também é abordada pela evolução de esquemas no nível lógico a adaptação das instâncias já existentes, definidas de acordo com estruturas que passaram por modificações. Esta adaptação é realizada a fim de garantir a integridade da base de dados, através da eliminação de inconsistências entre o esquema e os dados. A adaptação dos dados pode ser realizada por mecanismos que identifiquem as alterações realizadas em um esquema válido e as reflitam no conjunto de instâncias que referem-se a este esquema. Novamente, este mecanismo poderia ser executado após requisição do usuário da base de dados semiestruturados ou após identificação automática de alterações em um dado esquema.

Seja executado manual ou automaticamente, o processo de adaptação de dados XML consiste na execução de seqüências de cláusulas de atualização, que modificam os valores de elementos e atributos, além de nomes de componentes da estrutura de um documento ou mesmo alteram a organização estrutural de um documento XML.

Considerando-se a potencial distribuição de bases de dados semiestruturados, idealmente os processos de evolução e adaptação dos dados devem ser realizados com a base de dados operacional, uma vez que o bloqueio do acesso às instâncias do banco de dados em todos os nodos que compõem a base distribuída pode ser inviável. A inviabilidade de bloqueio de acesso às instâncias deve-se ao desconhecimento sobre o conjunto total de nodos que armazenam as instâncias que compõem o banco de dados semiestruturado.

Realizando-se a evolução do esquema com a base operacional, os usuários podem acessar instâncias inválidas em relação ao novo esquema. Alternativas a este cenário, que persistirá até que atualizações ocorram nos documentos ou no próprio esquema, podem ser administradas temporariamente por aplicações responsáveis por estas instâncias inconsistentes. A administração pode variar no seu nível de intervenção, consistindo no bloqueio de acesso a estas instâncias ou mesmo na execução do processo de adaptação da instâncias a partir de identificação de alterações no esquema ou sinalização de modificação no esquema recebida a partir de outro nodo na rede de dados.

Através da utilização de aplicações que gerenciem instâncias inconsistentes, idealmente estas devem ter conhecimento sobre todas as instâncias que compõem a base de dados semiestruturados, a fim de garantir tanto o bloqueio do acesso quanto a adaptação de todos os dados. Com isto, pode-se adotar uma abordagem homogênea a todas as instâncias da base de dados, independentemente de seu nível de distribuição.

No que refere-se à evolução do esquema externo, composto por visões construídas a partir de um esquema, as mesmas abordagens apresentadas são aplicáveis: aplicações específicas podem reconstruir visões a partir da análise das modificações realizadas no esquema ou as visões permanecerão em estado inconsistente até o momento em que o seu desenvolvedor altere-a, tornando-a novamente compatível com um determinado esquema.

Por fim, no que concerne a evolução de esquemas XML no nível físico, reestruturações ou reformatações podem ser necessárias em bases de dados semiestruturados que contam com um sistema gerenciador. Estruturas de dados definidas em um esquema podem de-

terminar a contigüidade de armazenamento das instâncias, o que possui conseqüências diretas sobre a agilidade com que estas instâncias são recuperadas por cláusulas de consulta, por exemplo.

Assim sendo, após a evolução do esquema de dados, a contigüidade do armazenamento dos dados ou mesmo o tamanho do bloco físico mínimo de armazenamento podem ser alterados após análise das novas características do esquema, a fim de assegurar um determinado nível de desempenho no processo de recuperação de dados.

2.2.2 Adaptação de documentos XML

O conceito de adaptação de dados semiestruturados em função de modificações nos esquemas praticamente não possui correspondente em bases de dados relacionais uma vez que modificações em esquemas relacionais são validadas pelo SGBD a fim de garantir a integridade da base de dados, o que já não ocorre com dados semiestruturados.

Uma vez que a definição de dados semiestruturados, mais especificamente o W3C XML, não prevê a existência de um gerenciador dos dados representados neste formato, a princípio não há um mecanismo que identifique a validade de modificações realizadas em esquemas XML. Some-se a este fato as características de distribuição e disjunção de esquemas e documentos XML, e tem-se que modificações em esquemas XML podem levar o conjunto formado por esquema e documentos que fazem referência a este esquema a um estado inconsistente.

Assim sendo, após uma modificação realizada em um esquema XML, é necessário propagar esta modificação para todos os documentos XML que fazem referência a este esquema, seja manualmente, através de um *script* de atualização ou através de uma aplicação com este fim específico. A adaptação pode consistir na remoção de elementos retirados no esquema até a inclusão de novos elementos em função de alterações na cardinalidade de elementos do esquema.

O processo de adaptação de documentos XML apresenta algumas dificuldades de implementação principalmente no que concerne adaptações desencadeadas em função de determinadas alterações em esquemas XML, como inclusão de novas estruturas de dados ou alteração de cardinalidade de elementos existentes. A inclusão de novas estruturas de dados no esquema possui como ação de adaptação a inclusão destas estruturas também nas instâncias existentes. Entretanto, a inclusão destas novas estruturas só pode ser realizada corretamente por usuários da base de dados, que conhecem os valores próprios do domínio da aplicação e que devem ser associados a estas novas estruturas. Isto configura-se como um desafio ao processo de adaptação de documentos XML realizado automaticamente por aplicações, uma vez que estas aplicações terão de solicitar intervenção do usuário da base de dados, ou deduzir de alguma maneira os valores a utilizar nas novas estruturas. A dedução dos valores pode ser feita através de regras embutidas nestas aplicações, baseadas em análises de padrões encontrados na base de dados ou pode ser baseada em padrões simples, como associação de determinados valores aos tipos de dados definidos em um formato de esquema XML.

Mesmo que uma aplicação realize a dedução dos valores a utilizar em novas estruturas, estes podem ser incorretos do ponto de vista da aplicação, pois a combinação de diferentes padrões detectados na base de dados pode resultar em estruturas de dados que violem regras próprias ao negócio modelado por uma determinada aplicação.

Uma das possibilidades para abordagem à adaptação de documentos XML, a intervenção do usuário, pode não ser possível em cenários onde mensagens XML são geradas dinamicamente e trocadas através de uma rede de dados entre diferentes aplicações. Uma

razão para a impossibilidade de utilização desta solução reside no fato que a adaptação dos documentos pelo usuário da base de dados precisa ser suportada pelas aplicações que geram os documentos. Sabendo-se que as aplicações desenvolvidas atualmente não oferecem suporte a este tipo de funcionalidade, até por falta de um padrão claro para abordagem à adaptação de documentos XML, o usuário fica impossibilitado de corrigir documentos inválidos, gerados dinamicamente por aplicações.

Já as operações de alteração de cardinalidade de elementos existentes em esquemas XML também possuem dificuldades de implementação. A cardinalidade mínima e máxima de elementos pode ser aumentada ou diminuída. No caso de aumento da cardinalidade mínima, pode ser necessária a adição de novos elementos aos elementos já existentes na base de dados. Neste caso, o mesmo problema abordado anteriormente, o da especificação de valores para novas estruturas, é verificado.

Já no caso da diminuição da cardinalidade máxima, pode verificar-se a necessidade de exclusão de elementos excedentes em relação à nova cardinalidade. Neste caso, diferentes abordagens de adaptação podem ser adotadas, como remoção dos últimos ou primeiros elementos excedentes. Entretanto, esta abordagem pode não ser a mais correta, uma vez que a remoção de determinados elementos pode ser regida por regras específicas ao domínio de uma aplicação.

Analisando-se as dificuldades de definição de um processo de adaptação de documentos XML, especialmente aquele que não conta com a intervenção de um usuário, tem-se que mesmo após a adaptação de um documento XML em relação a um esquema que passou por modificações, este documento pode permanecer inválido. Este estado inválido usualmente não é detectado até a execução de uma nova validação do documento em relação ao esquema e ocorre, por exemplo, quando as ações de adaptação consistem na inclusão de novas estruturas no documento, sendo que tais estruturas contam com informações de cardinalidade obrigatória que necessariamente devem ser especificadas pelo usuário final da aplicação.

Considerando-se as variações de cenários onde o processo de adaptação de documentos XML pode ocorrer, não há um padrão claro para implementação destas ações. Propostas científicas publicadas recentemente abordam a questão de maneiras distintas e mesmo a indústria de banco de dados não atingiu um consenso sobre esta questão, uma vez que os SGBDs mais representativos da atualidade não utilizam abordagens homogêneas, conforme descrito em maiores detalhes no próximo capítulo.

2.3 Revalidação de documentos XML

O processo de revalidação de documentos XML é de especial interesse à área de evolução de esquemas na medida em que permite identificar a validade de um documento XML com relação a um esquema após a evolução deste último. O processo usual de revalidação de documentos consiste em verificar se toda a estrutura do documento e os valores nele contidos são aceitos pelo esquema. Este, na verdade, é o processo de validação executado em um documento quando este é validado contra um esquema XML, sendo novamente executado em algum momento após as modificações no esquema.

Na literatura encontram-se trabalhos que realizam a revalidação de documentos em relação a esquemas utilizando-se de diferentes recursos a fim de agilizar este processamento. Há de se notar que é desejável um processo de revalidação executado no menor tempo possível, uma vez que este processo pode ser aplicável a um número previamente desconhecido de documentos XML.

Um exemplo de abordagem para a revalidação de documentos XML é apresentado em (RAGHAVACHARI; SHMUELI, 2004). Neste trabalho é definido um mecanismo que pré-processa esquemas XML tanto em formato XML Schema quanto DTD, a fim de revalidar documentos inicialmente válidos em relação a um esquema. O trabalho de Raghavachari ainda permite, entre a conformidade com o esquema original e a revalidação contra um esquema destino, a realização de determinadas modificações no documento. O trabalho define duas etapas distintas: na primeira, que corresponde à etapa de pré-processamento, os tipos definidos nos esquemas de origem e destino são avaliados e sua compatibilidade determinada. Determinada a compatibilidade entre os esquemas, o documento em si é validado em relação ao esquema destino, uma vez que informações como cardinalidade de elementos não são avaliadas na fase de pré-processamento por estarem presentes nas próprias instâncias.

Os autores do algoritmo provam que este possui complexidade variável em função do tamanho dos documentos revalidados, e não em função do tamanho dos esquemas pré-processados. Assim sendo, o processo de revalidação de documentos, mesmo que esses tenham sofrido modificações, varia conforme o número de elementos presentes no documento. Embora identifiquem o problema de adaptação de documentos tornados inválidos em função de modificações em esquemas, este problema não é endereçado neste trabalho.

Enquanto no trabalho de Raghavachari a abordagem consiste em verificar se a validade de um documento em relação a um esquema original é mantida em relação a um esquema modificado, em (BOUCHOU; ALVES, 2003) o objetivo é verificar se após modificações em um documento, este permanece válido em relação a um esquema cuja estrutura era respeitada pelo documento até a realização das modificações. Para atingir tal objetivo, os autores definem estruturas de dados para a representação das restrições definidas em um esquema, bem como do próprio documento XML.

De posse desta abstração de esquemas e documentos XML, os autores definem operações de modificação no documento XML, sendo que estas operações, quando utilizadas, mantêm a validade do documento existente. Estas operações garantem a validade do documento, pois elas são efetivamente executadas, alterando estruturas ou dados contidos no documento somente após uma avaliação que determina se a integridade do documento será mantida após sua execução.

Nesta validação executada antes da efetivação das operações de modificação reside a contribuição do trabalho, na medida em que somente a subárvore afetada pela modificação é revalidada, evitando-se a revalidação completa do documento em relação a um esquema. Naturalmente, a revalidação parcial descrita no trabalho possui maior desempenho do que a validação total de um documento, sendo que esta validação pode ser desencadeada em função de uma modificação em uma pequena porção do documento.

Em (BARBOSA et al., 2004) é adotada a mesma abordagem da proposta de Bouchou, onde as modificações são aplicadas a um documento e este é revalidado contra um esquema que não sofreu modificações. A proposta define um conjunto de operações de atualização de documentos XML, sendo uma destas operações fornecida como parâmetro ao algoritmo, junto com o documento a ser revalidado e o esquema, a fim de processar a revalidação em si.

Os autores afirmam realizar a revalidação estrutural e referencial de documentos XML em tempo de O($n \log n$), sendo a validade referencial identificada em função do uso de atributos que definem identificadores únicos de elementos do documento bem como referências a estes identificadores. A fim de atingir as ordens de complexidade citadas, os autores do algoritmo definem restrições no formato do esquema XML. As restrições

modelam o esquema XML de maneira que ele não possua rótulos de elementos repetidos, ou que caso haja repetição de rótulos de elementos, que estes rótulos ainda sejam intercalados com dois rótulos distintos, a fim de garantir uma distância mínima entre as ocorrências de um rótulo.

Além de aplicar restrições sobre a estrutura do esquema XML a fim de garantir um nível de complexidade, a proposta de Barbosa opera com as definições do formato DTD, de maneira que características adicionais que XML Schema possui em relação ao formato DTD não são abordadas pela proposta.

Em (PAPAKONSTANTINOU; VIANU, 2003) novamente o foco consiste em revalidar documentos XML a partir de modificações realizadas em documentos com referências a esquemas XML. Para tanto, os autores estendem algoritmos de divisão e conquista aplicados na resolução do problema de validação incremental de seqüências de caracteres (*strings*) em relação a expressões regulares, de maneira a permitir a revalidação parcial de documentos contra esquemas em formato DTD e XML Schema.

Embora apresente ganhos em relação ao processo de revalidação via força bruta, os autores da proposta apontam que o algoritmo ainda pode ser aprimorado. Estas melhorias visam suportar operações complexas de atualização dos documentos XML, tirando vantagem da realização de diversas operações de atualização no documento, na medida em que o processamento de diversas atualizações simultaneamente pode levar ao descarte de passos intermediários que não influenciariam no resultado final da revalidação. Ainda, a análise de diversas operações de atualização simultaneamente pode levar à conclusão de que a revalidação completa do documento pode ser mais eficiente.

A fim de avaliar as características e restrições de um conjunto de algoritmos de validação de algoritmos XML, diferentes aspectos destes algoritmos precisam ser considerados. Os critérios utilizados na comparação dos algoritmos apresentados são:

- suporte a operações complexas - identifica se o algoritmo possui suporte a operações como movimentação ou cópia de subárvores. Algoritmos que não possuam suporte a tais operações as modelam como seqüências de operações primitivas como inserção e exclusão de elementos;

- suporte a transações - aponta se o algoritmo aproveita-se da análise de seqüências de operações de modificação no processo de revalidação de documentos. Idealmente, o algoritmo deve oferecer tal característica, na medida em que diferentes operações de modificação podem operar em uma mesma subárvore, de maneira que apenas determinadas modificações devem ser consideradas na revalidação;

- iniciador da revalidação - define se o processo de revalidação de documentos XML é iniciado a partir de modificações em esquemas ou nos próprios documentos. Considerando-se o foco deste trabalho na evolução de esquemas XML, tem-se que idealmente o algoritmo deve iniciar o processo de revalidação de documentos a partir de modificações em esquemas XML;

- suporte a diferentes formatos de esquema - identifica se o algoritmo de revalidação opera com diferentes formatos de esquemas XML. Idealmente, o algoritmo deve operar com os formatos DTD e XML Schema, por serem os formatos mais utilizados de esquemas XML na atualidade.

A Tabela 2.1 sumariza os algoritmos apresentados e suas características, conforme os critérios de avaliação apresentados.

Tabela 2.1: Comparativo de algoritmos de revalidação de documentos XML

Item/Algoritmos	Raghavachari	Bouchou	Barbosa	Papakonstantinou
Operações complexas	√	√	×	×
Suporte a transações	√	×	×	×
Iniciador da revalidação	Esquema	Documento	Documento	Documento
Formatos de esquema	DTD, XML Schema	DTD	DTD, suporte parcial a XML Schema	DTD, XML Schema
	Legenda:	× Não possui	−− Não conhecido	

Analisando-se a Tabela 2.1, tem-se que o algoritmo de Raghavachari apresenta características desejáveis quando aplicado ao contexto de evolução de esquemas XML, na medida em que opera com dois formatos de esquemas, além de ser orientado a modificações em esquemas, o que alinha-se às necessidades da evolução de esquemas XML, onde um número inicialmente desconhecido de documentos deve ser revalidado após modificações em esquemas. Revalidações iniciadas a partir de modificações em documentos são aplicáveis a cenários onde um número sabidamente baixo de documentos são validados contra esquemas XML após sofrerem modificações, sendo tal característica apresentada pelas propostas de Bouchou, Barbosa e Papakonstantinou.

Independentemente do iniciador do processo de revalidação, algoritmos que apresentam suporte a operações complexas, como cópia e movimentação de elementos de um artefato XML, como os trabalhos de Raghavachari e Bouchou, permitem que a revalidação seja simplificada, na medida em que a revalidação não consistirá na validação do resultado de operações que constituem uma operação primitiva, e sim na validação do resultado de apenas uma operação complexa.

De maneira similar, a capacidade de executar validações que considerem um conjunto de operações, sejam elas primitivas ou complexas, tal como ocorre com o trabalho de Raghavachari, é desejável na medida em que diversas operações individuais podem operar sobre um mesmo fragmento XML, mas nem todas estas operações demandam a revalidação do fragmento modificado, pois uma operação de modificação pode anular o efeito de todo um conjunto de operações para fins de revalidação de documentos XML.

2.4 Sistemas observadores

Em (DYRESON, 2001) é definido o conceito de sistema observador, baseado em sistemas que realizam a leitura de dados e que normalmente não podem alterá-lo. Exemplos destes sistemas são navegadores *web*, que usualmente apenas lêem e exibem ao seu usuário os dados enviados por um servidor *web*.

No artigo de Dyreson, um sistema observador capta as modificações realizadas em conjuntos de documentos XML a fim de construir um histórico da evolução destes documentos. A observação em si ocorre através de consultas aos documentos XML efetuadas a intervalos periódicos, sendo que a cada consulta uma versão do documento é armazenada no banco de dados próprio do sistema observador. A partir da comparação entre as diferentes versões dos documentos capturadas pelo sistema observador é construído o histórico da evolução dos documentos.

A abordagem de observação de recursos mostra-se útil quando transposta para o ambiente de esquemas XML, pois permite a identificação da evolução de esquemas com o mínimo de intrusão nos cenários de execução das aplicações que fazem uso destes esquemas. Além de registrar a evolução dos esquemas através do armazenamento das diferentes versões de um esquema, um sistema observador permite que as modificações realizadas nos

esquemas não sejam restritas por formatos específicos de modelos de suporte à evolução de esquemas, dando liberdade ao administrador do esquema de realizar as modificações nestes recursos da maneira mais adequada segundo seu próprio ponto de vista.

2.5 Detecção de diferenças em esquemas XML

Algoritmos de detecção de diferenças em documentos XML constituem um meio para identificação de diferenças entre versões de esquemas XML codificados no próprio formato XML. Estes algoritmos recebem como entrada dois documentos ou esquemas XML e geram como saída uma seqüência finita de operações de modificação que identificam como transformar o primeiro no segundo.

Os diferentes algoritmos de detecção de diferenças em documentos XML encontrados na literatura possuem diferentes objetivos e valem-se de diferentes mecanismos para atingi-los. Uma diferença observada quando estes algoritmos são analisados refere-se ao desempenho e ao que espera-se como saída destes algoritmos. Alguns algoritmos optam por não gerar uma seqüência de operações de transformação, o *delta*, dita mínima em troca de ganhos de desempenho. Além disso, as seqüências de operações de transformação geradas por diferentes algoritmos podem variar, pois os algoritmos suportam diferentes transformações. Usualmente, todos os algoritmos suportam as operações básicas de inserção, exclusão e atualização de elementos e atributos, enquanto que algoritmos mais avançados suportam as operações de cópia e movimentação de elementos.

Os algoritmos de detecção de diferenças podem apresentar variações quanto ao modelo de árvore XML utilizado. No modelo não-ordenado apenas o aninhamento dos elementos, ou seja, a relação existente entre um elemento e seu ancestral é relevante para fins de identificação de diferenças. Já no modelo ordenado tanto a relação entre um elemento e seu ancestral quanto o posicionamento de um elemento em relação a seus irmãos podem ser utilizados na detecção de diferenças. Logo, neste modelo é possível identificar como diferenças as trocas de posição entre dois descendentes de um mesmo ancestral.

Um exemplo de algoritmo de detecção de diferenças em documentos XML é o X-Diff, apresentado em (WANG; DEWITT; CAI, 2003). Este algoritmo utiliza um conjunto básico de operações de transformação para construção dos deltas.

O X-Diff apresenta complexidade da ordem $O(n^2)$ e seu funcionamento baseia-se na geração de assinaturas *hash* para cada elemento dos documentos comparados, casamento dos nodos com assinaturas iguais e identificação de seqüências de alteração mínimas entre nodos não casados. O algoritmo adota critérios na identificação das operações de transformação que garantem a minimalidade das seqüências de operações de transformação.

Já em (COBENA; ABITEBOUL; MARIAN, 2002) é definido o XyDiff, um algoritmo para detecção de diferenças em documentos XML voltado para grandes quantidades de dados. Por ser voltado a grandes quantidades de dados, os autores do algoritmo evidenciam como maior objetivo o alto desempenho do algoritmo, e aceitam alguma perda de minimalidade na geração do *script* de diferenças entre dois documentos quaisquer.

O XyDiff possui complexidade de tempo de ordem $O(n * logn)$ e suporta as operações básicas de diferenciação e a operação de movimentação de elementos. O algoritmo se vale de características próprias do formato XML para sua operação, como a utilização de identificadores únicos de elementos na diferenciação entre documentos e o modelo ordenado de árvore XML.

Em linhas gerais, o algoritmo calcula a função *hash* dos elementos dos documentos XML e procura fazer o casamento destes valores nos dois documentos. O algoritmo

procura inicialmente realizar o casamento de árvores com maior peso, ou seja, de maior profundidade e maior número de elementos a fim de processar um maior número de elementos dos documentos. Elementos não casados são identificados como elementos inseridos ou excluídos recentemente. Já elementos casados, porém com ascendências distintas entre os dois documentos, são considerados elementos movidos de uma posição a outra.

O Xandy é definido em (LEONARDI; BHOWMICK; MADRIA, 2005) e adota uma abordagem para comparação de documentos XML diferente daquela adotada por X-Diff e XyDiff. Enquanto estes algoritmos realizam a comparação das instâncias XML na memória principal do processador, o Xandy opta por processar e armazenar documentos XML em um banco de dados relacional e utilizar o poder computacional do banco relacional na comparação entre os documentos XML.

Esta abordagem possui como característica relevante o fato de permitir a comparação de instâncias significativamente maiores do que aquelas suportadas pelos algoritmos X-Diff e XyDiff, uma vez que estes algoritmos realizam a carga em memória dos documentos completos, enquanto que o Xandy, apoiado por um banco de dados relacional, consegue processar uma maior quantidade de dados.

Xandy é composto por duas fases distintas: a carga dos documentos na base de dados onde a estrutura e o conteúdo dos documentos são processados e armazenados nas tabelas próprias do Xandy, e a comparação dos documentos em si, sendo que esta comparação utiliza o modelo não-ordenado de árvore XML. O algoritmo utiliza um conjunto básico de operações de inserção, exclusão e atualização de conteúdo para descrever as modificações realizadas nos documentos.

O XMLTreeDiff, definido em (EPSTEIN; CURBERA, 1998), é outro exemplo de algoritmo de diferenciação que realiza a carga dos documentos XML em memória. O XMLTreeDiff utiliza APIs DOM (*Document Object Model*), o que o impede de processar documentos de grande porte.

Este algoritmo suporta o modelo ordenado de árvore XML, utiliza um conjunto básico de operações para identificar as transformações realizadas em documentos e possui complexidade de ordem pelo menos $O(n^2)$, segundo (BEX; NEVEN; BUSSCHE, 2004).

A fim de permitir a comparação de características e limitações de cada método de detecção de diferenças de documentos XML, um conjunto de características deve ser avaliada em todos os algoritmos. Este conjunto de características avaliadas corresponde aos critérios de avaliação dos métodos de detecção de diferenças em documentos XML. Os critérios aplicados na comparação entre os algoritmos apresentados são:

- complexidade - identifica a complexidade de execução em tempo de um determinado algoritmo. Mesmo que usualmente esquemas XML sejam documentos com poucos elementos quando comparados a documentos XML em si, é desejável que o algoritmo de detecção de diferenças possua baixa complexidade, o que leva a um melhor desempenho do algoritmo;

- operações implementadas - o suporte a diferentes operações, além das operações consideradas básicas, leva à geração de *deltas* mais compactos e de mais fácil compreensão e implementação. Assim sendo, é desejável que os algoritmos ofereçam suporte a diferentes operações, como a movimentação e cópia de elementos;

- modelo de árvore XML - dois modelos de ordenação podem ser aplicados a documentos XML: o modelo ordenado, onde a ordenação entre elementos filhos de um

mesmo elemento é relevante, e o modelo não-ordenado, onde somente a relação entre um elemento e seu ancestral é relevante. Na detecção de diferenças em esquemas XML, é desejável que o algoritmo opere com o modelo ordenado de documentos;

- qualidade das saídas - identifica se a saída gerada por um algoritmo é correta, ou seja, se o delta gerado transforma corretamente um documento original em um segundo documento;

- tamanho das entradas / saídas - aponta se o algoritmo possui restrições ou características especiais no que refere-se ao tamanho das entradas e/ou saídas.

A Tabela 2.2 sumariza os algoritmos de detecção de diferenças apresentados e suas características, conforme os critérios de avaliação apresentados. Os algoritmos analisados compreendem um subconjunto dentre os inúmeros existentes, entretanto, são algoritmos relevantes na área na medida em que apresentam as características mais usuais na abordagem à detecção de diferenças em XML.

Tabela 2.2: Comparativo de algoritmos de detecção de diferenças em documentos XML

Item/Algoritmos	XyDiff	X-Diff	Xandy	XMLTreeDiff
Complexidade	O(*n log n*)	O(n^2)	$--$	O(n^2)
Operações implementadas	básicas e movimentação	básicas	básicas	básicas
Modelo da árvore XML	ordenado	não-ordenado	não-ordenado	ordenado
Qualidade das saídas	delta não-mínimo	delta não-ordenado	delta não-ordenado	delta correto
Tamanho das entradas / saídas	×	×	processamento de entradas de tamanho armazenáveis em SGBDs	entrada < 50Kb

Legenda: × Não possui $--$ Não conhecido

A primeira conclusão relevante que pode ser retirada da análise da Tabela 2.2 deriva do fato que algoritmos que operam com o modelo não-ordenado de árvores XML geram deltas inadequados quando da comparação de dois esquemas XML. Esta inadequação deve-se à impossibilidade de identificação de trocas de posição entre elementos irmãos, sendo que estas posições são fundamentais na estruturação de documentos XML que referenciam estes esquemas. Quanto ao algoritmo XMLTreeDiff, decorre de uma restrição no tamanho das entradas do algoritmo a sua inadequação para uso em detecção de diferenças em esquemas XML, uma vez que estes podem superar o tamanho aceito pelo algoritmo.

Dentre os algoritmos apresentados, apenas Xandy requer uma estrutura de execução mais complexa, com a utilização de um banco de dados para armazenamento e processamento dos documentos. Em ambientes de execução, onde os recursos computacionais são escassos, o algoritmo pode ter sua execução inviabilizada.

O algoritmo XyDiff possui complexidade menor em relação aos demais algoritmos uma vez que opta pela geração de um delta não mínimo. Um delta não-mínimo não implica em incorreção, implica na verdade na utilização de um número de operações de modificações maior do que o necessário para descrição de uma alteração em um documento ou esquema XML. A não-minimalidade dos deltas gerados por este algoritmo possui dois diferentes impactos: no espaço de armazenamento, caso estes deltas sejam armazenados para referência futura e processamentos desencadeados com base nestes deltas, uma vez que estes processamentos executarão mais operações do que o necessário em função da presença de operações não necessárias no delta.

A fim de validar as informações teóricas sobre os diferentes algoritmos, suas respectivas implementações, obtidas junto aos autores de cada algoritmo, foram avaliadas a fim de identificar características de desempenho e de qualidade de resultados gerados pelos algoritmos. Surpreendentemente, o algoritmo Xandy não foi capaz de identificar diferenças em documentos de maior tamanho (acima de 300 *megabytes*), ao contrário do que pode depreender-se da análise de (LEONARDI; BHOWMICK; MADRIA, 2005). O uso de uma base de dados relacional para armazenamento de documentos e posterior processamento não permitiu que documentos maiores fossem processados, e ainda introduziu um atraso no processamento de documentos de menor tamanho, o que fez com que o algoritmo tivesse um desempenho menor quando comparado ao X-Diff e ao XyDiff.

Os algoritmos X-Diff e XyDiff possuíram desempenho semelhante na maior parte dos casos. Tanto a qualidade das saídas de ambos os algoritmos é semelhante: as modificações são identificadas igualmente por ambos os algoritmos. O diferencial entre os dois algoritmos deve-se à utilização de operações de movimentação de elementos por parte do XyDiff, o que permite a geração de deltas mais compactos.

Já o algoritmo XMLTreeDiff não pôde ter seu desempenho avaliado uma vez que não foi possível executá-lo. A fim de avaliar o XMLTreeDiff, é necessária a construção de um ambiente de testes diferente daquele utilizado para avaliação dos demais algoritmos. Este ambiente distinto é baseado em uma versão antiga do Java Development Kit, plataforma utilizada para desenvolvimento do XMLTreeDiff. Um ambiente de teste distinto invalida o propósito dos testes, uma vez que os algoritmos seriam executados em ambientes diferentes, logo, o XMLTreeDiff não foi avaliado.

2.6 Considerações finais

Este capítulo apresentou definições relativas ao processo de evolução de esquemas, bem como descrições de técnicas relacionadas à evolução de esquemas XML e à adaptação de documentos que referenciam tais esquemas.

A evolução de esquemas XML difere da evolução de esquemas relacionais na medida em que as instâncias podem encontrar-se em diferentes nodos de rede, além de não serem administradas por um sistema gerenciador de transações e integridade, como usualmente ocorre em bancos relacionais. Tais características introduzem complexidades adicionais na evolução de esquemas XML, na medida em que modificações no esquema podem tornar o conjunto formado por esquema e instâncias inconsistente, uma vez que quaisquer modificações no esquema são permitidas. Some-se a isso o fato de que é necessária uma ação do administrador ou usuário da base de dados a fim de compatibilizar os documentos inválidos com a nova versão do esquema.

Técnicas de revalidação de documentos são apresentadas, sendo estas úteis no processo de verificação de compatibilidade entre documentos e esquemas, na medida em que são executadas com maior desempenho quando comparadas com técnicas de validação usuais, que consistem na validação total de um documento contra um esquema. As técnicas de revalidação apresentadas fazem uso de informações obtidas a partir da análise do esquema e do documento a fim de revalidar apenas as partes dos documentos afetadas pelas modificações nos esquemas ou nos documentos.

Já a definição de sistemas observadores permite a elaboração de mecanismos de suporte à evolução de esquemas XML minimamente intrusivos. Tais sistemas são responsáveis pela captação de dados sobre os esquemas em questão, sendo tais dados utilizados, por exemplo, na modelagem do processo de adaptação dos documentos que referenciam

os esquemas modificados.

Por fim, técnicas de detecção de diferenças em esquemas XML são apresentadas. Tais técnicas permitem a identificação das modificações realizadas em um esquema codificado em formato XML, sendo estas informações vitais para a determinação das ações de adaptação que devem ser executadas no conjunto de instâncias que referenciam um esquema. Tais ações de adaptação são executadas a fim de reestabelecer a integridade da base de dados, e as diferenças entre esquemas podem ser obtidas por diferentes algoritmos, com diferentes características, como por exemplo o seu próprio desempenho.

O próximo capítulo descreve em maiores detalhes e compara trabalhos que definem mecanismos de evolução de esquemas XML e adaptação de documentos regidos por estes esquemas.

3 REVISÃO BIBLIOGRÁFICA

Este capítulo apresenta uma revisão bibliográfica sobre trabalhos relacionados à evolução de esquemas XML e à propagação destas modificações a documentos XML que referenciam estes esquemas. As comparações realizadas entre os trabalhos utilizam critérios que representam características desejáveis em propostas que abordem a evolução de esquemas e a propagação às instâncias do banco de dados. Propostas comerciais de bancos de dados e projetos de pesquisa com definições de suporte à gerência da evolução de esquemas XML são analisados separadamente, entretanto, os mesmos critérios de avaliação são aplicados às duas classes de propostas.

3.1 Propostas de projetos de pesquisa

Nesta seção são apresentadas as características de projetos de pesquisa que abordam a evolução de esquemas XML e a propagação destas modificações para os documentos.

3.1.1 Proposta de Coox

Em (COOX, 2003) é definido um conjunto de invariantes de um esquema XML, que devem ser mantidas após o processo de evolução de um esquema a fim de garantir a integridade de bases de dados semiestruturados. A existência de um mecanismo que verifique a manutenção destas invariantes lembra tarefas executadas por um SGBD, uma vez que conforme proposto por Coox, caso estas invariantes não sejam mantidas, determinadas modificações em um esquema devem ser rejeitadas, uma vez que a base de dados passará a um estado inconsistente.

A proposta de Coox aborda as definições de esquemas em formato DTD e não aborda cenários como definições recursivas de elementos e recursão indireta de elementos em DTD. A proposta não aborda a propagação de modificações em esquemas para documentos existentes e aponta como trabalho futuro a identificação de diferenças entre dois esquemas XML, com o intuito de acelerar o processo de indexação de documentos XML.

3.1.2 Proposta de Bertino

Em (BERTINO et al., 2002) é apresentado um mecanismo para evolução de DTDs a partir de padrões encontrados em documentos XML. Através da aplicação de técnicas de mineração de dados, são detectados padrões sobre as estruturas de dados existentes tanto nos documentos XML quanto nos DTDs. Estes padrões são utilizados para associar um determinado conjunto de documentos a um esquema DTD, havendo a possibilidade de que documentos sem similaridades com quaisquer DTDs sejam identificados e armazenados para futura reavaliação.

A partir da análise de um certo número de documentos, determinado pelo usuário do mecanismo, é iniciada a fase de evolução de DTDs. Nesta fase, a partir dos padrões detectados nos XMLs associados a cada DTD, tem início a fase de evolução dos DTDs, com a inclusão, exclusão e reposicionamento de elementos. Após a fase de evolução dos DTDs, os documentos XML anteriormente não associados a nenhum esquema são reavaliados, a fim de determinar as suas novas associações.

Embora aborde a evolução de esquemas XML, o trabalho de Bertino não aborda esquemas definidos em XML Schema, e não aborda a propagação da evolução dos DTDs para os documentos XML, que em última instância, desencadearam o processo de evolução dos esquemas.

3.1.3 Proposta de Su

Em (SU et al., 2001) é definido o XEM (XML *Evolution Manager*), um sistema gerenciador do processo de evolução tanto de documentos quanto de esquemas XML codificados em formato DTD. O trabalho define um conjunto semanticamente completo e válido de operações de evolução dos esquemas, que devem atender a certas pré-condições a fim de serem executadas em um esquema.

As modificações nos esquemas que atendem às pré-condições definidas no trabalho são propagadas aos documentos XML. Entretanto, nem todas as possíveis modificações em um esquema XML são suportadas pelo mecanismo, uma vez que determinadas operações são bloqueadas, não sendo executadas pelo sistema gerenciador. Assim sendo, sob certos aspectos, a abordagem do XEM é semelhante a um banco de dados relacional, onde operações de modificação que tornem a base de dados inconsistente são bloqueadas.

O protótipo do XEM opera em um banco de dados orientado a objetos, onde os DTDs a serem manipulados pelo mecanismo são registrados. Somente após o registro dos DTDs é que os documentos XML são carregados na base de dados, sendo os documentos XML mapeados para objetos do banco de dados. Como os documentos que serão afetados pelo processo de evolução dos esquemas devem estar armazenados neste banco de dados, documentos que encontrem-se armazenados de outra maneira, como em um sistema de arquivos, ou que venham a ser gerados automaticamente por uma aplicação podem ser incompatíveis com a versão mais atual do esquema XML.

3.1.4 Proposta de Al-Jadir

Em (AL-JADIR; EL-MOUKADDEM, 2004) é apresentada uma técnica para suporte à evolução de esquemas XML definidos em formato DTD, além da propagação destas modificações para os documentos que fazem uso dos DTDs que passaram por processos de evolução.

A metodologia consiste em converter os DTDs em um esquema orientado a objetos no banco de dados F2/XML, convertendo em seguida os documentos que fazem referência ao DTD em uma instância de dados do esquema orientado a objetos. As modificações em DTDs são então mapeadas para cláusulas de modificação do esquema orientado a objetos. Estas cláusulas encontram-se associadas a *triggers* que visam em um primeiro momento identificar a validade da operação de modificação do esquema, e no caso de modificações válidas, adaptar os documentos ao novo formato do esquema.

Estas *triggers* implementam as invariantes definidas no trabalho, onde a consistência do banco de dados é garantida através da manutenção de determinadas propriedades do esquema e dos dados. Realizada a execução das cláusulas de modificação dos esquemas e a finalização da adaptação dos dados, tanto o DTD que gerou o esquema do banco de

dados quanto os dados em si devem ser extraídos novamente da base de dados, a fim de refletir as recentes modificações.

A verificação da natureza das modificações remete a uma abordagem tradicional, implementada em bancos de dados relacionais, onde modificações que levem a base a um estado inconsistente são bloqueadas. Como trabalho futuro, os autores apontam o suporte à evolução do formato XML Schema e testes de desempenho do mecanismo.

A ausência de suporte ao formato XML Schema é um ponto importante a ser considerado no trabalho, uma vez que este formato é consideravelmente mais complexo do que o formato DTD. As operações de conversão de dados entre bancos de dados orientados a objetos e XML naturalmente introduzem um atraso na evolução dos dados. A extensão do impacto desse atraso será definido através de análises de desempenho do mecanismo. Por fim, como o mecanismo definido apóia-se na utilização de um banco de dados, naturalmente há centralização dos dados que podem ser adaptados em função da evolução dos esquemas. A evolução de documentos encontrados em diferentes nodos de uma rede de dados seria feita através da carga desses documentos no banco de dados, o que deve ser feito por um mecanismo não referenciado no trabalho.

3.1.5 Proposta de Guerrini

Na proposta de Guerrini (GUERRINI; MESITI; ROSSI, 2005) é apresentado um mecanismo que realiza a propagação de modificações em esquemas para documentos XML. Esta propagação é realizada através da análise de quais foram as modificações realizadas no esquema, seguida de uma análise sobre o impacto destas modificações nos documentos. Caso o mecanismo determine que os documentos devem ser adaptados a fim de tornarem-se novamente válidos em relação à versão atual do esquema, há o início da fase de adaptação, que pode exigir intervenção do usuário. A intervenção do usuário pode ocorrer através da solicitação para especificação de valores de novos elementos.

Esta proposta busca melhorar o desempenho do processo de revalidação de documentos XML através da identificação das porções dos documentos que podem ter sido afetadas pela evolução do esquema. Desta maneira, apenas partes do documento XML são revalidadas quando da evolução do esquema, o que naturalmente apresenta um ganho de desempenho em relação à alternativa a este processo (RAGHAVACHARI; SHMUELI, 2004), que consiste na revalidação completa do documento XML.

O X-Evolution (MESITI et al., 2006), implementação da proposta de Guerrini, oferece uma interface gráfica para a evolução dos esquemas XML, e recebe como parâmetro um conjunto de documentos XML que fazem referência a este esquema. Assim sendo, ao registrar todas as modificações realizadas no esquema pelo usuário, o X-Evolution pode aplicar estas modificações ao conjunto de documentos fornecido como parâmetro.

Embora realize a propagação de modificações em esquemas para documentos, a proposta de Guerrini apresenta como deficiências o fato de exigir intervenção do administrador da base de dados na adaptação dos documentos, além de exigir conhecimento do conjunto total de documentos a ser adaptado quando da evolução dos esquemas.

3.1.6 Proposta de Bouchou

Em (BOUCHOU et al., 2006) não é definido um mecanismo de evolução de esquemas XML, mas sim de evolução de documentos XML com referências a esquemas. Entretanto, a proposta de Bouchou é comparada contra propostas de evolução de esquemas XML, pois o seu mecanismo define ações de revalidação parcial de documentos bem como a sua adaptação no caso de falha de conformidade de um documento a um esquema.

A proposta de Bouchou recebe como entradas um documento XML inicialmente válido em relação a um esquema e tornado inválido em função de atualizações realizadas em sua estrutura e um limiar de correção do documento. Este limiar corresponde a um número máximo de operações de atualização que podem ser realizadas no documento a fim de torná-lo novamente válido em relação ao esquema.

A partir destas entradas o mecanismo identifica todas as possíveis versões de documentos que corrigem o documento e tornam-o novamente compatível com o esquema, observando-se a restrição imposta pelo limiar de operações de modificação utilizadas para obtenção destas versões. Tais versões são então apresentadas ao usuário, para que este determine qual delas melhor corrige um documento XML inválido.

Além disso, o processo de correção de documentos é associado ao processo de revalidação parcial de documentos, onde somente as partes modificadas em um documento são validadas em relação a um esquema XML. De acordo com os autores da proposta, estes processos serão integrados a um *framework* para manipulação de documentos XML, sendo este *framework* responsável pelo processamento de modificações nos documentos, revalidações e adaptações.

3.2 Propostas comerciais

Nesta seção são apresentadas as características que SGBDs comerciais empregam na gerência do processo de evolução de esquemas semiestruturados e na propagação destas modificações aos documentos.

3.2.1 Implementação DB2

Em (NICOLA; LINDEN, 2005) é descrito o suporte à evolução de esquemas XML implementado pelo IBM DB2 *Universal Database*. Este SGBD segue dois princípios básicos no suporte à evolução de esquemas: não exigir que documentos ou esquemas sejam modificados durante o processo de armazenamento de esquemas no DB2 e permitir que um esquema XML evolua ao longo do tempo.

O primeiro princípio é implementado através de validações aplicadas a quaisquer atualizações realizadas em esquemas XML, com objetivo de identificar modificações que levem o conjunto de documentos que fazem uso deste esquema a um estado inconsistente. Esta abordagem de evolução de esquemas pode ser comparada àquela adotada em SGBDs tradicionais, onde modificações em esquemas que levem a base a um estado inconsistente são bloqueadas pelo SGBD.

O segundo princípio é atingido através da disponibilização de um conjunto de APIs (*Application Programming Interface*) que permite o versionamento de esquemas XML, bem como a identificação de qual versão do esquema foi utilizada para validação de um determinado registro. Desta maneira, o banco de dados permite que documentos XML armazenados em uma mesma coluna de uma tabela sejam associados a diferentes versões de um mesmo esquema.

3.2.2 Implementação Oracle

Em (ORACLE, 2004) é descrito o suporte oferecido pelo Oracle 10g à evolução de esquemas XML. Assim como o DB2, o Oracle opera com o registro de esquemas XML. Ou seja, antes de permitir o uso de um esquema XML em uma tabela qualquer, o Oracle exige que o esquema seja registrado na base de dados.

No processo de evolução de esquemas XML o Oracle toma um rumo diferente daquele

adotado pelo IBM DB2, na medida em que permite que sejam realizadas modificações em esquema que inicialmente levem a base de dados a um estado inconsistente. Entretanto, a fim de garantir a consistência da base de dados, o Oracle exige que o usuário responsável pela modificação do esquema também forneça um conjunto de transformações XSL (*Extensible Stylesheet Language*) a fim de adaptar os registros existentes na base de dados ao novo esquema XML.

Uma característica importante do processo de evolução de esquemas no Oracle 10g refere-se ao fato de que o SGBD exige que o usuário forneça novas versões de esquemas dependentes do esquema que está sendo modificado. O SGBD também exige que sejam fornecidas cláusulas XSL para adaptação de documentos com referências diretas ou indiretas ao esquema modificado.

3.2.3 Implementação SQL Server

Em (PAL; FUSSELL; DOLOBOWSKY, 2004) é descrito o mecanismo de evolução de esquemas XML do SQL Server 2005. Assim como o DB2 e o Oracle, o SQL Server exige que um esquema XML seja registrado antes de ser referenciado na validação de colunas do tipo XML. O restante do processo de evolução de esquemas XML é bastante semelhante àquele observado no Oracle: o usuário é responsável por modificar registros incompatíveis com a nova versão do esquema através da execução de *scripts* XSLT ou através da exclusão de registros inconsistentes.

Entretanto, como a modificação do esquema e a adaptação dos documentos existentes constituem passos distintos no SQL Server, caso a modificação no esquema gere um esquema mais restrito, esta modificação pode ser bloqueada pelo SGBD. Neste caso, o processo de evolução do esquema ainda pode ser atingida através da desassociação do esquema às colunas de tipo XML que o referenciam, alteração do esquema, alteração dos documentos XML existentes e reassociação das colunas de tipo XML ao esquema.

3.2.4 Implementação Tamino

O Tamino é um SGBD nativo XML, ou seja, trata-se de um gerenciador de bases de dados XML, responsável pelo controle de transações e processamento de consultas a dados armazenados em formato XML, entre outras funções próprias a SGBDs. No que refere-se a esquemas XML, o Tamino utiliza estruturas em formato próprio, chamadas Tamino Schema Definition (TSD), que correspondem a um subconjunto das definições do formato XML Schema. Os esquemas utilizados pelo Tamino podem ser criados a partir de esquemas definidos em formato DTD e XML Schema, entre outros.

Conforme definido em (SOFTWAREAG, 2006), a partir da criação de um esquema gerenciado pelo Tamino, e da criação de instâncias neste esquema, a evolução do esquema é regida por regras que impedem que este torne-se mais restritivo. Logo, somente elementos de cardinalidade opcional podem ser incluídos em tipos existentes, a cardinalidade mínima de elementos só pode ser diminuída, enquanto a cardinalidade máxima só pode ser aumentada. O SGBD aplica as modificações permitidas somente após a validação das instâncias existentes, a fim de garantir a integridade da base de dados.

3.3 Comparação entre propostas

As diferentes propostas e implementações analisadas neste capítulo compõem o estado da arte sobre a gerência da evolução de esquemas XML e a propagação de modificações de esquemas para documentos. Os critérios utilizados para avaliação dos trabalhos

analisados são:

- propagação de modificações: analisa se a proposta realiza a propagação das mudanças em esquemas para os documentos existentes que possuam referência ao esquema modificado;

- independência de ambiente para alteração de esquema: identifica propostas cujo processo de evolução de esquemas, mais especificamente a modificação da estrutura do esquema, pode ser realizada por quaisquer ferramentas de edição de esquemas, não sendo essa edição restrita a ferramentas definidas nas próprias propostas;

- adaptação de documentos em sistemas de arquivos: analisa se a proposta é capaz de propagar modificações em esquemas diretamente para as instâncias existentes em sistemas de arquivo, sem que estas instâncias tenham de ser carregadas em uma base de dados intermediária a fim de passarem por adaptações;

- adaptação sem intervenção do administrador: identifica se o usuário responsável pela modificação da estrutura do esquema também é responsável diretamente pelo processo de adaptação de documentos tornados inconsistentes em função de modificações realizadas no esquema;

- execução das operações de evolução on-line: aponta se a proposta permite a evolução do esquema enquanto requisições de consulta e alterações são emitidas ao banco de dados que encontra-se em processo de evolução. No caso específico de bases de dados XML, consiste em identificar se o esquema e as próprias instâncias permanecem acessíveis a usuários e aplicações;

- restrições de integridade dos esquemas: identifica se a proposta garante a consistência dos esquemas após a finalização das modificações. Ou seja, identifica se um esquema válido, quando modificado, dá origem a um novo esquema válido, considerando-se apenas a integridade do esquema em si, desconsiderando-se a relação de integridade das instâncias que referenciam este esquema;

- restrições de integridade dos documentos: analisa se a proposta mantém a integridade dos documentos que referenciam um esquema submetido a modificações, quando finalizada a evolução do esquema e a adaptação dos documentos;

- modelo de dados: aponta se a proposta utiliza um modelo de dados para registro do processo de evolução de um determinado esquema;

- taxonomia das operações de mudança: identifica se a proposta possui uma taxonomia específica para classificação das operações de modificação de esquema.

As Tabelas 3.1 e 3.2 sumarizam os trabalhos acadêmicos na área e as implementações comerciais, respectivamente, e suas características considerando-se os critérios de avaliação apresentados.

O primeiro fato relevante que pode ser concluído a partir da análise das Tabelas 3.1 e 3.2 consiste em que nem todas as propostas suportam a evolução e propagação das modificações em esquemas XML para os documentos que referenciam estes esquemas, como é o caso das propostas de Coox e Bertino. Isto não necessariamente constitui um problema

Tabela 3.1: Comparativo de trabalhos relacionados: modificação de esquemas e propagação de mudanças

Itens / Propostas	**Coox**	**Bertino**	**Su**	**Al-Jadir**	**Guerrini**	**Bouchou**
Propagação de modificações	×	×	√	√	√	×
Independência de ambiente para alteração de esquema	×	×	×	×	×	×
Adaptação de documentos em sistemas de arquivos	NA	×	×	×	√	√
Adaptação sem intervenção do administrador	NA	NA	√	√	×	×
Evolução on-line	−−	×	√	√	√	−−
Restrições de integridade dos esquemas	√	√	√	√	√	NA
Restrições de integridade dos documentos	×	√	√	√	√	√
Modelo de dados	×	×	×	×	×	×
Taxonomia de modificações	×	×	×	×	×	×

Legenda: √ Possui × Não possui −− Não conhecido NA Não Aplicável

Tabela 3.2: Comparativo de trabalhos relacionados: modificação de esquemas e propagação de mudanças em implementações comerciais

Itens / Propostas	**DB2**	**Oracle**	**SQL Server**	**Tamino**
Propagação de modificações	×	√	√	×
Independência de ambiente para alteração de esquema	×	×	×	×
Adaptação de documentos em sistemas de arquivos	×	×	×	×
Adaptação sem intervenção do administrador	√	×	×	×
Evolução on-line	√	√	√	√
Restrições de integridade dos esquemas	√	√	√	√
Restrições de integridade dos documentos	√	√	√	√
Modelo de dados	×	×	×	×
Taxonomia de modificações	×	×	×	×

Legenda: √ Possui × Não possui −− Não conhecido

na proposta ou implementação, uma vez que o enfoque ao suporte da evolução de esquemas pode ser como ocorre no IBM DB2, onde os esquemas XML não são modificados, e sim versionados, caso haja necessidade de definição de novas restrições no esquema.

Uma segunda consideração que pode ser extraída da análise das Tabelas 3.1 e 3.2 baseia-se no fato de que diversas propostas acadêmicas, como os trabalhos de Bertino, Su e Al-Jadir utilizam bancos de dados relacionais ou orientados a objetos para armazenamento dos documentos XML com referências a esquemas. Desta dependência, decorre o fato que documentos XML armazenados diretamente em sistemas de arquivos não são contemplados pelo processo de adaptação de documentos descrito por estes trabalhos, uma vez que estes operam com documentos já armazenados em bancos de dados.

Também decorrente do fato que diversas propostas armazenam documentos XML em bancos de dados relacionais ou orientados a objetos, alguns trabalhos como os de Su, Al-Jadir, e as próprias implementações do SQL Server e DB2, não permitem a execução de modificações nos esquemas que levem a base de dados a um estado inconsistente. Esta abordagem é bastante semelhante às abordagens de bancos de dados tradicionais, mas nem sempre pode se aplicar a bases de dados semiestruturados dada a sua natureza disjunta e distribuída, que leva ao desconhecimento do conjunto total de documentos que referenciam um esquema, sendo que estes documentos podem encontrar-se em diferentes nodos de uma rede de dados.

De acordo com as propostas analisadas, a modificação dos esquemas XML em si depende de um ambiente específico, seja um banco de dados, como é o caso das implementações Oracle, DB2 e SQL Server, ou de uma aplicação, como é o caso das propostas de Guerrini, Su e Al-Jadir. Isto anula uma das características dos artefatos XML, que consiste na facilidade de edição dos artefatos, que pode ser feita através de uma aplicação

como um simples editor de textos.

Ainda, pode-se observar que o processo de adaptação dos documentos XML é dependente do administrador da base de dados. O trabalho de Guerrini e a implementação do Oracle exigem que o usuário administrador da base de dados semiestruturados envolva-se diretamente no processo de adaptação dos documentos tornados inconsistentes em função de modificações em um esquema XML. A intervenção do administrador na adaptação dos documentos pode constituir-se em um problema uma vez que as informações necessárias para a adaptação dos documentos podem ser pertencentes ao domínio próprio de uma determinada aplicação, e desconhecida ao administrador da base de dados.

Finalizando a análise do envolvimento do administrador da base de dados na adaptação dos documentos, tem-se que os trabalhos de Su e Al-Jadir, além da implementação do DB2, não exigem tal envolvimento. Entretanto, há de se notar que estes trabalhos adotam uma abordagem semelhante à relacional, ou seja, modificações que levem a base semiestruturada a um estado inconsistente são rejeitadas. Desta maneira, não há um processo de adaptação propriamente dito a ser executado.

Mesmo a proposta de Bouchou, que modela a evolução de documentos XML com processos de revalidação parcial e adaptação de documentos a esquemas, possui características semelhantes às propostas de evolução de esquemas XML de Guerrini e do Oracle, como a exigência da intervenção do usuário no processo de adaptação dos documentos XML. Assim como as propostas de Su e Al-Jadir, há a necessidade de uma estrutura de suporte à evolução, neste caso, um *framework* de evolução, que pode não ser compatível com todos os cenários de utilização de artefatos XML.

Até mesmo o Tamino, que é uma implementação nativa de um banco de dados XML, oferece suporte limitado à evolução de esquemas XML, permitindo que apenas um subconjunto das possíveis modificações em esquemas XML sejam realizadas em esquemas gerenciados por este SGBD. Embora ofereça suporte adequado ao processamento de transações e consultas, o Tamino ainda devem consolidar o suporte à evolução de esquemas, que atualmente é bastante restrito.

Entretanto, mesmo propostas que oferecem suporte limitado à evolução de esquemas, como é o caso do Tamino, garantem a integridade interna do esquema após a sua evolução. Ou seja, o esquema resultante da aplicação das operações de modificação será válido quando analisado isoladamente. As propostas garantem tal integridade de diferentes maneiras, seja através de validações executadas previamente à confirmação das alterações, como ocorre na proposta de Al-Jadir, seja através de validações fornecidas por SGBDs relacionais, como ocorre na proposta de Bertino. Naturalmente, é imperativo que as operações de modificação executadas em um esquema gerem um novo esquema também válido, a fim de que havendo a necessidade de execução de um processo de adaptação de documentos, este seja executado tomando por base um esquema correto.

De maneira semelhante, encerradas as modificações em um esquema e finalizado o processo de adaptação dos documentos, estes devem ser garantidamente válidos. Dentre as propostas analisadas, todas aquelas que operam com adaptação de documentos, garantem a integridade dos documentos em relação ao esquema, sendo que algumas propostas lançam mão de recursos como a intervenção do administrador do esquema na adaptação dos documentos, como é o caso da proposta de Guerrini e da implementação Oracle. Há de se notar que cenários que não permitam a intervenção do administrador do esquema no processo de adaptação de documentos podem levar à geração de documentos não válidos em relação ao esquema, como por exemplo, quando a cardinalidade mínima de elementos é aumentada, ou quando novas estruturas de dados são definidas em um esquema. Na

maior parte dos casos, somente a intervenção do usuário final da aplicação resultará na criação de documentos válidos em relação a um esquema XML e válidos do ponto de vista das regras de negócio de uma aplicação.

Por fim, há de se notar que nenhuma das propostas analisadas possui suporte ao registro da evolução de um esquema XML. Tal característica permite o fácil levantamento de modificações realizadas ao longo do tempo em um esquema, com objetivo de uma melhor compreensão do processo evolutivo da base de dados.

3.4 Considerações finais

Este capítulo apresentou a abordagem de diferentes propostas e implementações para o processo de evolução de esquemas XML e a propagação destas modificações para os documentos que referenciam estes esquemas. A partir da análise destes trabalhos é possível identificar algumas características desejáveis que não são abordadas nestes trabalhos:

- facilidade de edição de esquemas XML: garantir que a edição de esquemas XML poderá ser realizada através de qualquer aplicação destinada a este fim, sem necessidade de utilização de uma interface específica para modificação do esquema;

- suporte às operações essenciais de modificação em esquemas XML: embora alguns dos trabalhos realizem a propagação das modificações para os esquemas, nem sempre todas as possíveis modificações em um esquema XML são suportadas, uma vez que determinadas alterações são bloqueadas, pois poderiam levar a base de dados a um estado inconsistente em função da existência de valores nos documentos considerados inválidos em relação à versão mais nova do esquema;

- adaptação de documentos XML armazenados diretamente em sistemas de arquivos e mensagens XML trocadas em uma rede de dados: documentos encontrados em sistemas de arquivos não são adaptados pela maioria das propostas enquanto que mensagens XML trafegadas entre aplicações não são contempladas por nenhuma proposta, uma vez que muitas destas propostas apóiam-se em SGBDs relacionais para armazenamento de seus documentos;

- adaptação de documentos sem envolvimento do usuário administrador da base de dados: os trabalhos analisados necessitam de envolvimento do administrador do banco de dados na adaptação dos documentos XML, seja através da modificação manual dos documentos, ou através da execução de *scripts* XSL. Em alguns casos, as informações necessárias à adaptação dos documentos podem ser próprias ao domínio da aplicação, e desconhecidas ao administrador da base de dados. Nestes casos, o usuário final da aplicação é responsável pela especificação destes valores.

A W3C não possui definição oficial sobre como gerenciar a evolução de esquemas XML. Tampouco há definição sobre como gerenciar o impacto que a evolução de esquemas possui em documentos e aplicações que referenciem estes esquemas. As propostas existentes para abordagem dos impactos gerados pela evolução de esquemas XML focam-se em apenas partes do problema: mensagens XML não são adaptadas às modificações realizadas em esquemas, diferentes aplicações exigem a intervenção direta de um usuário no processo de adaptação de instâncias do banco de dados, o que inviabiliza a adaptação de mensagens XML trafegadas entre aplicações em uma rede de dados ou a adaptação

de documentos às modificações dos esquemas exigem o emprego de aplicações altamente complexas como SGBDs relacionais.

A ausência de especificações por parte da W3C e as características identificadas nas demais propostas, que abordam apenas determinados cenários de uso de bases de dados semiestruturados, motivam o desenvolvimento deste trabalho, onde é definido um mecanismo de propagação da evolução de esquemas XML aos documentos em diferentes cenários de distribuição, sem intervenção do administrador da base de dados. O mecanismo que aborda estes tópicos é apresentado no próximo capítulo.

4 X-SPREAD

Este capítulo especifica formalmente o X-Spread - Um mecanismo automático para propagação da evolução de esquemas para documentos XML. A principal contribuição desse mecanismo é a definição de estratégias para a propagação das modificações em esquemas XML para os documentos que referenciam estes esquemas. O mecanismo aborda diferentes cenários de distribuição de esquemas e documentos XML e objetiva identificar modificações realizadas em esquemas XML e propagá-las a documentos XML a fim de torná-los novamente compatíveis com os esquemas modificados. A adaptação dos documentos é realizada sem o envolvimento do usuário administrador da base de dados e possui desempenho que insere atrasos mínimos nos seus diferentes cenários de atuação.

Este capítulo está organizado da seguinte forma: as abstrações de esquemas e documentos XML são apresentadas nas Seções 4.1 e 4.2. Estas abstrações são utilizadas na Seção 4.3, onde é definido o impacto que as modificações nos esquemas causam sobre documentos. A definição dos processos de revalidação associados a cada operação de modificação do esquema é apresentada na Seção 4.4, enquanto a Seção 4.5 define o processo de adaptação de documentos XML em estado inválido. Por fim, a Seção 4.6 apresenta considerações finais sobre a especificação do mecanismo.

Um artigo apresentando a visão geral do X-Spread foi publicado nos anais do Workshop de Teses e Dissertações em Bancos de Dados, no escopo do XXI Simpósio Brasileiro de Banco de Dados (SBBD2006) (SILVEIRA; GALANTE, 2006).

4.1 Abstração de esquemas XML

A abstração de esquemas XML adotada neste trabalho visa compreender o conjunto de definições que formam o W3C XML Schema. As definições do XML Schema englobam as definições de DTDs, por possuírem maior expressividade, o que traduz-se em controle não só da estrutura em si do documento, mas também dos valores aceitos pelos elementos do documento.

Uma restrição aplica-se na abstração de esquemas em formato XML Schema: esquemas são tratados como árvores, ao possuírem um elemento raiz. Considerando-se esta restrição, um esquema XML pode ser abstraído através de um grafo direcionado.

Definição 1 *Esquemas S são grafos ordenados definidos pelos conjuntos P, O, T e a função* tipo*, onde:*

- *P é o conjunto de partículas definidas no esquema;*
- *T é um conjunto de tipos de dados de usuário definidos no esquema S;*

- *O é um conjunto de pares ordenados o = (p_1, p_2), $p_1 \in (P \cup T \cup \epsilon)$, $p_2 \in (P \cup T)$, onde p_2 é descendente de p_1 em S;*

- *A função parcial* tipo *mapeia partículas de um esquema* S *a tipos de dados de usuário ou tipos pré-definidos no esquema XML:*

  tipo*: $P \rightarrow (T \cup$ tipos de dados de esquema)*

De acordo com a Definição 1, pares pertencentes ao conjunto O cuja primeira componente são iguais a ϵ representam partículas na raiz do esquema, sendo que um número irrestrito de partículas pode encontrar-se na raiz do esquema. O restante das partículas segue a estrutura hierárquica do esquema, onde a primeira componente dos demais pares do conjunto O possui referência a uma partícula definida nos conjuntos P ou T.

Os possíveis resultados da função *tipo* encontram-se no conjunto resultante da união do conjunto de tipos de usuário *T* e de tipos pré-definidos em um dado formato de esquema XML, aqui denominado TDE (*tipos de dados de esquema*). Esta função é parcial, pois ela encontra-se definida somente para partículas que fazem referência a um tipo de dados em sua definição.

Definição 2 *Um esquema S pode ser definido através de seus componentes, como S(s_{Rn}, s_1, s_2, ..., s_n, s_{Rn+1}, $s_n + 1$, ..., s_m), onde:*

- *s_{Rn} corresponde a um elemento localizado na raiz do esquema;*

- *s_1, s_2, ..., s_n correspondem aos elementos contidos por s_{Rn};*

- *s_{Rn+1} corresponde a um segundo elemento localizado na raiz do esquema;*

- *$s_n + 1$, ..., s_m correspondem aos elementos contidos por s_{Rn+1}.*

A Definição 2 aborda o fato de que esquemas XML podem ser encarados como composições de esquemas menores. Estes esquemas menores correspondem a elementos definidos no esquema, sendo estas definições auto-contidas.

Definição 3 *Toda partícula* p *possui as seguintes propriedades:*

- *id - identificador único da partícula de esquema;*

- *nome - nome da partícula no esquema;*

- *cardinalidade - número de ocorrências possíveis de uma determinada partícula em um documento XML. Os possíveis valores para esta propriedade são:*

  - *(0, 1): partículas opcionais;*
  - *(1, 1): partículas obrigatórias;*
  - *cardinalidade.min = p.cardinalidade$_1$;*
  - *cardinalidade.max = p.cardinalidade$_2$;*

- *domínio - define o domínio de uma partícula conforme especificado pelo autor do esquema.*

O atributo *domínio* de uma partícula de esquema, especificada na Definição 3, comporta valores especificados pelo autor do esquema ou uma expressão regular, que definem os valores aceitáveis em porções de documentos XML correspondentes a esta partícula. Já o atributo *id* possui seu valor definido a partir do caminhamento pré-definido da hierarquia de partículas de um esquema. Todas as partículas na raiz da hierarquia são visitadas inicialmente, sendo toda a subárvore à esquerda de uma partícula visitada antes que outro descendente à direita de uma partícula seja visitado.

As partículas definidas em um esquema são especializadas em elementos e atributos. Partículas genéricas conforme a Definição 3 não existem propriamente nesta abstração de esquemas XML: elas definem propriedades compartilhadas por elementos e atributos, que possuem regras específicas de formação ou restrição conforme o tipo de especialização. Logo, pode-se dizer que o conjunto P, que contêm as partículas definidas em um esquema, é formado pela união de conjuntos de elementos e atributos, detalhados a seguir.

Definição 4 *Todo elemento* e *possui as propriedades definidas em uma partícula de esquema* p*, além das seguintes propriedades:*

- *raiz - identifica se um elemento é encontrado na raiz do esquema. Todo esquema possui um conjunto não vazio de elementos onde esta propriedade possui valor igual a* verdadeiro*;*
- *cardinalidade - a definição da propriedade* cardinalidade *estende a definição de original de partículas, pois permite múltiplas ocorrências de um elemento:*
  - *(*m*,* n*): elementos com número de incidências variável, variando entre* m *e* n*, sendo* $m \leq n$*,* m*,* $n \in N$*.* n *pode ter um valor ilimitado, representado por* ∞*.*
- *subelementos - define uma expressão regular formada a partir dos subelementos diretos de um elemento do esquema e suas cardinalidades;*
  - |*subelementos*| *- determina a cardinalidade do conjunto de subelementos, independentemente da cardinalidade destas partículas;*
- *atributos - conjunto de atributos de um elemento. O conjunto* atributos *de um elemento* e $\in P$ *é formado por partículas de um tipo especial, denominado atributo.*

Definição 5 *Todo atributo* a *definido em um esquema possui as mesmas propriedades que uma partícula de esquema, entretanto, não podem ser encontrados na definição de um esquema exceto como pertencentes ao conjunto* a.atributos *de um elemento de esquema.*

As Definições 4 e 5 caracterizam dois constituintes de esquemas XML em formato DTD e XML Schema: elementos e atributos. Elementos e atributos possuem correspondentes diretos em documentos XML, e por definição um atributo não pode conter quaisquer outras partículas. Já elementos, podem conter um número ilimitado de subelementos ou atributos. Um terceiro constituinte de esquemas, disponível no formato XML Schema, corresponde a tipos de dado de usuário. Um tipo de dado definido pelo usuário (TDU) é uma estrutura reutilizável em diferentes porções do esquema. Uma partícula com referência a um tipo herda todas as características da estrutura de partículas especificada por este tipo.

Definição 6 *Todo tipo de dado* t *definido pelo usuário possui as seguintes propriedades:*

- *nome - nome do tipo de dado;*
- *tipo - referência não-circular a um conjunto de nomes de outros tipos de dado;*
- *derivação - especifica qual derivação é realizada no tipo de dado especificado pela propriedade* tipo*. Os valores aceitos são* união, lista, restrição *e* extensão*;*
- *subesquema - referência a um subesquema que define a estrutura hierárquica de elementos definida por um tipo de dados complexo. Os elementos referenciados por este subesquema são construídos a partir das disposições das Definições 4 e 5;*
- *subelementos - expressão regular formada a partir dos elementos encontrados na raiz do esquema definido na propriedade* subesquema*.*

De acordo com a Definição 6 qualquer partícula de um esquema pode referenciar um tipo de dado definido pelo usuário. Logo, elementos, atributos e tipos de dado de usuário podem referenciar um TDU. Esta possibilidade visa modelar as diferentes derivações de tipos definidas no formato XML Schema, como união de tipos, lista de tipos, restrição e extensão de tipos.

A função L(t), que recebe como parâmetro um tipo de dado definido no conjunto T, resulta na linguagem gerada por um tipo *t*, ou seja, todos os possíveis valores aceitáveis de acordo com a estrutura definida por *t*. Um dos propósitos da função L(t) é identificar se valores especificados em elementos de documentos XML atendem às restrições definidas por tipos de dados. Isto é atingido verificando-se se um determinado valor é contido pela linguagem gerada por *t*.

A Definição 6 referencia diferentes formas de derivação de tipos, especificadas através da propriedade *derivação*. Esta propriedade suporta os diferentes tipos de derivação definidos pelo formato W3C XML Schema, onde cada derivação restringe a estrutura do esquema de uma forma diferente. Estas restrições consistem em três diferentes casos:

- nas derivações *união* e *lista*, identificar se os tipos referenciados pelo parâmetro *tipo* são tipos válidos, definidos no conjunto T do esquema S;
- na derivação *restrição*, identificar se a linguagem gerada pelo esquema referenciado por *subesquema* é contida pela linguagem gerada pelo tipo referenciado por *tipo*;
- na derivação *extensão*, identificar se não há colisão de nomes entre os elementos de *subesquema* e o subesquema dos tipos referenciados por *tipo*.

Tais restrições podem ser verificadas através da função *verificaTipo*, que recebe como parâmetros um conjunto de tipos, a derivação adotada para referência a estes tipos, um subesquema a considerar em relação ao conjunto de tipos e uma referência ao esquema XML. A função *verificaTipo* é assim definida em pseudo-código pelo Algoritmo 1:

Algoritmo 1 Função *verificaTipo* - testa se *tipo* emprega *derivação* corretamente

```
1: verificaTipo ⇐ false
2: if tipo ⊆ (S.T ∪ Tipos definidos no formato de esquema XML) then
3:    if derivação = lista ∨ derivação = união then
4:       verificaTipo ⇐ true
5:    else if derivação = restrição then
6:       if L(subesquema) ≤ L(tipo.subesquema) then
7:          verificaTipo ⇐ true
8:       end if
9:    else if derivação = extensão then
10:      if ∀ o ∈ subesquema | o = (ϵ, p) ∧ ∀ o' ∈ tipo.subesquema | o' = (ϵ, p'), ∄ p.nome
         = p'.nome then
11:         verificaTipo ⇐ true
12:      end if
13:   end if
14: end if
15: return verificaTipo
```

Esquemas construídos a partir das disposições das Definições 1 a 6 são ditos **válidos**. Formalmente, um esquema válido atende às definições do Algoritmo 2:

Algoritmo 2 Validade de um esquema

```
1: S é válido
2: for all p ∈ P, ∃ o ∈ O | o = (p_A, p) ∧ S é válido do
3:    cardinalidadeMínima ⇐ (p.cardinalidade.min ≤ p.cardinalidade.max)
4:    cardinalidadeVálida ⇐ (p.cardinalidade ∈ {(0,1), (1, 1), (m, n)})
5:    nomePartículaDistinto ⇐ (∄ o' ∈ O | o' = (p_A, p') ∧ p'.nome = p.nome)
6:    nomeAtributoDistinto ⇐ (∄ a ∈ p.atributos, ∄ a' ∈ p.atributos | a.nome = a'.nome
      ∧ a.cardinalidade ∈ {(0, 1), (1, 1)}))
7:    if cardinalidadeMínima = false ∨ cardinalidadeVálida = false ∨ nomePartícula-
      Distinto = false ∨ nomeAtributoDistinto = false then
8:       S não é válido
9:    end if
10: end for
11: for all t ∈ T, ∃ o ∈ O | o = (t_A, t) ∧ S é válido do
12:   derivaçãoVálida ⇐ (t.derivação ∈ {união, lista, restrição, extensão})
13:   tiposReferidosExistem ⇐ (t.tipo ⊂ T)
14:   subesquemaVálido ⇐ true
15:   if t.subesquema ≠ ∅ ∧ t.subesquema não é válido then
16:      subesquemaVálido ⇐ false
17:   end if
18:   if derivaçãoVálida = false ∨ tiposReferidosExistem = false ∨ subesquemaVálido
      = false then
19:      S não é válido
20:   end if
21: end for
```

4.1.1 Exemplo de abstração de esquema XML

A fim de exemplificar as definições apresentadas, a Figura 4.1 apresenta um esquema XML codificado em formato XML Schema. Este esquema define a estrutura de dados utilizada para armazenamento de informações sobre departamentos de uma instituição, como funcionários alocados no departamento e o supervisor responsável pelo departamento. A representação gráfica da abstração deste esquema é ilustrada na Figura 4.2. As propriedades das partículas do esquema não são apresentadas na Figura 4.2 a fim de manter a clareza do texto.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified"
attributeFormDefault="unqualified">
 <xs:element name="departamento">
  <xs:complexType>
   <xs:sequence>
    <xs:element name="funcionarios">
     <xs:complexType>
      <xs:sequence>
       <xs:element name="funcionario" type="tipoPessoa" maxOccurs="unbounded"/>
      </xs:sequence>
     </xs:complexType>
    </xs:element>
    <xs:element name="localizacao">
     <xs:complexType>
      <xs:sequence>
       <xs:element name="endereco" type="xs:string"/>
       <xs:element name="cep" type="xs:string"/>
       <xs:element name="responsavel" type="xs:string"/>
      </xs:sequence>
     </xs:complexType>
    </xs:element>
    <xs:element name="nome" type="xs:string"/>
   </xs:sequence>
  </xs:complexType>
 </xs:element>
 <xs:complexType name="tipoPessoa">
  <xs:sequence>
   <xs:element name="cargo" type="xs:string"/>
   <xs:element name="nome" type="xs:string"/>
   <xs:element name="ramal" type="xs:integer"/>
  </xs:sequence>
 </xs:complexType>
</xs:schema>
```

Figura 4.1: Esquema XML em formato XML Schema

A Tabela 4.1 ilustra as propriedades dos elementos do esquema definido na Figura 4.1. Conforme a Definição 4, o elemento *departamento* possui a propriedade *raiz* com valor igual a *verdadeiro*. A propriedade *subelementos* contém a expressão que define a ordenação e cardinalidade dos elementos descendentes de cada elemento do esquema. Subelementos com cardinalidade opcional ou múltipla, como *funcionario*, possuem as suas cardinalidades sufixadas a seus nomes nas expressões que definem os subelementos dos elementos, como em *funcionários*, por exemplo.

Conforme ilustrado nas Figuras 4.1 e 4.2, o elemento *localizacao* possui um atributo, sendo suas propriedades descritas na Tabela 4.2. A especificação do formato XML, e conseqüentemente de seus esquemas, define que atributos de esquemas não possuem ordenação. Logo, a ordenação de atributos definida em um esquema não necessita ser respeitada pelos documentos. Entretanto, a fim de facilitar a identificação unívoca dos atributos de elemento, a estes é associado um índice na abstração apresentada.

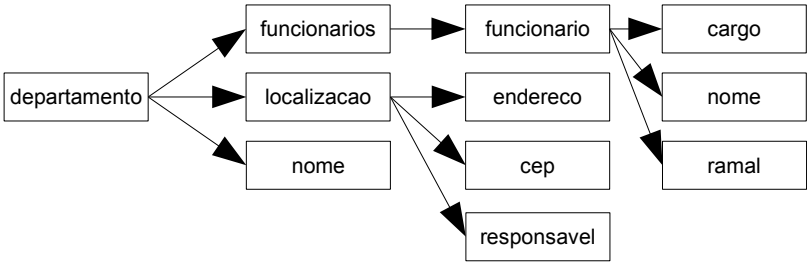

Figura 4.2: Representação gráfica da abstração de um esquema XML

Tabela 4.1: Abstração de elementos de esquema XML

Nome	Id	Tipo	Cardinalidade	Subelementos	Raiz
departamento	1	−−	$\langle 1, 1 \rangle$	funcionários localizacao nome	*verdadeiro*
funcionarios	1.1	−−	$\langle 1, 1 \rangle$	funcionario*	*falso*
funcionário	1.1.1	tipoPessoa	$\langle 0, \infty \rangle$	nome cargo ramal	*falso*
cargo	1.1.1.1	xs:string	$\langle 1, 1 \rangle$	−−	*falso*
nome	1.1.1.2	xs:string	$\langle 1, 1 \rangle$	−−	*falso*
ramal	1.1.1.3	xs:integer	$\langle 1, 1 \rangle$	−−	*falso*
localizacao	1.2	−−	$\langle 1, 1 \rangle$	endereco cep responsavel	*falso*
endereco	1.2.1	xs:string	$\langle 1, 1 \rangle$	−−	*falso*
cep	1.2.1	xs:string	$\langle 1, 1 \rangle$	−−	*falso*
nome	1.3	xs:string	$\langle 1, 1 \rangle$	−−	*falso*

Legenda: −− Não possui

Tabela 4.2: Atributos de um elemento do tipo *tipoDepartamento*

Nome	Id	Tipo	Cardinalidade
responsavel	1.2.3	xs:string	$\langle 1, 1 \rangle$

A Tabela 4.2 não descreve as propriedades *subelementos* e *raiz* do atributo *responsavel* uma vez que atributos possuem apenas as propriedades especificadas para partículas genéricas de um esquema XML, conforme as Definições 3 e 5. Atributos de esquema são partículas terminais em um esquema, de maneira que não podem conter outros elementos ou atributos. Tanto os elementos quanto os atributos definidos no esquema ilustrado na Figura 4.1 não possuem domínios de usuário especificados, portanto a propriedade *domínio* não é apresentada nas tabelas de propriedades do esquema.

O mesmo esquema S, quando representado através somente de seus elementos, desprovidos de atributos, é assim definido:

- P = {departamento, funcionarios, funcionario, cargo, funcionario.nome, ramal, localização, endereço, cep, responsável, nome}

- O = {(ϵ, departamento), (departamento, funcionarios), (funcionarios, funcionario), (funcionario, ramal), (funcionario, nome), (funcionario, ramal), (departamento, localização), (localização, endereço), (localização, cep), (localização, responsável), (departamento, nome)}

- T = {tipoPessoa}

4.2 Abstração de documentos XML

A abstração apresentada a seguir modela características do formato XML conforme definido pela W3C. Documentos XML podem ser modelados como árvores ordenadas, onde a ordem entre os descendentes de um dado elemento é fator determinante na composição da árvore.

Definição 7 *Documentos XML podem ser modelados como uma árvore ordenada D(E, L,* rótulo*,* valor*,* tipoNodo*), onde:*

- *E é o conjunto de nodos estruturais da árvore, ou seja, elementos e atributos;*
- *L define o conjunto de vértices da árvore, formado por pares ordenados de nodos pertencentes ao conjunto E, tal como* $(e_1, e_2)_i$*, onde* e_1 *é o elemento ascendente de* e_2*, e o índice* i *indica a posição de* e_2 *dentre os descendentes de* e_1*. A raiz da árvore possui como primeira componente um elemento vazio:* l = $(\epsilon, e_R)_0$, $e_R \in E$*;*
- *Dado um alfabeto* Σ_R *de rótulos, a função* rótulo *mapeia nodos do conjunto E a palavras contidas no conjunto* Σ_R+*;*

  rótulo*:* $E \rightarrow \Sigma_R$+
- *Dado um alfabeto* Σ_V *de valores, a função* valor *mapeia nodos da árvore a seus respectivos valores, sendo estes valores definidos no conjunto de palavras formadas por* Σ_V**;*

  valor*:* $E \rightarrow \Sigma_V$*
- *Dado o conjunto de nodos E, a função* tipoNodo *retorna* atributo *para os nodos que representam atributos de elementos no documento XML, e* elemento *para os demais nodos.*

Documentos XML admitem a presença de apenas um elemento como raiz do documento, o que gera a restrição na Definição 7 sobre a formação do conjunto *L*. A notação e_i indica o documento XML com raiz no i-ésimo descendente do nodo e, dada a estrutura de árvore definida no conjunto E, enquanto |e| representa a profundidade da subárvore com raiz em e.

A definição da função *rótulo* indica que seus resultados são mapeados para o conjunto Σ_R+, que corresponde ao conjunto de todas as possíveis palavras formadas pelo alfabeto Σ_R à exceção da palavra vazia, simbolizada por ϵ. Desta maneira, garante-se que todo vértice da árvore possui um rótulo associado.

A função *corresp* realiza a correspondência entre um nodo n de um documento d e uma partícula p de um esquema S. A função verifica se o nodo p atende à cadeia de descendentes definida pelo esquema S, além de fazer uso da função *adj*. A função *adj* identifica se os nodos adjacentes a n também são correspondentes aos elementos adjacentes a p no esquema S. As duas funções são definidas através de pseudo-código nos Algoritmos 3 e 4:

Algoritmo 3 Função *corresp* - identifica se um nodo *n* do documento *d* corresponde à partícula *p* do esquema *S*

1: *corresp* $\Leftarrow$ **false**
2: **if** $\exists$ S' $\subseteq$ S | S'.P = { p_0, p_{00}, ..., p_{0m}, p_1, p_{10}, ..., p_{1m}, ..., p_n, p_{n0}, ..., p_{nm} } **then**
3: encadeamentoElementos $\Leftarrow$ ($\forall$ n | n > 0 $\wedge$ n < |S'.P|, p_n.nome $\in$ (p_{n-1}.subelementos $\cup$ p_{n-1}.atributos))
4: adjacênciasElementos $\Leftarrow$ ($\forall$ p $\in$ S'.P, $\exists$ (n_A, n) $\in$ d.L $\rightarrow$ (adj(n, d, p, S) = verdadeiro))
5: **if** encadeamentoElementos = **true** $\wedge$ adjacênciaElementos = **true then**
6: *corresp* $\Leftarrow$ **true**
7: **end if**
8: **end if**
9: **return** *corresp*

Algoritmo 4 Função *adj* - identifica se um nodo *n* do documento *d* encontra-se corretamente posicionado no documento em relação aos elementos adjacentes da partícula *p* no esquema *S*

1: *adj* $\Leftarrow$ **false**
2: **if** $\exists$ (p_A, p) $\in$ S.O | p = p_A.subelementos$_i$ **then**
3: p_E $\Leftarrow$ { p_A.subelementos$_0$ ... p_A.subelementos$_{i-1}$ }
4: p_D $\Leftarrow$ { p_A.subelementos$_{i+1}$ ... p_A.subelementos$_{|p_A.subelementos|-1}$
5: **if** $\exists$ (n_A, n) $\in$ d.L $\rightarrow$ n = $(n_A)_j$ **then**
6: $n_E = \epsilon$
7: $n_D = \epsilon$
8: **if** j > 0 $\wedge$ $\sum_{i=j-1}^{0}$ $n_E = \epsilon$ $\wedge$ *rótulo*$(n_A)_i \neq$ *rótulo*(n) **then**
9: $n_E = (n_A)_i$
10: **end if**
11: **if** j < |n_A| - 1 $\wedge$ $\sum_{i=j+1}^{|n_A|-1}$ $n_D = \epsilon$ $\wedge$ *rótulo*$(n_A)_i \neq$ *rótulo*(n) **then**
12: $n_D = (n_A)_i$
13: **end if**
14: **if** ($n_E = \epsilon \wedge p_E = \emptyset$) $\vee$ ($n_E \neq \epsilon \wedge \exists$ $p_T \in p_E$ | *rótulo*(n_E) = p_T.nome) **then**
15: **if** ($n_D = \epsilon \wedge p_D = \emptyset$) $\vee$ ($n_D \neq \epsilon \wedge \exists$ $p_T \in pDE$ | *rótulo*(n_D) = p_T.nome) **then**
16: *adj* $\Leftarrow$ **true**
17: **end if**
18: **end if**
19: **end if**
20: **end if**
21: **return** *adj*

4.2.1 Exemplo de abstração de documento XML

A fim de exemplificar a definição de abstração de documentos XML, a Figura 4.3 apresenta um documento XML compatível com o esquema XML definido em 4.1.1. Tal documento pode ser abstraído conforme os dados apresentados na Tabela 4.3.

Conforme a Definição 7, apenas um nodo encontra-se na raiz do documento. No caso do documento apresentado na Figura 4.3, o nodo raiz é *departamento*. Este nodo possui um atributo, *responsável*, identificado pelo retorno da função *tipoNodo*. Nodos sem valores, como *funcionários*, retornam ϵ quando submetidos à função *valor*.

```
<?xml version="1.0" encoding="UTF-8"?>
<departamento xsi:noNamespaceSchemaLocation="esquema1.xsd"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <funcionarios>
    <funcionario>
      <cargo>Analista de sistemas</cargo>
      <nome>Funcionário 1</nome>
      <ramal>414</ramal>
    </funcionario>
  </funcionarios>
  <localizacao responsavel="Supervisor 1">
    <endereco>Av. Bento Gonçalves, 9500</endereco>
    <cep>90000-000</cep>
  </localizacao>
  <nome>Departamento de TI</nome>
</departamento>
```

Figura 4.3: Exemplo de documento XML

Tabela 4.3: Abstração de um documento XML

Nodo	Vértice	*rótulo(nodo)*	*valor(nodo)*	*tipoNodo(nodo)*
departamento	⟨ϵ, departamento⟩	departamento	ϵ	*elemento*
funcionarios	⟨departamento, funcionarios⟩	funcionarios	ϵ	*elemento*
funcionario	⟨funcionarios, funcionario⟩	funcionario	ϵ	*elemento*
cargo	⟨funcionario, cargo⟩	cargo	Analista de sistemas	*elemento*
nome	⟨funcionario, nome⟩	nome	Funcionário 1	*elemento*
ramal	⟨funcionario, ramal⟩	ramal	414	*elemento*
localizacao	⟨departamento, localizacao⟩	localizacao	ϵ	*elemento*
responsavel	⟨localizacao, responsavel⟩	responsavel	Supervisor 1	*atributo*
endereco	⟨localizacao, endereco⟩	endereco	Av. Bento Gonçalves, 9500	*elemento*
cep	⟨localizacao, cep⟩	cep	90000-00	*elemento*
departamento.nome	⟨departamento, nome⟩	nome	Departamento de TI	*elemento*

O mesmo documento D é representado através de seus conjuntos de nodos e vértices, bem como as funções *rótulo*, *valor* e *tipoNodo*, sendo estes conjuntos assim definidos:

- E = {departamento, funcionarios, funcionario, cargo, funcionario.nome, ramal, localizacao, reponsavel, endereco, cep, nome}
- L = { (ϵ, departamento), (departamento, funcionarios), (funcionarios, funcionario), (funcionario, cargo), (funcionario, funcionario.nome), (funcionario, ramal), (departamento, localizacao), (localizacao, endereco), (localizacao, cep), (departamento, nome) }

4.3 Operações de modificação de esquemas

Dada a grande gama de aspectos abordados pelos formatos de esquemas XML mais difundidos atualmente, tem-se que também o número de modificações possíveis nestes esquemas é alto. Estas modificações podem afetar a organização das hierarquias de elementos definidas no esquema ou mesmo valores permitidos nos elementos dos documentos regidos por um esquema. O controle de valores de elementos ocorre através da especificação de listas de valores permitidos ou de tipos de dados associados a elementos.

No que se refere à consistência do conjunto formado por esquema e documentos XML, as diferentes operações de modificação de esquemas podem ser classificadas em operações **neutras** e **desestabilizadoras**. Operações **neutras** são aquelas que mantêm o banco de dados semiestruturado em estado consistente, enquanto operações **desestabilizadoras** são aquelas que levam a base de dados a um estado inconsistente. Exemplos de

operações neutras são a alteração da cardinalidade de um elemento, através do aumento do número máximo de elementos permitidos em uma coleção de elementos, e a definição de um novo tipo de dados de usuário. Exemplos de operações desestabilizadoras são a inclusão de um elemento de cardinalidade obrigatória e a alteração do tipo de dado de um atributo para um tipo mais restrito.

Estendendo-se a definição da função *L*, apresentada na Seção 4.1, a fim de identificar a linguagem gerada por um esquema S, tem-se que operações neutras são descritas pela Definição 8. Operações desestabilizadoras podem ser entendidas como a definição dual de operações neutras, ou seja, são as operações que não atendem à Definição 8.

Definição 8 *Dados os esquemas S e S', sendo S' resultante de* m*(S), toda operação* m *é dita neutra se L(S) = L(S').*

Quanto à consistência interna do esquema XML, algumas operações podem ser ditas **inválidas**, na medida em que levam o esquema a um estado inconsistente. Tal inconsistência pode ser atingida através de referências incorretas a tipos de dados, ou através de especificações incompatíveis de cardinalidades mínimas e máximas para um elemento. Analogamente, operações **válidas** são operações de modificação que sempre originam um novo esquema internamente consistente. Operações válidas são formalizadas na Definição 9:

Definição 9 *Dados os esquemas S e S', sendo S' resultante de* m*(S), toda operação* m *é válida se S e S' são esquemas válidos conforme a definição apresentada na Seção 4.1.*

Quando o administrador da base de dados possui seu trabalho apoiado por uma ferramenta de gerência de esquemas XML, as inconsistências apresentadas por operações inválidas, ou seja, aquelas que não atendem à Definição 9, podem ser detectadas ou bloqueadas pela ferramenta. Caso estas inconsistências sejam efetivadas, ou seja, disponibilizadas ao usuário final, elas serão identificadas somente quando de uma tentativa de validação de um documento em relação a este esquema. Neste caso, a validação será bloqueada, pois o esquema encontra-se em formato inválido.

Considerando-se que o foco deste trabalho é a propagação de modificações de esquemas para documentos, esquemas inconsistentes não terão suas modificações propagadas aos documentos. Logo, a inconsistência interna do esquema fica restrita a este, evitando que documentos válidos tornem-se inconsistentes em função de alterações incorretas no esquema. Assim sendo, as operações de modificação modeladas possuem pré-condições que devem ser garantidas a fim de manter a consistência interna do esquema.

O restante desta seção analisa as diferentes operações de modificação de esquemas XML, e o impacto que estas possuem nas estruturas modeladas em um esquema através da abstração destas modificações nos mesmos termos utilizados na Seção 4.1. Cada operação é analisada a partir de sua execução em um esquema arbitrário S(P, O, T, *tipo*), o que resulta em um novo esquema S', detalhado a partir das características de cada operação. As operações de modificação são sumarizadas na Tabela 4.4, sendo analisadas individualmente em maiores detalhes no restante da seção.

As operações descritas a seguir permitem a construção de uma vasta gama de esquemas XML. A escolha por este conjunto de operações ocorreu de maneira a permitir manipulação das características especificadas pela abstração de esquemas adotada neste trabalho. Além disso, este conjunto de operações de modificação reflete operações especificadas por trabalhos relacionados à evolução de esquemas XML. Entretanto, esquemas

que utilizam recursos de grupos de substituição ou qualificadores de elementos, como elementos finais ou abstratos, não são modelados por este trabalho. Logo, a especificação das operações de modificação não aborda estas características.

Tabela 4.4: Operações de modificação de esquemas XML

Nome	Tipo	Descrição
Inclusão de elemento	×	Inclusão de um novo elemento na hierarquia de elementos do esquema
Alteração de nome de elemento	×	Troca de nome de um elemento existente em um esquema
Exclusão de subesquema	×	Exclusão de um elemento, acompanhada da exclusão de descendentes
Cópia de elemento	×	Inclusão de um elemento a partir da replicação de um elemento já existente
Movimentação de elemento	×	Troca de ancestral de um elemento existente no esquema
Alteração de tipo de partícula	×	Alteração do tipo de dados referenciado por uma partícula do esquema
Alteração de cardinalidades	×	Modificação do número de ocorrências mínimas e máximas de uma partícula
Alteração do domínio de partícula	×	Modificação do conjunto de valores de domínio de uma partícula
Inclusão de atributo	×	Inclusão de novo atributo no conjunto de atributos de um elemento
Exclusão de atributo	×	Exclusão de um atributo associado a um elemento
Alteração de nome de atributo	×	Troca do nome de um atributo associado a um elemento do esquema
Inclusão de tipo de dado	−−	Definição de novo tipo de dados no esquema
Exclusão de tipo de dado	−−	Exclusão de um tipo de dados definido no esquema
Alteração de nome de tipo	−−	Troca do nome de um tipo de dados definido no esquema
Cópia de tipo	−−	Criação de uma cópia de um tipo de dados
Alteração na derivação de tipo	×	Troca do tipo de derivação utilizado para referência a um segundo tipo
Alteração de tipo derivado	×	Alteração do tipo de dados referenciado por um tipo de dados
Alteração de subesquema de tipo	×	Troca do subesquema definido por um tipo de dados complexo

Legenda: × Desestabilizadora −− Neutra

4.3.1 Inclusão de elemento

A inclusão de um elemento p_N em um esquema pode ocorrer em qualquer nível da hierarquia das estruturas modeladas pelo esquema, desde que o nome do novo elemento não seja utilizado no mesmo nível hierárquico onde o elemento será inserido. A inclusão de um elemento p_N como descendente *i* de p, é assim especificada:

- Pré-condição: $\forall$ o $\in$ O | o = (p, p') $\rightarrow$ p' $\in$ P $\wedge$ p'.nome $\neq$ p_N.nome
- Esquema resultante: S'(P', O', T, *tipo*), onde:
  - P' = P $\cup$ p_N
  - O'= O $\cup$ (p, p_N), p $\in$ (O $\cup$ ϵ)
  - p $\neq$ ϵ $\rightarrow$ p.subelementos = p.subelementos$_0$, ... p.subelementos$_{i-1}$, p_N, p.subelementos$_{i+1}$, ..., p.subelementos$_n$
- Classificação da operação: desestabilizadora, se p_N.cardinalidade.min $\geq$ 1

Exemplo
Os exemplos de modificação de esquemas tomam por base o esquema apresentado na Seção 4.2.1. A partir deste esquema, tem-se que a inclusão de um novo elemento p_N de rótulo *id* e de cardinalidade obrigatória como descendente do elemento *departamento*, tal que i é igual a 0, gera um novo esquema S'(P', O', T, *tipo*), tal que:

- P' = {departamento, id, funcionarios, funcionario, cargo, funcionario.nome, ramal, localização, endereço, cep, responsável, nome}
  - departamento.subelementos = (id, funcionarios, nome)

- O' = {(ϵ, departamento), (departamento, id), (departamento, funcionarios), (funcionarios, funcionario), (funcionario, ramal), (funcionario, nome), (funcionario, ramal), (departamento, localização), (localização, endereço), (localização, cep), (localização, responsável), (departamento, nome)}

- T = {tipoPessoa}

4.3.2 Alteração de nome de elemento

Qualquer elemento p_A, descendente de p em um esquema pode ter seu nome alterado para *novoNome*, desde que esta nova denominação não cause conflito com os demais descendentes de p. Considerando-se a abstração de esquemas apresentada, a alteração de nome de elementos é assim especificada:

- Pré-condição: $\forall$ o $\in$ O | o = (p, p') $\rightarrow$ p' $\in$ P $\wedge$ p'.nome $\neq$ *novoNome*

- Esquema resultante: S'(P', O, T, *tipo*), onde:

  - P' = P | $\exists$ $p_A \in$ P | p_A.nome $\Leftarrow$ *novoNome*

- Classificação da operação: desestabilizadora

4.3.3 Exclusão de elemento

A exclusão de um elemento p_E descendente de p é assim especificada:

- Esquema resultante: S'(P', O', T', *tipo*), onde:

  - P' = P - E_E.P, onde E_E corresponde ao subesquema com raiz em p_E, conforme Definição 2
  - O' = O - E_E.O
  - T' = T - E_E.T
  - p $\neq$ ϵ $\rightarrow$ p.subelementos = p.subelementos$_0$, ..., p.subelementos$_{i-1}$, p.subelementos$_{i+1}$, ..., p.subelementos$_j$

- Classificação da operação: desestabilizadora

Exemplo
A exclusão de p_E, elemento *localizacao* descendente de *departamento*, leva à geração de um novo esquema S'(P', O', T, *tipo*), tal que:

- P' = {departamento, funcionarios, funcionario, cargo, funcionario.nome, ramal, nome}

  - departamento.subelementos = (funcionarios, nome)

- O' = {(ϵ, departamento), (departamento, funcionarios), (funcionarios, funcionario), (funcionario, ramal), (funcionario, nome), (funcionario, ramal), (departamento, nome)}

- T = {tipoPessoa}

4.3.4 Cópia de elemento

A cópia de um elemento pode ser entendida como a replicação de um elemento de origem p_O, i-ésimo descendente de p, como j-ésimo descendente p_D de um elemento p_{AD}. A operação de cópia de elemento é assim especificada:

- Pré-condição: $\forall$ o $\in$ O | o = (p_{AD}, p_{AD}') $\rightarrow$ p_{AD}' $\in$ P $\wedge$ p_{AD}'.nome $\neq$ p_D.nome
- Esquema resultante: S'(P', O', T, *tipo*), onde:
  - P' = P $\cup$ p_D
  - O' = O $\cup$ (p_{AD}, p_D)
  - $p_{AD} \neq \epsilon \rightarrow$ p_{AD}.subelementos = p_{AD}.subelementos$_0$, ..., p_{AD}.subelementos$_{i-1}$, p_D, p_{AD}.subelementos$_i$, ..., p_{AD}.subelementos$_k$
- Classificação da operação: desestabilizadora se p_D.cardinalidade.min $\geq$ 1

4.3.5 Movimentação de elemento

A movimentação de elementos de um esquema pode ser ententida como a cópia de um elemento p_O, i-ésimo descendente de p como p_D, j-ésimo descendente de p_{AD}, seguida da exclusão de p_O. Esta operação é assim especificada:

- Pré-condição: $\forall$ o $\in$ O | o = (p_{AD}, p_{AD}') $\rightarrow$ p_{AD}' $\in$ P $\wedge$ p_{AD}'.nome $\neq$ p_D.nome
- Esquema resultante: S'(P', O', T, *tipo*), onde:
  - P' = P - p_O $\cup$ p_D
  - O' = O - (p, p_O) $\cup$ (p_{AD}, p_D)
  - p $\neq \epsilon \rightarrow$ p.subelementos = p.subelementos$_0$, ..., p.subelementos$_{i-1}$, p.subelementos$_{i+1}$, ..., p.subelementos$_j$
  - $p_{AD} \neq \epsilon \rightarrow$ p_{AD}.subelementos = p_{AD}.subelementos$_0$, ..., p_{AD}.subelementos$_{i-1}$, p_D, p_{AD}.subelementos$_i$, ..., p_{AD}.subelementos$_k$
- Classificação da operação: desestabilizadora

4.3.6 Alteração de tipo de partícula

Uma partícula p_A, descendente de p, pode ter seu tipo alterado para *novoTipo*, sendo que este novo tipo pode ser pré-definido pelo formato utilizado para representação do esquema ou definido pelo usuário. A alteração de tipo de partícula é assim especificada:

- Pré-condição: *novoTipo* $\notin$ T $\rightarrow$ *novoTipo* $\in$ TDE
- Esquema resultante: S'(P, O, T, *tipo*), onde:
  - *tipo* | *tipo*(p_A) = *novoTipo*
- Classificação da operação: desestabilizadora, se *novoTipo* $\in$ (ID, IDRef, IDRefs) $\vee$ L(*novoTipo*) $\subset$ L(p_A.tipo)

4.3.7 Alteração de cardinalidades de partícula

A alteração de cardinalidades de uma partícula p_C, i-ésimo descendente de p, consiste na especificação de valores *minCardinalidade* e *maxCardinalidade*, como as cardinalidades mínimas e máximas de p_C. A alteração de cardinalidades de uma partícula desestabiliza a base de dados quando a nova cardinalidade mínima é maior que a original ou quando a nova cardinalidade máxima é inferior à original. Esta operação é assim especificada:

- Pré-condição: *minCardinalidade* $\geq$ *maxCardinalidade*
- Esquema resultante: S'(P', O, T, *tipo*), onde:
  - P' = P | p_C.cardinalidade.min $\Leftarrow$ *minCardinalidade* $\wedge$ p_C.cardinalidade.max $\Leftarrow$ *maxCardinalidade*
- Classificação da operação: desestabilizadora, se p_C.cardinalidade.min < *minCardinalidade* $\vee$ (p_C.cardinalidade.max $\neq \infty \rightarrow$ p_C.cardinalidade.max > *maxCardinalidade*)

4.3.8 Alteração do domínio de partícula

A alteração do domínio de uma partícula p_D consiste na associação de *novoDomínio* à propriedade *domínio* da partícula p_D. Esta operação é assim especificada:

- Esquema resultante: S'(P', O, T, *tipo*), onde:
  - P' = P | p_D.domínio $\Leftarrow$ *novoDomínio*
- Classificação da operação: desestabilizadora, se *novoDomínio* $\subset$ p_D.domínio

4.3.9 Inclusão de atributo

A inclusão de um atributo a_N, descendente do elemento p no esquema S deve atender à pré-condição de unicidade de nome do atributo considerando-se os demais atributos associados ao elemento p. Esta operação é assim especificada:

- Pré-condição: $\forall$ a $\in$ p.atributos $\rightarrow$ a.nome $\neq$ a_N.nome
- Esquema resultante: S'(P', O', T, *tipo*)
  - P' = P $\cup$ a_N
  - O' = O $\cup$ (p, a_N)
  - p.atributos = p.atributos $\cup$ a_N
- Classificação da operação: desestabilizadora, se a_N.cardinalidade.min = 1

Exemplo
A inclusão de um atributo a_N, *id*, no elemento *departamento*, leva à geração de um novo esquema S'(P', O', T, *tipo*), tal que:

- P' = {departamento, id, funcionarios, funcionario, cargo, funcionario.nome, ramal, localização, endereço, cep, responsável, nome}
  - departamento.atributos = (id)

- O' = {(ϵ, departamento), (departamento, id), (departamento, funcionarios), (funcionarios, funcionario), (funcionario, ramal), (funcionario, nome), (funcionario, ramal), (departamento, localização), (localização, endereço), (localização, cep), (localização, responsável), (departamento, nome)}
- T = {tipoPessoa}

4.3.10 Exclusão de atributo

A exclusão de um atributo a_E, descendente do elemento p é modelada pela remoção deste atributo do esquema, sem impacto no restante da estrutura do esquema, uma vez que atributos constituem elementos terminais da estrutura de um esquema.

- Esquema resultante: S'(P', O', T, *tipo*)
  - P' = P - a_E
  - O' = O - (p, a_E)
  - p.atributos = p.atributos - a_E
- Classificação da operação: desestabilizadora

4.3.11 Alteração de nome de atributo

A alteração do nome de um atributo a_A para *novoNome*, descendente do elemento p, demanda análise do conjunto de atributos associados a p, a fim de identificar se o novo nome já é utilizado por um outro atributo. Caso o nome já esteja sendo utilizado, a modificação deve ser bloqueada.

- Pré-condição: $\forall$ a $\in$ p.atributos $\rightarrow$ a.nome $\neq$ *novoNome*
- Esquema resultante: S'(P', O, T, *tipo*), onde:
  - P' = P | $\exists$ $a_A \in$ P | a_A.nome $\Leftarrow$ *novoNome*
- Classificação da operação: desestabilizadora

4.3.12 Inclusão de tipo de dado

A inclusão de um tipo t_N em um esquema pode ocorrer em qualquer nível da estrutura hierárquica definida no esquema representado por uma partícula p, de acordo com a abstração de esquemas apresentada na Seção 4.1. O nome do tipo de dados não deve encontrar-se definido entre os descendentes de p:

- Pré-condição: p.raiz = verdadeiro $\wedge$ $\forall$ o $\in$ O | o = (p, p') $\rightarrow$ p' $\in$ T $\wedge$ $\nexists$ p'.nome = t_N.nome
- Esquema resultante: S'(P, O', T', *tipo*), onde:
  - O' = O $\cup$ (p, t_N)
  - T' = T $\cup$ t_N
- Classificação da operação: neutra

4.3.13 Exclusão de tipo de dado

A exclusão de um tipo de dado t_D, descendente de p é dita válida somente quando não há referências ao tipo t_D por parte de outro tipo de dado definido pelo usuário no esquema. Esta operação é assim especificada:

- Pré-condição: $\forall$ t $\in$ T, $\nexists$ t' | t_D.nome $\in$ t'.tipo $\vee$ $\nexists$ p $\in$ P, p.tipo = t_D.nome
- Esquema resultante: S'(P, O', T', *tipo*), onde:
  - O' = O - (p, t_D)
  - T' = T' - t_D
- Classificação da operação: neutra

4.3.14 Alteração de nome de tipo

A alteração do nome de um tipo t_A, descendente de p, para *novoNome* implica em identificar se outro tipo localizado no mesmo nível hierárquico de t_A já utiliza o novo nome. Caso nenhum tipo seja encontrado e não haja referências de outros tipos ou elementos a t_A, o esquema resultante será consistente. Esta modificação é assim especificada:

- Pré-condição: $\forall$ o $\in$ O | o = (p, p'), $\nexists$ p' $\in$ T | p'.nome = *novoNome* $\wedge$ $\nexists$ e $\in$ P | e.tipo = t_A.nome $\wedge$ $\nexists$ t' $\in$ T | t_A.nome $\in$ t'.tipo
- Esquema resultante: S'(P, O, T', *tipo*), onde:
  - T' = T | $\exists$ t_A $\in$ T | t_A.nome $\Leftarrow$ *novoNome*
- Classificação da operação: neutra

4.3.15 Cópia de tipo

A cópia de um tipo t descendente do elemento p para o novo tipo t_N deve ocorrer com alteração do nome de t_N para *novoNome*, a fim de evitar conflito no nome dos tipos. Esta operação é assim especificada:

- Pré-condição: $\forall$ o $\in$ O | o = (p, t'), t' $\in$ T $\rightarrow$ t'.nome $\neq$ *novoNome*
- Esquema resultante: S'(P, O', T', *tipo*), onde:
  - O' = O $\cup$ (p_D, t_N)
  - T' = T $\cup$ t_N
- Classificação da operação: neutra

Exemplo
A cópia de um tipo t_N, *tipoPessa*, a fim de gerar um novo tipo *tipoSupervisor*, leva à geração de um novo esquema S'(P, O, T', *tipo*), tal que:

- P = {departamento, funcionarios, funcionario, cargo, funcionario.nome, ramal, localização, endereço, cep, responsável, nome}
- O = {(ϵ, departamento), (departamento, funcionarios), (funcionarios, funcionario), (funcionario, ramal), (funcionario, nome), (funcionario, ramal), (departamento, localização), (localização, endereço), (localização, cep), (localização, responsável), (departamento, nome)}
- T = {tipoPessoa, tipoSupervisor}

4.3.16 Alteração na derivação de tipo

A alteração de derivação adotada por um tipo t_M para *novaDerivação* implica em identificar se o subesquema definido por t_M, tipo definido no esquema S, é válido em relação aos tipos referenciados por t_M, caso o conjunto t_M.tipo seja diferente de vazio. Esta modificação é assim especificada:

- Pré-condição: verificaTipo(t_M.tipo, *novaDerivação*, t_M.subesquema, S) = verdadeiro
- Esquema resultante: S'(P, O, T', *tipo*), onde:
  - T' = T | $\exists$ $t_M \in$ T $\rightarrow$ t_M.derivação $\Leftarrow$ *novaDerivação*
- Classificação da operação: desestabilizadora

4.3.17 Alteração de tipo derivado

A alteração tipo derivado por um tipo t_M para *novoTipo* implica em identificar se o conjunto *novoTipo* é contido pelo conjunto T. Caso não seja contido, *novoTipo* deve ser formado por um conjunto de tipos de dados pré-definido no formato de esquema XML em questão. Esta operação de modificação é assim especificada:

- Pré-condição: *novoTipo* $\neq$ t_M.nome verificaTipo(*novoTipo*, t_M.derivação, t_M.subesquema, S) = verdadeiro
- Esquema resultante: S'(P, O, T', *tipo*), onde:
  - T' = T | $\exists$ $t_M \in$ T $\rightarrow$ t_M.tipo $\Leftarrow$ *novoTipo*
- Classificação da operação: desestabilizadora

4.3.18 Alteração de subesquema de tipo

A alteração do subesquema de um tipo t_M para *novoSubEsquema* implica em identificar se a estrutura do subesquema é válida. Esta validação ocorre através da aplicação de todas as operações de modificações descritas ao longo da Seção 4.3. Entretanto, caso um tipo t_M derive um outro tipo definido em S.T, é necessário identificar se a estrutura do tipo atende às restrições impostas para derivação de tipos. Tais validações e a operação de modificação em si são assim especificadas:

- Pré-condição: t_M.tipo $\neq$ $\emptyset$ $\rightarrow$ verificaTipo(t_M.tipo, t_M.derivação, *novoSubEsquema*, S) = verdadeiro
- Esquema resultante: S'(P, O, T', *tipo*), onde:
  - T' = T | $\exists$ $t_M \in$ T $\rightarrow$ t_M.subesquema $\Leftarrow$ *novoSubEsquema*
- Classificação da operação: desestabilizadora

Exemplo
O tipo de dados t_M, *tipoPessoa*, definido no esquema de exemplo, possui o subesquema S(P, O, T, *tipo*) em sua definição original, assim definido:

- P = {cargo, nome, ramal}

- O = {(ϵ, cargo), (ϵ, nome), (ϵ, ramal)}
- T = { }

A alteração do esquema de dados do tipo t_M, *tipoPessoa*, através da inclusão de um elemento *matricula*, causa a geração de um novo subesquema S'(P', O', T, *tipo*) para o tipo t_M, assim definido:

- P = {cargo, nome, ramal, matricula}
- O = {(ϵ, cargo), (ϵ, nome), (ϵ, ramal), (ϵ, matricula)}
- T = { }

4.4 Revalidação de documentos

A validação de documentos em relação a um esquema XML consiste em identificar se as restrições definidas no esquema são respeitadas por um documento. Já a revalidação parcial de documentos consiste na validação de documentos com uso de conhecimento sobre partes modificadas no esquema, o que restringe o processo a elementos correspondentes a porções modificadas no esquema. Com foco em porções dos artefatos XML, o processo é executado em tempo menor do que a revalidação completa de um documento.

A validação de documentos D em relação a um elemento p_v contido em um fragmento de esquema s_f é realizada pela função total *valida*, definida pelo Algoritmo 5:

Algoritmo 5 Função *valida*- identifica se um conjunto de documentos *D* é válido em relação a uma partícula p_v, contida no fragmento de esquema s_f

```
1:  valida ⇐ true
2:  for all d ∈ D do
3:    if ∃ p_v ∈ s_f.P ∧ valida = true then
4:      validaEstrutura = validaEstrutura(d, s_f, p_v)
5:      validaCardinalidade = validaCardinalidade(d, s_f, p_v)
6:      if p.atributos ≠ ∅ then
7:        validaCardinalidadeAtributo ⇐ true
8:        for all ∀ a ∈ p.atributos do
9:          validaCardinalidadeAtributo ⇐ validaCardinalidadeAtributo ∧ validaCar-
            dinalidadeAtributo(d, s_f, p_v, a)
10:       end for
11:     end if
12:     validaValores ⇐ validaValores(d, s_f, p_v)
13:     if validaEstrutura = false ∧ validaCardinalidade = false ∧ validaCardinalida-
        deAtributo = false ∧ validaValores = false then
          valida ⇐ false
14:     end if
15:   end if
16: end for
17: return valida
```

A função *valida* e os diferentes processos de revalidação de documentos XML, definidos no restante desta seção, fazem uso de diversas funções auxiliares. Uma delas é a

função *existeElemento*, que identifica se um conjunto de documentos D possui um nodo e que corresponda às definições de um elemento p, definido no fragmento de esquema s_f. Esta função é assim definida através de pseudo-código pelo Algoritmo 6:

Algoritmo 6 Função *existeElemento* - identifica se um elemento *p*, contido em um fragmento de esquema s_f, existe em um conjunto de documentos *D*

```
1: existeElemento ⇐ true
2: for all d ∈ D do
3:    if existeElemento = true∧ ( ∄ e ∈ d.E ∨ corresp(e, d, p, s_f) = false) then
4:       existeElemento ⇐ false
5:    end if
6: end for
7: return existeElemento
```

A função *existeAtributo* identifica se nodos correspondentes à partícula de esquema p possuem atributos de nome *nomeAtributo*. A associação entre a partícula p e um atributo de nome *nomeAtributo* existe no fragmento de esquema s_f, e deve ser mantida no conjunto de documentos D. Esta função é assim modelada:

Algoritmo 7 Função *existeAtributo* - identifica se um elemento *p*, contido em um fragmento de esquema s_f, possui um atributo de nome *nomeAtributo* em um conjunto de documentos *D*

```
1: existeAtributo ⇐ true
2: for all d ∈ D ∧ existeAtributo = true do
3:    existeAtributo ⇐ existeAtributo ∧ (∃ a ∈ d.E ∧ rótulo(a) = nomeAtributo ∧ tipo-
      Nodo(a) = atributo)
4:    existeAtributo ⇐ existeAtributo ∧ (∃ e ∈ d.E ∧ corresp(e, d, p, s_f) ∧ ∃ l ∈ d.L | l =
      (e, a))
5: end for
6: return existeAtributo
```

A função *validaCardinalidade* verifica se as cardinalidades mínimas e máximas de nodos pertencentes ao conjunto de documentos D e correspondentes à partícula p , definida no fragmento de esquema s_f, atendem às definições de cardinalidade da partícula p. Esta função é assim descrita através de pseudo-código no Algoritmo 8:

Algoritmo 8 Função *validaCardinalidade* - identifica se a cardinalidade de um elemento *p*, contido em um fragmento de esquema s_f, é correta no conjunto de documentos *D*

```
1: validaCardinalidade ⇐ true
2: for all d ∈ D ∧ validaCardinalidade = true do
3:    if ∃ C ⊆ d.L ∧ |C| ≥ p.cardinalidade.min ∧ |C| ≤ p.cardinalidade.max ∧ then
4:       for all (n_1, n_2) ∈ C ∧ validaCardinalidade = true do
            validaCardinalidade ⇐ validaCardinalidade ∧ corresp(n_2, d, p, s_f)
5:       end for
6:    end if
7: end for
8: return validaCardinalidade
```

Já a função *validaCardinalidadeAtributo* identifica se a cardinalidade de atributos em D, correspondentes a um atributo a, associado ao elemento p, atendem às restrições de cardinalidade definidas para o atributo a no fragmento de esquema s_f. Esta função é assim descrita pelo Algoritmo 9:

Algoritmo 9 Função *validaCardinalidadeAtributo* - identifica se a cardinalidade de um atributo *a*, contido pelo elemento *p* em um fragmento de esquema s_f, é correta no conjunto de documentos *D*

1: *validaCardinalidadeAtributo* $\Leftarrow$ **true**
2: **for all** d $\in$ D $\wedge$ *validaCardinalidadeAtributo* = **true do**
3: **if** $\exists$ C $\subseteq$ d.L $\wedge$ |C| $\geq$ a.cardinalidade.min $\wedge$ |C| $\leq$ a.cardinalidade.max $\wedge$ **then**
4: **for all** (n_1, n_2) $\in$ C $\wedge$ *validaCardinalidadeAtributo* = **true do**
validaCardinalidadeAtributo $\Leftarrow$ *validaCardinalidadeAtributo* $\wedge$ *corresp*(n_1, d, p, s_f) $\wedge$ *rótulo*(n_2) = a.nome
5: **end for**
6: **end if**
7: **end for**
8: **return** *validaCardinalidadeAtributo*

A função *validaValores* visa identificar se os valores de nodos em D, correspondentes a uma partícula de esquema p_s, atendem às restrições definidas para p_s no fragmento de esquema s_f. Estas restrições podem aplicar-se ao comprimento do valor, à sua formação ou ao tipo do valor: numérico, datas, etc. Esta função é assim modelada através de pseudo-código pelo Algoritmo 10:

Algoritmo 10 Função *validaValores* - identifica se nodos no conjunto de documentos *D*, correspondentes ao elemento *p*, definido em um fragmento de esquema s_f, possuem valor correto

1: *validaValores* $\Leftarrow$ **true**
2: **for all** d $\in$ D $\wedge$ *validaValores* = **true do**
3: **for all** l $\in$ d.L | l = (e_1, e_2) $\wedge$ *validaValores* = **true do**
validaValores $\Leftarrow$ *validaValores* $\wedge$ rótulo(e_1) = s_f.nome $\wedge$ *corresp*(e_2, d, p_s, s_f) $\wedge$ (valor(e_2) $\in$ L(p_s.domínio) $\vee$ valor(e_2) $\in$ L(p_s.tipo))
4: **end for**
5: **end for**
6: **return** *validaValores*

Por fim, a função *validaEstrutura* identifica se a subárvore formada pelos descendentes diretos de nodos correspondentes a uma partícula de esquema p atende às restrições de formação definidas para a partícula p no fragmento de esquema s_f. Esta função é definida pelo Algoritmo 11. A operação somatório, utilizada na função *validaEstrutura*, corresponde à concatenação dos rótulos dos elementos descendentes de um elemento p.

O restante desta seção analisa as ações de revalidação correspondentes às operações de modificação desestabilizadoras definidas na Seção 4.3. Cada ação de revalidação é analisada a partir do impacto que modificações em um fragmento de esquema s_f, pertencente a um esquema S e descendente de um elemento p, possui em um conjunto de documentos D. Assume-se que previamente à realização de modificações no esquema, o conjunto de documentos D já possuía referências ao esquema.

Algoritmo 11 Função *validaEstrutura* - identifica se a estrutura de nodos no conjunto de documentos *D* correspondente ao elemento *p*, definido em um fragmento de esquema s_f, é correta em relação à definição do esquema

1: *validaEstrutura* $\Leftarrow$ **true**
2: **for all** d $\in$ D $\wedge$ *validaEstrutura* = **true do**
3: **if** $\exists$ L' $\subseteq$ d.L | L' = {(e, e_i'), ..., (e, e_n')} **then**
4: **if** ($\sum_{i=0}^{|L'|-1}$ rótulo(e_i') | l' = (e, e_i')) $\notin$ L(p.subelementos) **then**
5: *validaEstrutura* $\Leftarrow$ **false**
6: **end if**
7: **end for**
8: **end for**
9: **return** *validaEstrutura*

4.4.1 Revalidação sobre a inclusão de elemento

A inclusão de um elemento p_N na raiz de um fragmento de esquema s_f demanda a validação completa do elemento correspondente a p_N no conjunto de documentos D. Tal validação é necessária somente quando o novo elemento possui cardinalidade obrigatória:

- Pré-condição: p_N.cardinalidade.min ≥ 1
- Regra de validação: valida(D, s_f, p_N) = verdadeiro

4.4.2 Revalidação sobre a alteração de nome de elemento

A alteração do nome de um elemento p_A, encontrado na raiz do fragmento de esquema s_f para *novoNome*, dando origem a um novo elemento p_A', demanda a validação dos nomes de elementos correspondentes no conjunto de documentos D. Tal validação é necessária em toda troca de nome de elemento:

- Regra de validação: existeElemento(D, s_f, p_A') = verdadeiro $\wedge$ existeElemento(D, s_f, p_A) = falso

4.4.3 Revalidação sobre a exclusão de elemento

A exclusão de um elemento p_E descendente de p demanda a verificação sobre o processamento desta exclusão também nos documentos XML. Os documentos não devem apresentar referências à raiz do subesquema s_f, esquema com raiz em p_E, tampouco aos demais elementos definidos na hierarquia do subesquema. Esta validação é assim especificada:

- Regra de validação: $\forall$ p $\in$ p_E.P, existeElemento(D, s_f, p) = falso

4.4.4 Revalidação sobre a cópia de elemento

A criação de um elemento p_D, j-ésimo descendente de p_{AD} a partir das propriedades de p_O, i-ésimo descendente de p implica na validação do elemento correspondente nos documentos contidos no conjunto D, caso p_D possua cardinalidade obrigatória. Tal validação é assim especificada:

- Pré-condição: p_D.cardinalidade.min ≥ 1
- Regra de validação: valida(D, s_f, p_D) = verdadeiro

4.4.5 Revalidação sobre a movimentação de elemento

A movimentação de um elemento p_O, i-ésimo descendente de p para a j-ésima posição dentre os descendentes de p_{AD} como p_D implica na validação da existência das estruturas correspondentes a p_D no conjunto de documentos D e na ausência de estruturas correspondentes a p_O no conjunto D. Tais validações são assim modeladas, onde s_D corresponde ao subesquema com raiz em p_{AD} e s_O corresponde ao subesquema com raiz em p_O:

- Regra de validação: existeElemento(D, s_D, p_{AD}) = verdadeiro $\wedge$ existeElemento(D, s_O, p_O) = falso

4.4.6 Revalidação sobre a alteração de tipo de partícula

A alteração do tipo referenciado por uma partícula p_A para *novoTipo* pode implicar em três processos distintos de revalidação. O primeiro processo é verificado quando o novo tipo referenciado por p_A encontra-se definido no conjunto T pertencente ao esquema S. Neste caso, é necessário validar se a estrutura hierárquica e os valores definidos pelas estruturas correspondentes a p_A no conjunto de documentos D atendem às restrições definidas por *novoTipo*. Tal validação é assim especificada:

- Pré-condição: L(*novoTipo*) $\not\supset$ L(p_A.tipo)
- Regra de validação: valida(D, s_f, p_A) = verdadeiro

Nos casos em que *novoTipo* é igual a ID, é necessário verificar se todos os nodos correspondentes a p_A em D possuem valores únicos no documento. Este processo de revalidação é assim especificado:

- Pré-condição: *novoTipo* = ID
- Regra de validação: $\forall$ d $\in$ D, $\forall$ n $\in$ d.E, $\nexists$ n' $\in$ d.E |

  corresp(n, d, s_f, p_A) = verdadeiro $\wedge$ *valor*(n) = *valor*(n')

Por fim, nos casos em que *novoTipo* é igual a IDRef ou IDRefs, é necessário verificar se os valores especificados por nodos correspondentes a p_A em D, são correspondentes a valores de outros nodos do documento. Tal revalidação é assim especificada:

- Pré-condição: *novoTipo* $\in$ (IDRef, IDRefs)
- Regra de validação: $\forall$ d $\in$ D, $\forall$ n $\in$ d.E, $\exists$ n' $\in$ d.E |

  corresp(n, d, s_f, p_A) = verdadeiro $\wedge$ *valor*(n') $\in$ *valor*(n)

4.4.7 Revalidação sobre a alteração de cardinalidades de partícula

A revalidação de um conjunto de documentos D, quando da alteração das cardinalidades de uma partícula p_C, consiste em identificar se as novas cardinalidades *minCardinalidade* e *maxCardinalidade* de p_C, caso a partícula em questão seja um elemento de esquema, ou a_C, caso a partícula seja um atributo descendente de p_C, são respeitadas nas estruturas correspondentes a p_C em D. Esta validação é assim especificada:

- Pré-condição: p_C.cardinalidade.min < *minCardinalidade* $\vee$ p_C.cardinalidade.max > *maxCardinalidade*

- Regras de validação:

  p_C.subelementos = $\epsilon \rightarrow$ validaCardinalidade(D, s_f, p_C) = verdadeiro

  p_C.subelementos $\neq \epsilon \rightarrow$ validaCardinalidadeAtributos(D, s_f, p_C, a_C) = verdadeiro

4.4.8 Revalidação sobre a alteração do domínio de partícula

A alteração do domínio de um elemento p_D para *novoDomínio* implica na verificação dos valores existentes nas estruturas correspondentes a p_D no conjunto de documentos representado por D. Esta validação é necessária apenas quando o novo conjunto de domínio é mais restrito do que o antigo domínio:

- Pré-condição: *novoDomínio* $\not\subset$ p_D.domínio
- Regra de validação: validaValores(D, s_f, p_D) = verdadeiro

4.4.9 Revalidação sobre a inclusão de atributo

A validação quando da inclusão de um atributo a_N no conjunto de atributos de um elemento p consiste em identificar se os atributos presentes em D atendem às especificações de a_N no esquema S. Tal validação é necessária apenas em situações onde a_N possui cardinalidade obrigatória:

- Pré-condição: a_N.cardinalidade.min ≥ 1
- Regra de validação: validaAtributo(D, s_f, p, a_N) = verdadeiro

4.4.10 Revalidação sobre a exclusão de atributo

A exclusão de um atributo a_E, descendente do elemento p implica em identificar se no conjunto de documentos D não há referências a um atributo a_E nas estruturas correspondentes a p em D. Esta validação é assim especificada:

- Regra de validação: existeAtributo(D, s_f, p, a_E.nome) = falso

4.4.11 Revalidação sobre a alteração de nome de atributo

A troca de nome de um atributo a_A, descendente de um elemento p, para *novoNome*, implica em identificar se o conjunto de documentos D possui referências ao atributo a_A através de *novoNome*, não havendo ocorrências do antigo nome de a_A. Tal validação é assim especificada:

- Regra de validação: existeAtributo(D, s_f, p, *novoNome*) = verdadeiro $\wedge$ existeAtributo(D, s_f, p, a_A.nome) = falso

4.4.12 Revalidação sobre a alteração na derivação de tipo

A alteração da derivação de um tipo t_M para *novaDerivação* implica em identificar se as estruturas em D, correspondentes às partículas que referenciam t_M, são válidas após a modificação de derivação. Tal validação é assim especificada:

- Regra de validação: $\forall$ p $\in$ P | $t_M \in$ p.tipo $\rightarrow$ valida(D, s_f, p) = verdadeiro

4.4.13 Revalidação sobre a alteração de tipo derivado

A alteração do tipo derivado por um tipo t_M em um esquema S implica em validar se todas estruturas em D correspondentes a partículas que referenciam o tipo t_M possuem conteúdo válido de acordo com o novo conjunto de tipos referenciado por t_M. Tal validação é assim especificada:

- Regra de validação: $\forall$ p $\in$ P | $t_M \in$ p.tipo $\rightarrow$ valida(D, s_f, p) = verdadeiro

4.4.14 Revalidação sobre a alteração de subesquema de tipo

A alteração do subesquema de um tipo t_M implica em validar se as estruturas correspondentes em D às partículas com referências a t_M em S são válidas, considerando-se o novo subesquema definido por t_M. Tal validação é assim especificada:

- Regra de validação: $\forall$ p $\in$ P | $t_M \in$ p.tipo $\rightarrow$ valida(D, s_f, p) = verdadeiro

4.5 Adaptação de documentos

Operações de adaptação em documentos XML são realizadas somente quando as operações de revalidação definidas na Seção 4.4 falham: nestes casos o documento encontra-se inválido em relação a um esquema. De posse do conhecimento de qual regra de revalidação está sendo violada por um documento, é possível realizar uma ação de adaptação específica.

Há de se notar que o processo de adaptação de documentos definido neste trabalho não aborda a questão da concorrência na atualização dos documentos. Assume-se que além do processo de adaptação de documentos em função de modificações nos esquemas, nenhum outro processo encontra-se realizando modificações nos documentos. Caso o processo de adaptação de documentos e algum outro processo realizem modificações concorrentes nos documentos XML, existe a possibilidade de que estes sejam levados a um estado inválido em relação ao esquema.

Também aplicações que utilizem esquemas XML podem necessitar de adaptação, uma vez que estruturas de dados da aplicação ou consultas baseadas em um esquema podem tornar-se inválidos em função das modificações. Usualmente estas modificações devem ser realizadas manualmente pelo desenvolvedor da aplicação, a partir da análise das modificações realizadas nos esquemas. É importante ressaltar que este trabalho não aborda a adaptação de aplicações em função de modificações nos esquemas: o foco do trabalho concentra-se na adaptação de documentos tornados inválidos quando da evolução de esquemas.

De uma maneira geral, pode-se dizer que todas as ações de adaptação possuem como pré-condição de execução a função valida(D, s_f, p_V) com valor igual a *falso*, onde s_f corresponde ao fragmento de esquema que contém o elemento modificado p_V. A partir de modificações em um esquema *s*, um conjunto de documentos *D* dá origem a um novo conjunto *D'* em função de adaptações realizadas nos documentos originais. As ações de adaptação são detalhadas a seguir.

4.5.1 Adaptação à inclusão de elemento

A adaptação à inclusão de um elemento p_N no esquema XML demanda a inclusão do novo nodo nos documentos, bem como a estrutura de partículas especificada pelo elemento, caso este possua cardinalidade obrigatória. Há de se notar que nos casos em

que o elemento possui cardinalidade obrigatória e referencia um tipo de dado de esquema, o processo de adaptação é finalizado somente quando o novo nodo é inserido e o valor de cada nova ocorrência do nodo nos documentos é especificado.

O presente trabalho não aborda o cenário de adaptação de inclusão de elementos obrigatórios e que referenciem tipos de dados de esquema. Nestes casos, a adaptação definida consiste na inclusão do nodo, sendo que a especificação do valor associado ao novo nodo fica a cargo do usuário da base de dados. Esta operação de adaptação utiliza como parâmetro um fragmento de esquema s_p, que corresponde à subárvore de s_f com raiz em p_N, sendo assim especificada:

- Documentos resultantes: $\forall$ d' $\in$ D', d'(E', L', *rótulo*, *valor*, *tipoNodo*), onde:
  - $\forall (p_1, p_2) \in s_p.O \mid p_2 \in s_p.P \wedge p_2.\text{cardinalidade.min} > 0$,
    $\sum_{i=0}^{p_2.cardinalidade.min}$ (E' = E $\cup$ e_N $\mid$
    rótulo(e_N) = p_2.nome $\wedge$
    (*tipoNodo*(e_N) = *elemento* $\rightarrow$ $p_2 \notin p_1$.atributos))
  - $\forall (p_1, p_2) \in s_p.O \mid p_2 \in s_p.P \wedge p_2.\text{cardinalidade.min} > 0$,
    $\sum_{i=0}^{p_2.cardinalidade.min}$ (L' = L $\cup$ (e_A, e_N) $\mid$
    rótulo(e_A) = p_1.nome $\wedge$ *tipoNodo*(e_A) = *elemento* $\wedge$
    rótulo(e_N) = p_2.nome $\wedge$ (*tipoNodo*(e_N) = *elemento* $\rightarrow$ $p_2 \notin p_1$.atributos)

Exemplo

O exemplo de adaptação de documento à inclusão de um novo elemento em um esquema considera o esquema resultante do exemplo apresentado na Seção 4.3.1. Assim sendo, o fragmento de esquema s_p(P, O, T, *tipo*) é assim definido:

- P = {id}
- O = {(departamento, id)}
- T = { }
- *tipoNodo*

O documento apresentado na Seção 4.3, adaptado a este novo esquema, é definido por d'(E', L', *rótulo*, *valor*, *tipoNodo*), onde:

- E' = {departamento, funcionarios, funcionario, cargo, funcionario.nome, ramal, localizacao, reponsavel, endereco, cep, nome} $\cup$ {id},
- L' = { (ϵ, departamento), (departamento, funcionarios), (funcionarios, funcionario), (funcionario, cargo), (funcionario, funcionario.nome), (funcionario, ramal), (departamento, localizacao), (localizacao, endereco), (localizacao, cep), (departamento, nome) } $\cup$ {(departamento, id)},
- *rótulo*,
- *valor*,
- *tipoNodo*

4.5.2 Adaptação à alteração de nome de elemento

A troca do nome de um elemento p_A, descendente de p em um esquema S, para *novoNome* demanda a alteração do nome dos nodos correspondentes a p_A no conjunto de documentos D. De acordo com a abstração de documentos XML adotada neste trabalho, somente o resultado da função *rótulo* é afetado na adaptação à troca de nome de nodos, sem alterações nos demais componentes da representação do documento:

- Documentos resultantes: $\forall$ d' $\in$ D', d'(E, L, *rótulo*, *valor*, *tipoNodo*), onde:
  - *rótulo* | $\forall$ e $\in$ E, *rótulo*(e) = *novoNome*

4.5.3 Adaptação à exclusão de elemento

A exclusão de um elemento p_E em um esquema implica na exclusão dos elementos correspondentes a toda hierarquia de p_E também nos documentos em D. A exclusão de um elemento torna as funções *rótulo*, *valor* e *tipoNodo* indefinidas para todos os nodos correspondentes aos elementos de p_E. A adaptação à exclusão de elementos é assim especificada:

- Documentos resultantes: $\forall$ d' $\in$ D', d'(E', L', *rótulo*, *valor*, *tipoNodo*), onde:
  - E' = E - e - E_E,
    corresp(e, d, p_E, s_f) = verdadeiro,
    E_E = $\cup$ (e_0.E $\cup$ ... $\cup$ e_n.E)
  - L' = L - (e_A, e) - $\cup$ (e_0.L $\cup$ ... $\cup$ e_n.L)

Exemplo
O exemplo de adaptação de documento à exclusão de um elemento em um esquema considera o esquema resultante do exemplo apresentado na Seção 4.3.3. Neste alteração, o elemento p_E que está sendo excluído é *localizacao*. Assim sendo, o fragmento de esquema E_e(P, O, T, *tipoNodo*), correspondente ao subesquema com raiz no elemento p_E, é assim definido:

- P = {localizacao, endereco, cep, responsavel}
- O = {(localizacao, endereco), (localizacao, cep), (localizacao, responsavel)}
- T = { }
- *tipoNodo*

O documento apresentado na Seção 4.3, adaptado a este novo esquema, é definido por d'(E', L', *rótulo*, *valor*, *tipoNodo*), onde:

- E' = {departamento, funcionarios, funcionario, cargo, funcionario.nome, ramal, localizacao, reponsavel, endereco, cep, nome} - {localizacao} - {responsavel, endereco, cep},
- L' = { (ϵ, departamento), (departamento, funcionarios), (funcionarios, funcionario), (funcionario, cargo), (funcionario, funcionario.nome), (funcionario, ramal), (departamento, localizacao), (localizacao, endereco), (localizacao, cep), (departamento, nome) } - {(departamento, localizacao)} - {(localizacao, endereco), (localizacao, cep)},

- *rótulo*,
- *valor*,
- *tipoNodo*

4.5.4 Adaptação à cópia de elemento

A adaptação à cópia de um elemento p_D, a partir das definições de um elemento p_O, implica na criação do nodo correspondente e_N, descendente de e, no conjunto de documentos D, bem como a estrutura de partículas de cardinalidade obrigatória especificada por este elemento. Assim como na adaptação à inclusão de elemento, verifica-se a ocorrência do problema relativo à especificação do valor do elemento copiado. Como a cópia de um elemento pode ser entendida como a criação de um novo elemento no esquema, a mesma solução é adotada nos dois cenários: o usuário final da base de dados fica responsável pela especificação dos valores de elementos copiados.

Assim como na adaptação à criação de elementos no esquema, a adaptação à cópia de elementos recebe como parâmetro um fragmento de esquema s_p, que corresponde à subárvore de s_f que contêm o elemento p_D. Esta operação de adaptação de documentos é assim especificada:

- Documentos resultantes: $\forall$ d' $\in$ D', d'(E', L', *rótulo*, *valor*, *tipoNodo*), onde:
  - $\forall$ (p_1, p_2) $\in$ s_p.O | $p_2 \in s_p$.P $\wedge$ p_2.cardinalidade.min > 0,
    $\sum_{i=0}^{p_2.cardinalidade.min}$ (E' = E $\cup$ e_N |
    rótulo(e_N) = p_2.nome $\wedge$
    (*tipoNodo*(e_N) = *elemento* $\rightarrow$ $p_2 \notin p_1$.atributos))
  - $\forall$ (p_1, p_2) $\in$ s_p.O | $p_2 \in s_p$.P $\wedge$ p_2.cardinalidade.min > 0,
    $\sum_{i=0}^{p_2.cardinalidade.min}$ (L' = L $\cup$ (e_A, e_N) |
    rótulo(e_A) = p_1.nome $\wedge$ *tipoNodo*(e_A) = *elemento* $\wedge$
    rótulo(e_N) = p_2.nome $\wedge$ (*tipoNodo*(e_N) = *elemento* $\rightarrow$ $p_2 \notin p_1$.atributos)

4.5.5 Adaptação à movimentação de elemento

A movimentação de um elemento de esquema, dando origem ao elemento p_D a partir de um elemento original p_O demanda a movimentação dos nodos e_D correspondentes a p_O em D. A movimentação realizada nos nodos e_D visa alterar o ancestral de e_D, a fim de que o novo ancestral seja o nodo correspondente ao elemento p_D. Esta operação de adaptação é assim especificada:

- Documentos resultantes: $\forall$ d' $\in$ D', d'(E, L', *rótulo*, *valor*, *tipoNodo*), onde:
  - L' = L - L_A $\cup$ L_N,
    $L_A \subset$ L | $\forall$ (e_N, e_D) $\in$ L_A, *corresp*(e_D, d, p_O, s_f) = verdadeiro
    $L_N = \sum_{i=0}^{|L_A|} \bigcup$ { (e_A, e_D) } | *corresp*(e_A, d, p_D, s_f) = verdadeiro

4.5.6 Adaptação à alteração de tipo de partícula

A alteração do tipo de dado referenciado por uma partícula p_A para *novoTipo* implica, como processo de adaptação, em alterar o valor ou a estrutura dos nodos correspondentes a p_A em D. Nos casos em que p_A, elemento raiz do fragmento de esquema s_f, passa a referenciar um tipo de dado de esquema mais restrito do que aquele originalmente referenciado, o valor dos nodos correspondentes, e quaisquer estruturas de nodos associadas ao tipo original de p_A, devem ser removidas de D. A mesma ação de adaptação aplica-se a tipos de dados de usuário que não possuem definição de subesquema. Há de se notar que todos os tipos de dado de esquema do formato XML Schema não possuem definição de subesquema: apenas os valores considerados válidos são restringidos por tipos de dado de esquema.

- Documentos resultantes: $\forall$ d' $\in$ D', d'(E', L', *rótulo*, *valor*, *tipoNodo*), onde:
  - *novoTipo*.subesquema = $\emptyset \wedge p_A$.tipo.subesquema = $\emptyset \rightarrow$
    $\forall$ e $\in$ d.E |
    corresp(e, d, p_A, s_f) = verdadeiro $\wedge$ *valor*(e) $\notin$ L(*novoTipo*), *valor*(e) = ϵ

Quando p_A referencia por extensão *novoTipo* e este é descendente do antigo tipo de p_A, a adaptação consiste em incluir em D os nodos correspondentes aos elementos definidos por *novoTipo*. No caso em que p_A referencia por restrição *novoTipo*, e este também referencia o antigo tipo de p_A, nenhum processo de adaptação deve ser executado, uma vez que a estrutura de nodos de D é compatível com *novoTipo*.

- Documentos resultantes: $\forall$ d' $\in$ D', d'(E', L', *rótulo*, *valor*, *tipoNodo*), onde:
  - *novoTipo*.subesquema $\neq \emptyset \wedge p_A$.tipo.subesquema $\neq \emptyset \wedge$ *novoTipo* $\supseteq p_A$.tipo
    $\rightarrow \forall (e_1, e_2) \in$ *novoTipo*.subesquema.L |
    e_2.cardinalidade.min > 0 $\wedge$
    $\nexists$ n $\in$ d.E, *corresp*(n, d, e_2, *novoTipo*.subesquema) = verdadeiro $\rightarrow$
    $\sum_{i=0}^{e_2.cardinalidade.min}$ d.E = d.E $\cup$ n_T | *rótulo*(n_T) = e_2.nome

Já nos casos em que p_A passa a referenciar um tipo de dado definido pelo usuário com subesquema, sendo que *novoTipo* não possui referências ao tipo original referenciado por p_A, a adaptação consiste em remover de D os nodos correspondentes ao antigo tipo de p_A e gerar uma nova estrutura de nodos correspondente a *novoTipo*.

- Documentos resultantes: $\forall$ d' $\in$ D', d'(E', L', *rótulo*, *valor*, *tipoNodo*), onde:
  - *novoTipo*.subesquema $\neq \emptyset \wedge p_A$.tipo.subesquema $\neq \emptyset \wedge$ L(*novoTipo*) $\cap$ L(p_A.tipo) = $\emptyset \rightarrow$
    $\forall$ e $\in$ d.E, $\exists$ p $\in p_A$.tipo.subesquema.P |
    corresp(e, d, p, p_A.tipo.subesquema) = verdadeiro $\rightarrow$
    d.E' = d.E - e
    d.L' = d.L - l' | $\forall$ k $\in$ l', k = (e_I, e)
    $\forall (e_1, e_2) \in$ *novoTipo*.subesquema.L |
    e_2.cardinalidade.min > 0 $\wedge$
    $\nexists$ n $\in$ d.E, corresp(n, d, e_2, *novoTipo*.subesquema) = verdadeiro $\rightarrow$
    $\sum_{i=0}^{e_2.cardinalidade.min}$ d.E = d.E $\cup$ n_T | *rótulo*(n_T) = e_2.nome

Nos casos de falha na revalidação em que *novoTipo* é igual a ID, o processo de adaptação consiste em remover os valores duplicados, a fim de garantir a unicidade dos identificadores. Tal processo de adaptação é assim especificado:

- Documentos resultantes: $\forall$ d' $\in$ D', d'(E, L, *rótulo*, *valor*, *tipoNodo*), onde:
  - $\exists$ C = (n_0, ..., n_i) $\subseteq$ d.E |
    |C| > 1 $\wedge$ $\forall$ n_i $\in$ C, *valor*(n_i) = v,
    valor(n_0) = v $\wedge$ *valor*(n_i) = ϵ | i > 0

Por fim, nos casos em que ocorre falha de revalidação em que *novoTipo* é igual a IDRef ou IDRefs, o processo de adaptação consiste em retirar o valor dos elementos que causaram a falha na revalidação por referenciarem identificadores inexistentes. Este adaptação é assim especificada:

- Documentos resultantes: $\forall$ d' $\in$ D', d'(E, L, *rótulo*, *valor*, *tipoNodo*), onde:
  - $\forall$ n $\in$ d.E, $\not\exists$ n' $\in$ d.E | *corresp*(n, d, p_A, s_f) = verdadeiro $\wedge$ *valor*(n') $\in$ *valor*(n), *valor*(n) = ϵ

4.5.7 Adaptação à alteração de cardinalidade de partícula

O incremento da cardinalidade mínima ou decremento da cardinalidade mínima de uma partícula p_C, descendente de p_A, implica adaptação dos documentos nos casos em que a cardinalidade mínima original é menor e nos casos em que a cardinalidade máxima é maior, respectivamente. Nos casos em que a cardinalidade mínima é aumentada, a adaptação consiste em incluir novos nodos correspondentes a p_C até que a nova cardinalidade mínima exigida seja atingida, sem que os valores referentes a estes nodos sejam especificados na adaptação: fica a cargo do usuário final da base de dados a especificação destes valores. Considerando-se que n_C corresponde ao conjunto de nodos correspondentes a p_C já existentes em d e que s_f corresponde ao fragmento de esquema com raiz em p_C, esta operação de adaptação é assim especificada:

- Documentos resultantes: $\forall$ d' $\in$ D', d'(E', L', *rótulo*, *valor*, *tipoNodo*), onde:
  - $\sum_{i=|n_C|}^{p_C.cardinalidade.min}$
    $\forall$ (p_1, p_2) $\in$ p_C.O $\wedge$ p_2.cardinalidade.min > 0,
    E' = E $\cup$ e_N | *corresp*(e_N, d, p_2, s_f) = verdadeiro $\rightarrow$ $p_2 \notin p_1$.atributos)
  - $\sum_{i=|n_C|}^{p_C.cardinalidade.min}$
    $\forall$ (p_1, p_2) $\in$ p_C.O $\wedge$ p_2.cardinalidade.min > 0,
    L' = L $\cup$ (e_A, e_N) |
    corresp(e_A, d, p_1, s_f) = verdadeiro $\wedge$
    corresp(e_N, d, p_2, s_f) = verdadeiro $\rightarrow$ $p_2 \notin p_1$.atributos)

Já no caso em que a cardinalidade máxima é diminuída, os documentos que violam esta nova restrição terão nodos correspondentes a p_C retirados dos documentos. Considerando-se a ordenação existente entre elementos de um mesmo nível em um documento XML, sempre os últimos nodos correspondentes a p_C serão retirados, até que a nova cardinalidade máxima seja atingida. Esta operação de adaptação é assim especificada:

- Documentos resultantes: $\forall$ d' $\in$ D', d'(E', L', *rótulo*, *valor*, *tipoNodo*), onde:
  - E' = E - n_D, (p_{Ai}.E $\cup$ ... $\cup$ p_{An}.E) $\in$ n_D | i > p_C.cardinalidade.max
  - L' = L - l_D, (p_{Ai}.L $\cup$ ... $\cup$ p_{An}.L) $\in$ n_D | i > p_C.cardinalidade.max

4.5.8 Adaptação à alteração de domínio de partícula

A falha na validação de nodos em D correspondentes a p_D, um elemento de esquema cujo domínio foi modificado, implica em retirar do documento XML o valor dos nodos. Esta adaptação é assim especificada:

- Documentos resultantes: $\forall$ d' $\in$ D', d'(E, L, *rótulo*, *valor*, *tipoNodo*), onde:
  - *valor* = *valor* | $\forall$ (n_1, n_2) $\in$ d.L, *valor*(n_2) = ϵ
  - *rótulo* = *rótulo* | $\forall$ (n_1, n_2) $\in$ d.L, *rótulo*(n_2).nome = p_D.nome

4.5.9 Adaptação à inclusão de atributo

A inclusão de um atributo a_N em um elemento p de um esquema demanda a inclusão de atributos correspondentes no conjunto de documentos D, caso a cardinalidade mínima de a_N seja igual a 1. Esta operação de adaptação é assim especificada:

- Documentos resultantes: $\forall$ d' $\in$ D', d'(E', L', *rótulo*, *valor*, *tipoNodo*), onde:
  - E' = E $\cup$ n_A | *rótulo*(n_A) = a_N.nome $\wedge$ *tipoNodo*(n_A) = *atributo*
  - L' = L $\cup$ (n_D, n_A) | *corresp*(n_D, d, p, s_f) = verdadeiro

Exemplo
O exemplo de adaptação de documento à inclusão de um novo atributo em um esquema considera o esquema resultante do exemplo apresentado na Seção 4.3.9. Os documentos d'(E', L', *rótulo*, *valor*, *tipoNodo*), resultantes da adaptação ocasionada pela modificação do esquema são assim definidos:

- E' = {departamento, funcionarios, funcionario, cargo, funcionario.nome, ramal, localizacao, reponsavel, endereco, cep, nome} $\cup$ {id},
  - *tipoNodo*(id) = atributo
- L' = { (ϵ, departamento), (departamento, funcionarios), (funcionarios, funcionario), (funcionario, cargo), (funcionario, funcionario.nome), (funcionario, ramal), (departamento, localizacao), (localizacao, endereco), (localizacao, cep), (departamento, nome) } $\cup$ {(departamento, id)},
- *rótulo*,
- *valor*,
- *tipoNodo*

4.5.10 Adaptação à exclusão de atributo

A adaptação de documentos XML quando da exclusão de um atributo a_E, descendente de um elemento p em um esquema XML, consiste na exclusão dos atributos correspondentes a a_E em D. Este processo de adaptação de documentos é assim especificado:

- Documentos resultantes: $\forall$ d' $\in$ D', d'(E', L', *rótulo*, *valor*, *tipoNodo*), onde:
  - E' = E - n_E | *rótulo*(n_E) = a_E.nome $\wedge$ *tipoNodo*(n_E) = *atributo*
  - L' = L - (n_A, n_E) | *corresp*(n_A, d, p, s_f) = verdadeiro

  na raiz da hierarquia

4.5.11 Adaptação à alteração de nome de atributo

A troca do nome de um atributo a_A, descendente de p em um esquema XML, para *novoNome*, demanda a alteração do nome dos atributos correspondentes a a_A em D para *novoNome*. Esta operação de adaptação é assim especificada, conforme a abstração de documentos XML adotada neste trabalho:

- Documentos resultantes: $\forall$ d' $\in$ D', d'(E, L, *rótulo*, *valor*, *tipoNodo*), onde:
  - *rótulo*= *rótulo*, $\forall$ a $\in$ E | *rótulo*(a) = *novoNome*

4.5.12 Adaptação à alteração na derivação de tipo

Quando da alteração na derivação de um tipo referenciado por nodos em D através de partículas p_D em um esquema S, o processo de adaptação dos nodos encontra-se ligado ao novo tipo de derivação *novaDerivação*. Quando a nova derivação de um tipo for *lista*, o conjunto de nodos n_D, formado por nodos considerados inválidos em função da troca de derivação, terá sua hierarquia de descendentes excluída, bem como valores associados a esta hierarquia:

- Documentos resultantes: $\forall$ d' $\in$ D', d'(E', L', *rótulo*, *valor*, *tipoNodo*), onde:
  - E' = E - (n_{D0}.E $\cup$ ... $\cup$ n_{Dn}.E) | n < |n_D|
  - L' = L - (n_{D0}.L $\cup$ ... $\cup$ n_{Dn}.L) | n < |n_D|

Nos casos em que *novaDerivação* for igual a *restrição*, a ação de adaptação consiste em retirar nodos não correspondentes à definição do tipo t_M:

- Documentos resultantes: $\forall$ d' $\in$ D', d'(E', L', *rótulo*, *valor*, *tipoNodo*), onde:
  - E' = E - (n_{D0}.E $\cup$ ... $\cup$ n_{Dn}.E) |
    $\forall$ n_{Di} $\in$ (n_{D0}.E $\cup$ ... $\cup$ n_{Dn}.E), $\nexists$ p $\in$ t_M.subesquema.P |
    corresp(n_{Di}, d, p, s_f) = verdadeiro
  - L' = L - (n_{D0}.L $\cup$ ... $\cup$ n_{Dn}.L) |
    $\forall$ n_{Di} $\in$ (n_{D0}.L $\cup$ ... $\cup$ n_{Dn}.L), $\nexists$ (p_1, p_2) $\in$ t_M.subesquema.L |
    corresp(n_{Di}, d, p_2, s_f) = verdadeiro

Por fim, nos casos em que *novaDerivação* for igual a *extensão*, a adaptação de nodos inválidos consiste em remover nodos n_D não correspondentes à definição de t_M e incluir no documento XML os nodos faltantes como descendentes de n_A, nodo correspondente a t_M:

- Documentos resultantes: $\forall$ d' $\in$ D', d'(E', L', *rótulo*, *valor*, *tipoNodo*), onde:
  - E' = E - (n_{D0}.E $\cup$ ... $\cup$ n_{Dn}.E) $\cup$ n_I |
    $\forall$ $n_{Di} \in$ (n_{D0}.E $\cup$ ... $\cup$ n_{Dn}.E), $\nexists$ p $\in$ t_M.subesquema.P |
    corresp(n_{Di}, d, p, s_f) = verdadeiro
    $\forall$ n $\in$ n_I, $\nexists$ n' $\in$ E, $\nexists$ p $\in$ p_D.tipo.subesquema.P |
    rótulo(n') = *rótulo*(n) $\wedge$ *corresp*(n', d, p, p_D.tipo.subesquema) = verdadeiro
  - L' = L - (n_{D0}.L $\cup$ ... $\cup$ n_{Dn}.L) $\cup$ n_L |
    $\forall$ $n_{Di} \in$ (n_{D0}.L $\cup$ ... $\cup$ n_{Dn}.L), $\nexists$ (p_1, p_2) $\in$ t_M.subesquema.L |
    corresp(n_{Di}, d, p_2, s_f) = verdadeiro
    $\forall$ (n_1, n_2) $\in$ n_L, $\exists$ (l_1, l_2) $\in$ t_M.subesquema.L |
    corresp(n_1, d, l_1, s_f) = verdadeiro $\wedge$ *corresp*(n_2, d, l_2, s_f) = verdadeiro

4.5.13 Adaptação à alteração de tipo derivado

Quando da alteração do tipo derivado por um tipo t_M e o conjunto t_M.tipo referencia apenas um tipo de dado, a adaptação do conjunto de documentos D consiste em remover nodos sem correspondência em t_M.tipo, e incluir novos nodos, previamente não existentes em D. Nos casos em que o conjunto t_M.tipo possui referência a mais de um tipo de dado, a adaptação dos documentos XML fica a cargo do usuário final da base de dados, que escolherá o tipo de dados adequado a modelar os nodos de D, dentre aqueles contidos no conjunto t_M.tipo. Esta ação de adaptação é assim especificada:

- Pré-condição: |t_M.tipo| = 1
- Documentos resultantes: $\forall$ d' $\in$ D', d'(E', L', *rótulo*, *valor*, *tipoNodo*), onde:
  - E' = E - (n_{D0}.E $\cup$ ... $\cup$ n_{Dn}.E) $\cup$ n_I |
    $\forall$ $n_{Di} \in$ (n_{D0}.E $\cup$ ... $\cup$ n_{Dn}.E), $\nexists$ p $\in$ t_M.subesquema.P |
    corresp(n_{Di}, d, p, t_M.tipo.subesquema) = verdadeiro
    $\forall$ n $\in$ n_I, $\nexists$ n' $\in$ E, $\nexists$ p $\in$ t_M.subesquema.P |
    rótulo(n') = *rótulo*(n) $\wedge$ *corresp*(n, d, p, t_M) = verdadeiro
  - L' = L - (n_{D0}.L $\cup$ ... $\cup$ n_{Dn}.L) $\cup$ n_L |
    $\forall$ $n_{Di} \in$ (n_{D0}.L $\cup$ ... $\cup$ n_{Dn}.L), $\nexists$ (p_1, p_2) $\in$ t_M.subesquema.L |
    corresp(n_{Di}, d, p_2, t_M.subesquema)
    $\forall$ (n_1, n_2) $\in$ n_L, $\exists$ (l_1, l_2) $\in$ t_M.subesquema.L |
    corresp(n_1, d, l_1, t_M.subesquema) $\wedge$ *corresp*(n_2, d, l_2, t_M.subesquema)

4.5.14 Adaptação à alteração de subesquema de tipo

A ação de adaptação correspondente à modificação do subesquema de um tipo de dado t_M corresponde à inclusão de nodos não existentes e remoção de nodos correspondentes a elementos retirados do tipo t_M. Este processo de adaptação é assim especificado:

- Documentos resultantes: $\forall$ d' $\in$ D', d'(E', L', *rótulo*, *valor*, *tipoNodo*), onde:
  - E' = E - (n_{D0}.E $\cup$ ... $\cup$ n_{Dn}.E) $\cup$ n_I |
    $\forall$ $n_{Di} \in$ (n_{D0}.E $\cup$ ... $\cup$ n_{Dn}.E), $\nexists$ p $\in$ t_M.subesquema.P |
    corresp(n_{Di}, d, p, t_M.subesquema)
    $\forall$ n $\in$ n_I, $\nexists$ n' $\in$ E, $\exists$ p $\in$ t_M.subesquema.P |
    rótulo(n') = *rótulo*(n) $\wedge$ *corresp*(n', d, p, t_M.subesquema) = verdadeiro
  - L' = L - (n_{D0}.L $\cup$ ... $\cup$ n_{Dn}.L) $\cup$ n_L |
    $\forall$ $n_{Di} \in$ (n_{D0}.L $\cup$ ... $\cup$ n_{Dn}.L), $\nexists$ (p_1, p_2) $\in$ t_M.subesquema.L |
    corresp(n_{Di}, d, p_2, t_M.subesquema) = verdadeiro
    $\forall$ (n_1, n_2) $\in$ n_L, $\exists$ (l_1, l_2) $\in$ t_M.subesquema.L |
    corresp(n_1, d, l_1, t_M.subesquema) = verdadeiro $\wedge$
    corresp(n_2, d, l_2, t_M.subesquema) = verdadeiro

Exemplo
O exemplo de adaptação de documento à alteração do subesquema de um tipo de dados considera o esquema S'(P, O, T, *tipoNodo*), assim definido:

- P = {departamento, funcionarios, funcionario, dadosPessoais, matricula, inicial, sobrenome, ramal, localização, endereço, cep, responsável, nome}
- O = {(ϵ, departamento), (departamento, funcionarios), (funcionarios, funcionario), (funcionario, dadosPessoais), (dadosPessoais, matricula), (),dadosPessoais, inicial), (dadosPessoais, sobrenome), (dadosPessoais, ramal), (departamento, localização), (localização, endereço), (localização, cep), (localização, responsável), (departamento, nome)}
- T = {tipoPessoa}

O documento apresentado na Seção 4.3, adaptado ao novo esquema do tipo *tipoPessoa*, é definido por d'(E', L', *rótulo*, *valor*, *tipoNodo*), onde:

- E' = {departamento, funcionarios, funcionario, cargo, funcionario.nome, ramal, localizacao, reponsavel, endereco, cep, nome} - {cargo, funcionario.nome, ramal} $\cup$ {dadosPessoais, matricula, inicial, sobrenome, ramal},
- L' = { (ϵ, departamento), (departamento, funcionarios), (funcionarios, funcionario), (funcionario, cargo), (funcionario, funcionario.nome), (funcionario, ramal), (departamento, localizacao), (localizacao, endereco), (localizacao, cep), (departamento, nome) } - {(funcionario, cargo), (funcionario, funcionario.nome), (funcionario, ramal)} $\cup$ {(funcionario, dadosPessoais), (dadosPessoais, matricula), (dadosPessoais, inicial), (dadosPessoais, sobrenome), (dadosPessoais, ramal)},
- *rótulo*,
- *valor*,
- *tipoNodo*

4.6 Considerações finais

Este capítulo apresentou o X-Spread, um mecanismo que aborda a propagação do processo de evolução de esquemas XML para documentos que referenciam estes esquemas. Em função da ausência de um padrão para tratamento do impacto que a evolução de esquemas possui sobre documentos XML, este mecanismo visa abordar os efeitos do processo de evolução de esquemas XML. Processo este que ocorre freqüentemente durante o processo de desenvolvimento de aplicações, seja em função de correções na base de dados ou alterações em requisitos técnicos ou de negócio.

O processo de evolução de esquemas XML é modelado pelo X-Spread como uma série de operações de modificação sobre estes esquemas. As operações de modificação modeladas permitem a construção de uma ampla gama de esquemas. As características do formato XML Schema não abordadas por estas operações dizem respeito a grupos de substituição e a definição do XML Schema de unicidade de elementos e referências a estes elementos. A identificação de modificações realizadas em um esquema pode iniciar um processo de revalidação dos documentos XML, a fim de identificar se estes ainda encontram-se em conformidade em relação ao esquema.

Os documentos XML que não encontram-se em conformidade com o esquema modificado passam por um processo de adaptação, onde busca-se colocá-los novamente em estado válido em relação ao esquema. O processo de adaptação do X-Spread possui uma limitação, na medida em que a especificação de valores de elementos de documentos não pode ser realizada pelo mecanismo. Tal tarefa fica a cargo do usuário final da base de dados, uma vez que este possui conhecimento sobre a realidade modelada pela base de dados semiestruturada. A adaptação realizada pelo usuário da base de dados ocorre com o uso de uma aplicação específica para edição de documentos XML ou através das aplicações que criaram tal artefato.

Além de afetar documentos que constituem a base de dados, a evolução de esquemas possui diferentes impactos nas aplicações que utilizam tais esquemas. Aplicações podem tornar-se inconsistentes por construírem estruturas de dados ou cláusulas de consulta a partir das definições de um esquema. Considerando-se que os esquemas XML, a exemplo de esquemas relacionais, são artefatos externos às aplicações, somente a intervenção de um desenvolvedor pode devolver a aplicação a um estado consistente. Embora a correção de aplicações em função de modificações em esquemas seja um problema a ser tratato, este trabalho aborda somente o impacto que a evolução de esquemas XML possui sobre as instâncias componentes da base de dados semiestruturados.

O próximo capítulo especifica a arquitetura para implementação do X-Spread e apresenta o protótipo implementado.

5 ARQUITETURA E IMPLEMENTAÇÃO DO X-SPREAD

Este capítulo especifica a arquitetura necessária para implementação do X-Spread e apresenta as características do protótipo implementado de acordo com as definições de arquitetura apresentadas. Por fim, a avaliação do mecanismo proposto é apresentada. Nesta avaliação, o protótipo do X-Spread é comparado a métodos já existentes para revalidação de documentos XML.

O módulo de Detecção de Diferenças, apresentado na Seção 5.1.1, é implementado pela ferramenta XSDelta. Esta ferramenta é especificada em (PERINI; SILVEIRA; GALANTE, 2006), artigo publicado na III Sessão de Demos do Simpósio Brasileiro de Banco de Dados.

5.1 Arquitetura

A arquitetura do X-Spread especifica as definições do mecanismo apresentadas no Capítulo 4. O mecanismo atua como um sistema observador, sendo a sua primeira tarefa a identificação das modificações realizadas em um determinado esquema XML. O conjunto de esquemas monitorados pelo X-Spread é configurado pelo administrador da base de dados. A partir da identificação de alterações em um esquema, estas alterações são analisadas em detalhes, a fim de distinguir qual a real natureza da modificação, como quais porções do esquema foram modificadas, quais os novos valores devem ser especificados. A arquitetura geral do X-Spread e o fluxo das informações entre os componentes é ilustrada na Figura 5.1.

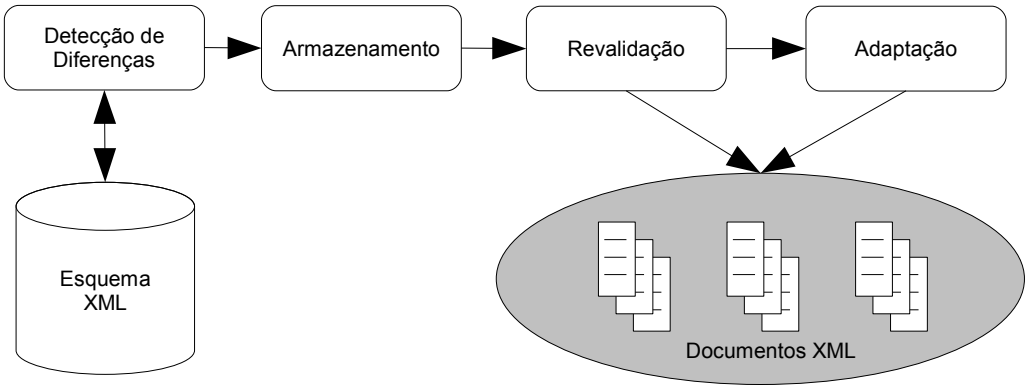

Figura 5.1: Arquitetura geral do X-Spread

A partir da identificação das modificações realizadas no esquema XML, estas são armazenadas em uma base de dados própria do mecanismo. O armazenamento destas

informações visa a construção de um histórico sobre o processo de evolução de um esquema XML. Este histórico permite também a adaptação de diferentes documentos XML, gerados a partir de diferentes versões de um mesmo esquema XML.

Uma vez que as modificações realizadas em um esquema encontram-se armazenadas na base de dados própria do X-Spread, o processo de revalidação de documentos XML que referenciam o esquema modificado é iniciado. Caso algum documento XML tenha falhado no processo de revalidação, e de posse de quais são estes documentos, o X-Spread inicia a adaptação destes documentos. O objetivo do processo de adaptação é tornar o documento XML novamente compatível com o esquema XML, considerando-se as modificações realizadas no esquema.

Esta arquitetura aplica-se a documentos armazenados em sistemas de arquivos e a mensagens XMLs trocadas entre diferentes aplicações através de uma rede de dados. A abordagem do X-Spread a estes dois tipos de documentos XML exige a definição de novas capacidades para cada componente do mecanismo, entretanto, o fluxo de ações do X-Spread é similar para os dois tipos de documentos XML. As próximas seções analisam em detalhes os componentes do mecanismo.

5.1.1 *Módulo de Detecção de Diferenças*

A Figura 5.2 ilustra o módulo de detecção de diferenças que monitora a realização de modificações em esquemas XML e identifica quais são estas mudanças. O componente Observador realiza a monitoração de esquemas através de consultas a esquemas executadas a intervalos definidos pelo administrador da base de dados. Quando da primeira consulta a um esquema, caso este não se encontre armazenado na base de dados do X-Spread, ele será armazenado junto com o valor resultante da aplicação de funções de dispersão ao esquema, como o MD5[1].

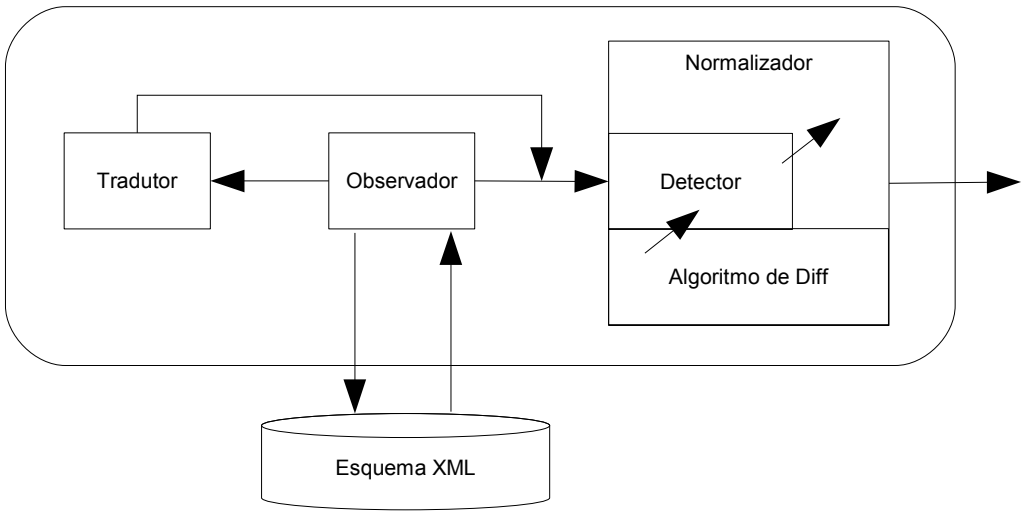

Figura 5.2: Arquitetura do módulo Detecção de Diferenças

Consultas subseqüentes recuperam o conteúdo completo do esquema XML e novamente aplicam a função de dispersão ao esquema. Caso o novo valor calculado seja diferente daquele armazenado para o esquema na base de dados do X-Spread, a nova versão

[1]http://tools.ietf.org/html/rfc1321

do esquema é armazenada na base de dados. Realizado o armazenamento da versão do esquema, o processo de detecção de diferenças entre as versões do esquema é iniciado.

A comparação entre as duas versões do esquema é executada pelo componente Detector, através da aplicação de algoritmos de detecção de diferenças entre documentos XML. Assim sendo, a partir da aplicação do algoritmo de detecção de diferenças nas duas versões do esquema XML, o X-Spread realiza a normalização das saídas do algoritmo através do componente Normalizador. Estas saídas são processadas e armazenadas na base de dados do X-Spread em formato próprio, de maneira que não haja acoplamento entre o mecanismo como um todo e um determinado algoritmo. Atualmente o X-Spread emprega o algoritmo XyDiff. Entretanto, por armazenar as saídas do algoritmo em um formato próprio, o X-Spread pode ser adaptado para utilizar um novo algoritmo mais eficiente no futuro.

Naturalmente, o suporte a um novo algoritmo de detecção de diferenças exige novas implementações no X-Spread, na medida em que este precisará interpretar o conjunto de possíveis resultados do novo algoritmo. Este é o único esforço envolvido na adaptação do X-Spread a um novo algoritmo de detecção de diferenças.

O componente Normalizador possui como objetivo identificar, a partir do resultado da execução de um algoritmo de detecção de diferenças em documentos XML, qual partícula foi afetada pela modificação no esquema e qual o ancestral desta partícula. Estes valores são necessários nos casos em que ocorre a inclusão ou exclusão de um elemento, por exemplo. Nos casos em que ocorre a cópia ou movimentação de uma partícula, é necessário identificar o antigo e o novo ancestral da partícula em questão. Caso o valor de algum atributo definido no esquema tenha sido modificado, o novo valor também deve ser determinado.

Com a identificação dos itens afetados pela alteração do esquema, como partículas modificadas e seus ancestrais, valores alterados em atributos e elementos, o componente Normalizador mapeia estes itens para as operações de modificação especificadas pelo X-Spread. O esforço envolvido neste processo de mapeamento está diretamente relacionado com a qualidade das saídas geradas pelo algoritmo de detecção de diferenças e com o número de operações suportadas por estes algoritmos.

O X-Spread é construído visando abordar as características do formato XML Schema, um dos formatos de esquema XML mais poderosos e difundidos na atualidade. Entretanto, as definições do formato DTD, embora menos poderosas do que o formato XML Schema, também são bastante utilizadas em aplicações comerciais e projetos científicos. A fim de abordar também o processo de evolução de esquemas em formato DTD, o componente Tradutor realiza a conversão de esquemas em formato DTD para formato XML Schema. A partir do esquema resultante, o restante do mecanismo X-Spread tem seu funcionamento inalterado. O X-Spread realiza a conversão de esquemas DTD para XML Schema através da utilização do pacote Castor[2]. Da mesma maneira, futuramente o componente Tradutor pode suportar a tradução de novos formatos de esquemas XML para XML Schema através do emprego de diferentes algoritmos ou pacotes de tradução.

Monitorando um conjunto de esquemas XML e analisando versões distintas de um esquema, o módulo de detecção de diferenças disponibiliza ao módulo de armazenamento um conjunto de diferenças existentes entre versões de um mesmo esquema. Estas diferenças encontram-se no formato próprio do X-Spread, e são produzidas pelas seguintes ações, executadas periodicamente a fim de identificar modificações em esquemas:

[2]http://castor.codehaus.org/

1. Componente Observador consulta esquema
2. Componente Tradutor analisa esquema
   (a) Se esquema não encontra-se em formato XML Schema, o componente Tradutor realiza a tradução
3. Componente Observador calcula o valor da função da dispersão para o esquema
4. Se valor da função de dispersão é diferente daquele armazenado na base de dados X-Spread
   (a) Componente Detector submete a nova versão do esquema e a última versão do esquema armazenada na base de dados do X-Spread ao algoritmo de detecção de diferenças
   (b) Componente Normalizador processa as saídas do componente Detector e submete as suas saídas ao módulo de armazenamento

5.1.2 *Módulo de Armazenamento*

Os objetivos do módulo de armazenamento são armazenar configurações do mecanismo, versões e metadados de esquemas e disponibilizar *interfaces* de consulta e armazenamento destes dados aos outros módulos do X-Spread. Os componentes que integram a arquitetura interna do módulo são ilustrados na Figura 5.3.

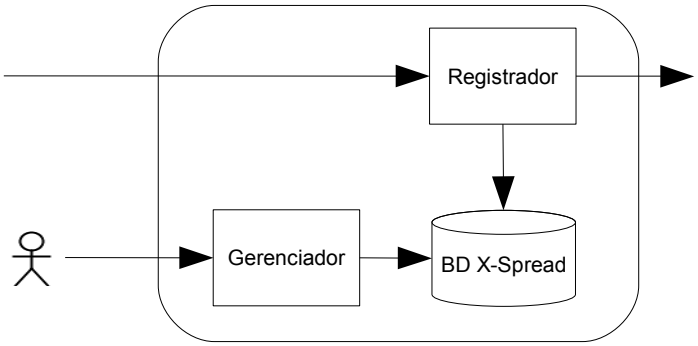

Figura 5.3: Arquitetura interna do módulo de armazenamento

O componente Gerenciador disponibiliza uma *interface* ao administrador da base de dados, a fim de que este especifique quais são os esquemas XML cuja evolução deve ser monitorada. O administrador da base de dados também poderá definir o conjunto de documentos que devem ser revalidados pelo mecanismo, as portas de comunicação a monitorar em um servidor, o intervalo de tempo entre consultas a esquemas e as operações de adaptação ignoradas pelo mecanismo. A configuração de operações de adaptação ignoradas pelo mecanismo modela cenários em que a exclusão de elementos ou valores de documentos não deve ser realizada.

Ao administrador da base de dados também é disponibilizada uma *interface* de consulta ao registro de atividades do mecanismo, a fim de que este possa verificar quais eventos ocorreram em bases de dados monitoradas pelo X-Spread. O registro dos eventos de modificação de esquemas, revalidação e adaptação de documentos visa disponibilizar

informações de auditoria, permitindo ao administrador da base de dados rastrear as ações do mecanismo durante sua execução. Esta mesma *interface* permite a ativação da consulta a um esquema monitorado pelo X-Spread, o que pode desencadear a execução do restante do mecanismo de revalidação e adaptação de documentos a partir da identificação de uma nova versão do esquema.

O modelo de dados utilizado pelos diferentes componentes do módulo de armazenamento é ilustrado na Figura 5.4. As estruturas *Esquema*, *Documento*, *OperaçõesHabilitadas*, *Servidor*, *PortasComunicação* e *Configuração* são manipuladas pelo componente Gerenciador, enquanto as restantes são manipuladas pelo componente Registrador.

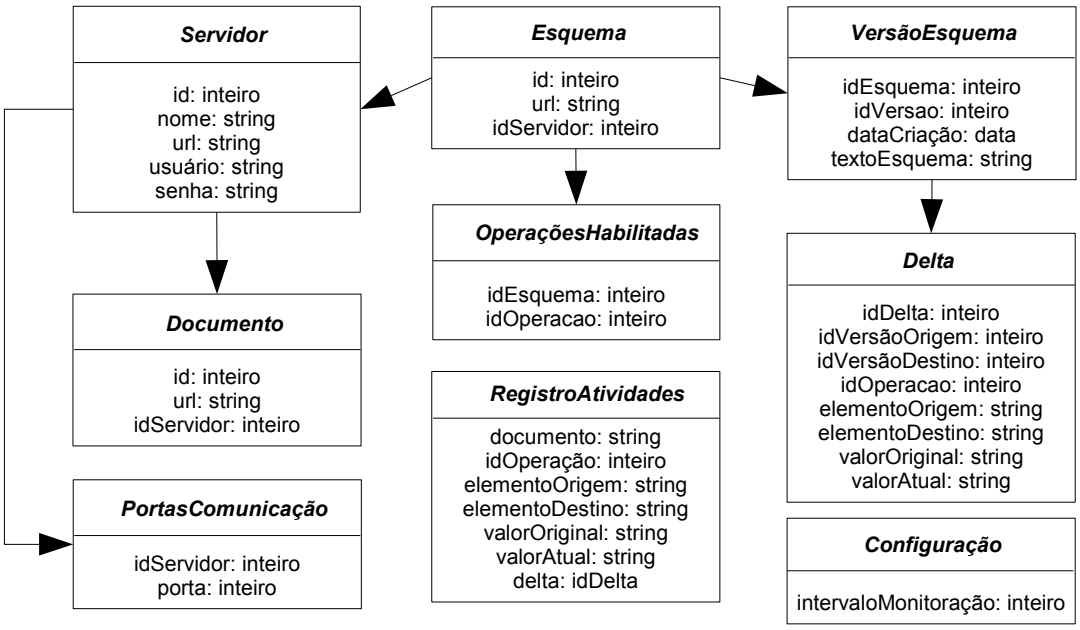

Figura 5.4: Modelo de dados utilizado pelo módulo de armazenamento

A estrutura *Esquema* armazena a identificação de um esquema XML, enquanto *Documento* registra documentos que devem ser revalidados e adaptados. *OperaçõesHabilitadas* armazena as ações de revalidação e adaptação habilitadas para um esquema, enquanto a estrutura *Servidor* define informações de acesso a servidores de arquivos. A estrutura *PortasComunicação* armazena portas de comunicação de servidores monitoradas pelo X-Spread, enquanto *Configuração* armazena parâmetros de operação do X-Spread.

Quando da alteração de informações armazenadas pelas estruturas controladas pelo componente Gerenciador, uma mensagem contendo as configurações modificadas é enviada aos demais módulos do X-Spread. Esta mensagem é emitida a fim de que os demais módulos utilizem em seus *caches* internos as configurações atualizadas do mecanismo. A emissão desta mensagem pelo módulo de armazenamento evita que os demais módulos realizem consultas periódicas pela configuração atual do mecanismo.

A estrutura de dados *RegistroAtividades* é compartilhada pelos diferentes componentes do módulo de armazenamento, pois é atualizada pelo componente Registrador e consultada pelo componente Gerenciador. A consulta do componente Gerenciador ocorre quando o administrador da base de dados pesquisa as informações de auditoria do X-Spread. As atualizações de dados nesta estrutura são desencadeadas pelos módulos de revalidação e adaptação, que comunicam ao módulo de armazenamento informações so-

bre documentos cuja revalidação falhou e necessitam ser submetidos a adaptação.

O módulo de armazenamento também atualiza a estrutura *RegistroAtividades* quando da identificação de diferenças entre duas versões de um esquema. As versões de um esquema são armazenadas na estrutura *VersãoEsquema*, enquanto as diferenças encontradas entre versões de um esquema são registradas na estrutura *Delta*. A atualização destas estruturas ocorre a partir de informações ao módulo de armazenamento pelo módulo de detecção de diferenças.

O componente Registrador disponibiliza aos demais módulos do X-Spread, *interfaces* para armazenamento e consulta a versões de esquemas e diferenças entre versões de esquemas. Quando do registro de uma nova versão de esquema, o módulo de armazenamento é invocado pelo módulo de detecção de diferenças, recebendo como parâmetros a identificação da versão original do esquema, a nova versão deste, e as diferenças encontradas, já em formato próprio do X-Spread. Neste formato próprio, gerado pelo componente Normalizador, encontra-se a identificação dos elementos modificados no esquema, seus ancestrais e possíveis novos valores de elementos e atributos.

Tão logo as informações da nova versão de esquema estejam armazenadas no esquema de dados do módulo de armazenamento, este registra assincronamente em *RegistroAtividades* as modificações identificadas entre duas versões de um esquema e invoca o módulo de revalidação, dando continuidade ao fluxo de execução do X-Spread. O registro em *RegistroAtividades* visa disponibilizar ao administrador da base de dados, dentre as informações de auditoria, a identificação de modificações realizadas em esquemas.

Na invocação do módulo de revalidação, a versão atual do esquema e aquela imediatamente anterior são fornecidas como parâmetros, bem como as operações de modificação identificadas entre estas versões. Operações de modificação consideradas neutras não são fornecidas como parâmetro ao módulo de revalidação, uma vez que elas não introduzem inconsistências na base de dados semiestruturados.

5.1.3 *Módulo de Revalidação*

O módulo de revalidação possui um único objetivo, revalidar um documento em relação a um conjunto de modificações realizadas em um esquema XML. Este objetivo é aplicado a diferentes cenários, na medida em que este documento pode encontrar-se armazenado em um servidor ou pode ser transmitido através de uma rede de dados entre duas aplicações. A arquitetura interna do módulo de revalidação é ilustrada na Figura 5.5.

Ao ser invocado pelo módulo de armazenamento, o módulo de revalidação deve atuar sobre documentos armazenados em um servidor. Neste caso, o componente Revalidador recebe uma requisição com a identificação das duas últimas versões do esquema modificado, bem como o conjunto de modificações desestabilizadoras realizadas no esquema. O componente Revalidador consulta o seu *cache* interno, a fim de identificar o conjunto de documentos relacionados ao esquema em questão que deve ser revalidado.

De posse do conjunto de documentos sobre os quais deve atuar, o componente verifica se para cada um destes documentos, as ações de modificação do esquema introduziram alguma inconsistência na base de dados. As diferentes ações de revalidação habilitadas pelo administrador da base de dados e armazenadas no *cache* do módulo, são realizadas através de consultas à estrutura dos documentos, conforme definido na Seção 4.4. Documentos cujo acesso encontra-se indisponível são adicionados a uma fila interna do componente Revalidador, juntamente com as ações de revalidação a executar neste documento. Periodicamente esta fila será consultada, a fim de verificar a disponibilidade de acesso aos documentos nela presentes. Caso os documentos encontrem-se acessíveis, a

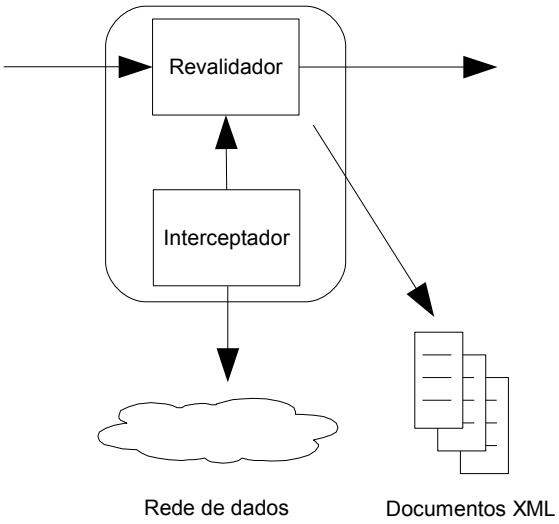

Figura 5.5: Arquitetura interna do módulo de revalidação

sua revalidação será retomada e o documento será retirado desta fila.

O componente Revalidador avalia serialmente um conjunto de documentos, processando cada documento em relação a cada modificação e identificando a necessidade de adaptação de um documento antes que outro seja avaliado. O processamento serial diminui o tempo de trabalho do mecanismo em cada documento, o que implica em menor possibilidade de acesso concorrente de outras aplicações a documentos processados pelo X-Spread. Conforme definido na Seção 4.5, este trabalho não define a gerência do acesso concorrente a documentos XML. O X-Spread evita acessos concorrentes a documentos pelo próprio mecanismo, executando revalidações em função de novas versões de esquema somente quando revalidações anteriores encontram-se finalizadas. A revalidação de documentos não requer acesso exclusivo a estes artefatos, de maneira que a revalidação ocorre enquanto outras aplicações que utilizam estes artefatos encontram-se em funcionamento.

As operações de modificação correspondentes a ações de revalidação executadas com sucesso não são enviadas ao módulo de adaptação. Desta maneira, o módulo de adaptação recebe como parâmetro a identificação de um documento e um conjunto de modificações que introduziram inconsistências no documento em relação ao esquema. Quando o módulo de adaptação é invocado, o componente Revalidador envia uma mensagem de auditoria ao módulo de armazenamento, a fim de registrar quais ações de revalidação falharam para um documento. Esta mensagem é processada assíncronamente, a fim de permitir o prosseguimento do fluxo de execução do mecanismo independentemente de seu registro.

Quando da revalidação de um documento em relação a uma versão de esquema, ao documento é adicionada uma marca de revalidação. Esta marca consiste na identificação da versão do esquema para a qual o documento foi revalidado. Nos casos em que o documento já apresenta uma marca de revalidação, esta será substituída, a fim de refletir a revalidação em relação à versão mais recente do esquema. Com o intuito de evitar a alteração do conteúdo do documento com a introdução de estruturas ou valores não pertinentes, a marca de revalidação é inserida como um comentário no documento.

A introdução da marca de revalidação nos documentos XML oferece informações que permitem a revalidação de documentos em diferentes cenários. Quando a versão identificada no documento não corresponde à última versão do esquema, é necessário realizar

a revalidação do documento. Em determinados cenários, a marca de revalidação do documento pode referenciar versões de esquema que não sejam a versão atual do esquema ou sua antecessora imediata. Nestes casos, o documento deve ser submetido a processos sucessivos de revalidação. Nestas revalidações, o documento será avaliado *n* vezes, onde *n* corresponde ao número de versões de esquema existentes entre a versão identificada pelo documento e a versão atual do esquema.

Quando o documento armazenado em um servidor não possui uma marca de revalidação, a revalidação será realizada considerando as modificações que deram origem ao esquema atual. Ainda que a revalidação em relação a estas modificações não falhe, existe a possibilidade de que o documento não seja válido em relação ao esquema. Esta possibilidade justifica-se por eventuais inconsistências em função de modificações realizadas em versões intermediárias do esquema. Estas versões intermediárias não foram consideradas em revalidações do documento, dada a ausência de marcas de revalidação no documento, logo, a possibilidade de incompatibilidade com o esquema existe.

A fim de abordar o cenário onde o documento é submetido sem falhas a sua primeira revalidação parcial, terminado este processo, o documento é revalidado completamente em relação ao esquema. Esta revalidação identificará se o documento é totalmente compatível com o esquema. Caso a revalidação completa falhe, o documento será revalidado em relação às versões antecessoras da versão atual do esquema. Estas revalidações ocorrerão até a identificação de uma versão do esquema com a qual o documento é compatível. A partir da identificação desta versão, o documento será submetido a processos de revalidação parcial e às adaptações correspondentes. Com isso, o documento se tornará compatível com a versão atual do esquema.

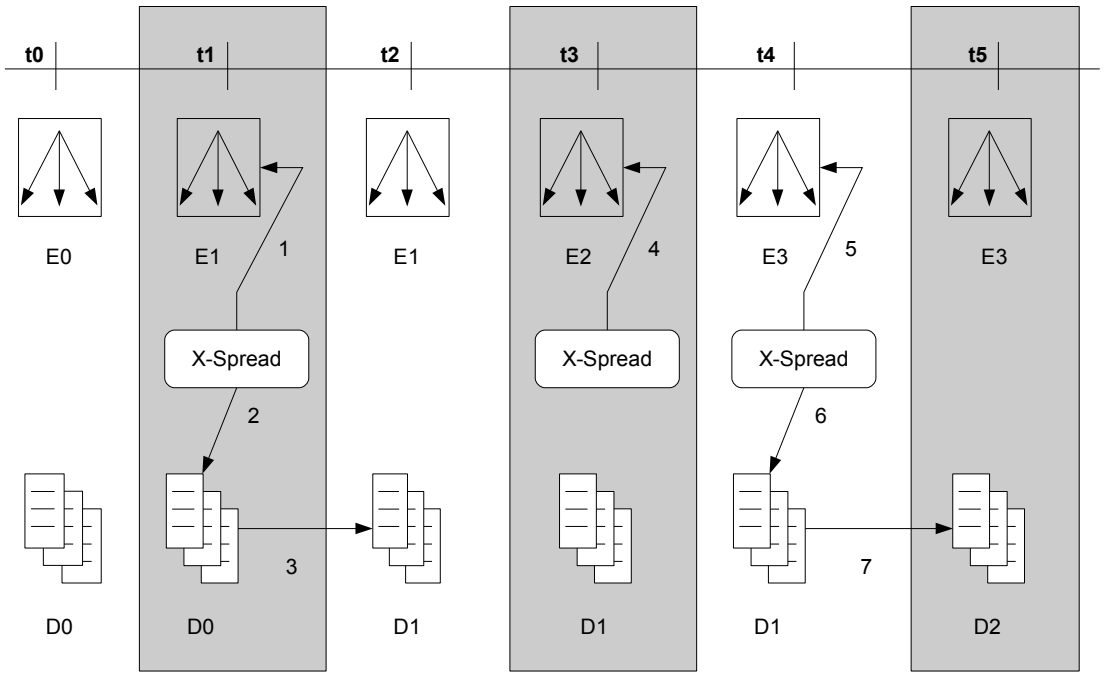

Figura 5.6: Fluxo de revalidação e adaptação de documentos a partir da evolução de esquemas XML

A Figura 5.6 ilustra o fluxo de evolução de um esquema e os processos de revalidação executados nos documentos que referenciam este esquema. Esta figura ilustra seis momentos de tempo distintos na evolução de um esquema, assim caracterizados:

- $\mathbf{t}_0$ - esquema E_0 é monitorado pelo X-Spread. A este esquema encontra-se associado o conjunto de documentos D_0. A totalidade de documentos deste conjunto é compatível com o esquema E_0;

- $\mathbf{t}_1$ - esquema E_1 é gerado. Este esquema é uma nova versão de E_0, correspondendo a uma versão modificada de E_0. O X-Spread executa diferentes ações em função da geração desta nova versão do esquema:

  - **1** - as diferenças entre E_0 e E_1 são detectadas;
  - **2** - o conjunto de documentos D_0 é revalidado em relação às diferenças existentes entre E_0 e E_1;
  - **3** - o conjunto de documentos D_1 é originado a partir das adaptações realizadas em D_0.

- $\mathbf{t}_2$ - a adaptação dos documentos em D_1 é finalizada. Este conjunto de documentos é compatível com a versão de esquema E_1;

- $\mathbf{t}_3$ - a versão de esquema E_2 é gerada a partir de E_1. A ação 4 representa a comparação realizada pelo X-Spread entre as versões de esquema E_1 e E_2. Nenhuma adaptação no conjunto de documentos D_1 é necessária em função da criação desta nova versão de esquema;

- $\mathbf{t}_4$ - a versão de esquema E_3 é gerada. O X-Spread executa uma série de ações devido à criação desta nova versão do esquema:

  - **5** - as diferenças entre as versões de esquema E_2 e E_3 são detectadas;
  - **6** - o conjunto de documentos D_1 é revalidado em relação à versão de esquema E_3;
  - **7** - o conjunto de documentos D_2 é originado a partir de adaptações realizadas em D_1.

- $\mathbf{t}_5$ - a totalidade dos documentos contidos em D_2 é compatível com E_3.

Cenários alternativos àquele ilustrado na Figura 5.6 são possíveis. Esta possibilidade consiste, por exemplo, na inclusão em D_1 de um documento criado a partir da versão E_2 do esquema. Neste caso, o documento poderia ser validado com sucesso em relação às diferenças existentes entre E_2 e E_3 e falhar na revalidação completa em relação a E_3. Assim, o documento teria de ser revalidado em relação às versões antecessoras à versão atual do esquema, a fim de identificar aquela com a qual o documento é compatível. A partir da identificação desta versão, o documento seria adaptado em relação às diferenças existentes entre todas as versões de esquema que levaram à criação de E_3.

Outro cenário alternativo consiste na ausência de registro para a versão de esquema E_2 na base de dados do X-Spread. Uma versão esquema pode não ser registrada na base de dados do X-Spread, pois ela existiu durante um intervalo de tempo inferior ao intervalo de consultas emitidas pelo X-Spread ao esquema. Ou ainda, esta versão de esquema pode não ter sido disponibilizada para consulta do X-Spread, entretanto, foi utilizada para geração de documentos. Nestes casos, o componente Revalidador pode não encontrar uma versão de esquema compatível com um documento. Logo, estes documentos não passarão pelo processo ao qual o restante dos documentos são submetidos, onde são adaptados em

relação às versões intermediárias do esquema, até se tornarem novamente compatíveis com a versão atual do esquema.

Quando o documento XML é transmitido através de uma rede de dados, o módulo de revalidação é responsável por sua interceptação e revalidação. O componente Interceptador atua como um *sniffer* de rede, sendo responsável por monitorar portas de comunicação de um servidor, capturando documentos com referências a esquemas cuja evolução é monitorada pelo X-Spread. A fim de diferenciar documentos armazenados em servidores de arquivos e documentos interceptados em redes de dados, os últimos são designados como mensagens XML.

Usualmente, mensagens XML não apresentarão nenhuma marca de revalidação. Nestes casos, assim como ocorre em documentos armazenados em sistemas de arquivos, o processo de revalidação utiliza como parâmetros as operações de modificação que originaram a última versão do esquema.

O fluxo de ações aos quais os documentos são submetidos, onde a compatibilidade destes em relação ao esquema é verificada quando do término de sua primeira revalidação parcial, não aplica-se a mensagens. Isto deve-se à restrição imposta pelas aplicações que geraram e que consumirão esta mensagem nos tempos de transmissão da mensagem. A mensagem deve ser enviada através da rede e processada por sua aplicação destino dentro de um limite de tempo. Caso este limite seja violado, a aplicação destino da mensagem pode erroneamente assumir que a mensagem não foi transmitida corretamente e iniciar um tratamento de erro, quando na verdade a mensagem está sendo processada pelo X-Spread.

Assim, mensagens são interceptadas na rede e submetidas a revalidação executada pelo componente Revalidador. Se necessário, são adaptadas, como ocorre com documentos. Entretanto, não há inclusão de marcas de revalidação nem verificação de compatibilidade entre a mensagem e o esquema. No envio de mensagens XML ao módulo de adaptação, ao invés de enviar a identificação do documento e um conjunto de modificações, como ocorre com documentos XML, o componente Interceptador envia o conteúdo da mensagem juntamente com as modificações. Desta maneira, o módulo de adaptação identificará que deve retornar o conteúdo do documento adaptado ao componente Interceptador, a fim de que este encaminhe o documento a seu destino na rede de dados.

Após o envio da mensagem já revalidada e adaptada pela rede de dados, o componente Interceptador efetua uma revalidação completa da mensagem. Caso esta revalidação falhe, o componente passa a revalidar a mensagem em relação às versões antecessoras da versão atual do esquema. O objetivo deste processo consiste em identificar a versão mais recente do esquema com a qual a mensagem é compatível. A combinação de versão de esquema e nodo de rede que originou a mensagem é armazenada em um *cache* interno do componente. Futuras revalidações de mensagens originadas deste nodo de rede considerarão a versão de esquema em *cache*, e não a versão atual do esquema.

5.1.4 *Módulo de Adaptação*

O módulo de adaptação possui como objetivo devolver documentos inválidos a um estado de compatibilidade em relação a um esquema. O módulo de adaptação recebe como parâmetros a identificação de um documento ou o conteúdo de uma mensagem XML e um conjunto de modificações aplicadas ao esquema, cujas revalidações correspondentes falharam. As estruturas do documento são manipuladas a partir do conjunto de modificações, conforme definições apresentadas na Seção 4.5.

A diferenciação existente para o módulo de adaptação entre documentos e mensagens XML consiste no fato de que mensagens adaptadas são retornadas ao componente Inter-

ceptador do módulo de revalidação, que as envia pela rede de dados. Quando efetua a manipulação de documentos, o módulo de adaptação é responsável por gravar estes em um servidor de arquivos.

Ao iniciar o processo de adaptação, seja de documentos ou mensagens, o módulo de adaptação consulta seu *cache* interno a fim de identificar quais, dentre as modificações realizadas no esquema, gerarão ações de adaptação nos documentos. Este *cache* é populado com as configurações do mecanismo, especificadas pelo administrador do banco de dados.

Assim como o módulo de revalidação, o módulo de adaptação possui uma fila interna que identifica documentos e ações de adaptação. Documentos nesta fila encontravam-se inacessíveis quando o módulo tentou realizar seu processamento. Esta fila é composta apenas por documentos, sem presença de mensagens XML. Mensagens não encontram-se nesta fila, pois elas são enviadas ao módulo de adaptação somente quando recebidas pelo módulo de revalidação, ou seja, elas encontram-se disponíveis para manipulação. Periodicamente esta fila é consultada, a fim de identificar a disponibilidade de acesso dos documentos, com o intuito de finalizar a adaptação destes.

Quando da finalização da adaptação de um documento, uma mensagem de auditoria é enviada ao módulo de armazenamento, identificando quais adaptações foram executadas no documento. Uma mensagem similar é enviada ao módulo de armazenamento quando mensagens XML são adaptadas. Entretanto, a mensagem de auditoria referente à adaptação de mensagens XML deve conter em sua representação o novo conteúdo e também o conteúdo original da mensagem XML. Assim, permite-se ao administrador do banco de dados verificar quais exatamente foram as modificações executadas na mensagem XML, dada a caracterização deste artefato, que não é armazenado para referências futuras.

Os documentos resultantes do processo de adaptação refletem modificações realizadas no esquema, consideradas desestabilizadoras e habilitadas a gerar ações de adaptação pelo administrador do banco de dados. Os documentos adaptados podem ser compatíveis com o esquema ou podem exigir a especificação de valores para estruturas definidas no documento durante o processo de adaptação. Esta exigência decorre, por exemplo, da inclusão de elementos de cardinalidade obrigatória no esquema. Este novo elemento do esquema deve possuir correspondente no documento, e deve possuir um valor. Este valor não é definido pelo módulo de adaptação, ficando esta tarefa a cargo do usuário final da base de dados. O usuário fica encarregado desta tarefa, realizada através de aplicações que manipulam documentos XML, pois este possui conhecimento sobre as regras definidas pelo domínio modelado pela base de dados.

5.2 Protótipo

O protótipo do X-Spread é construído com base na especificação da arquitetura do mecanismo, conforme definições apresentadas na Seção 5.1. Com base no Sun Java 5 (SUN MICROSYSTEMS, 2004), e as seguintes bibliotecas e aplicativos foram utilizados na construção deste protótipo:

- jaxen (CODEHAUS, 2004) - biblioteca com suporte a pesquisas XPath em Java, utilizada pelos módulos de revalidação e adaptação;

- MySQL (AB, 2007) - banco de dados interno do X-Spread, acessado pelo módulo de armazenamento.

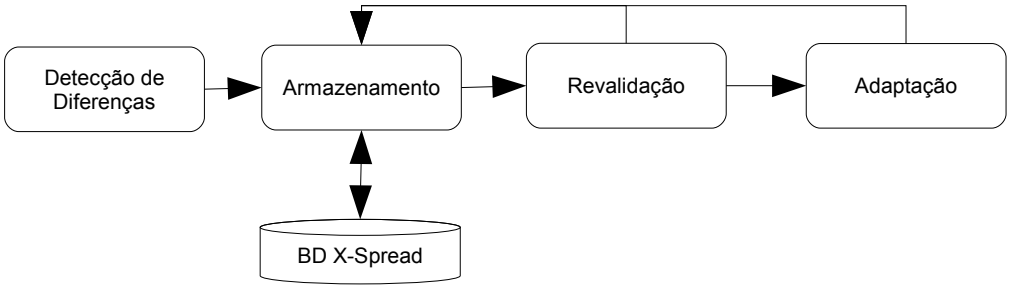

Figura 5.7: Arquitetura do protótipo do X-Spread e chamadas entre os componentes

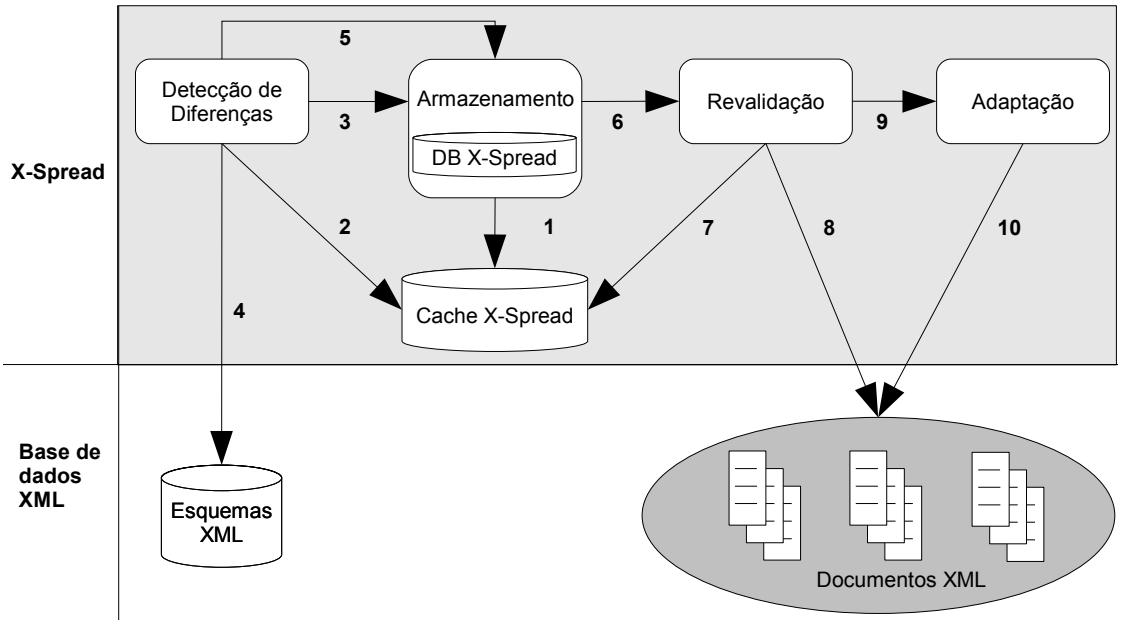

Figura 5.8: Arquitetura do protótipo do X-Spread, invocações entre componentes e artefatos externos

A Figura 5.7 apresenta os principais componentes do protótipo do X-Spread. Esta figura também apresenta os fluxos de comunicação existentes entre estes componentes e o banco de dados do X-Spread.

O X-Spread consulta a intervalos regulares, através de execuções periódicas de uma *thread* específica, um conjunto de esquemas e documentos configurados pelo administrador do banco de dados. Esta consulta também é realizada em dois diferentes cenários: quando a configuração de esquemas e documentos monitorados pelo X-Spread é alterada e quando a consulta a esquemas é iniciada manualmente pelo administrador do banco de dados. Quando a configuração de esquemas e documentos monitorados pelo X-Spread é modificada, seu *cache* é atualizado com as configurações mais recentes.

5.2.1 Processamento da evolução de esquema

A Figura 5.8 ilustra a estrutura do X-Spread e o fluxo de ações executado quando da revalidação de um conjunto de documentos. Por razões de clareza, o banco de dados do X-Spread é encapsulado pelo módulo de armazenamento. As ações executadas pelo X-Spread, conforme numeração na figura, são:

1. o módulo de armazenamento atualiza o *cache* interno do X-Spread, com a identi-

ficação de esquemas e documentos monitorados, conforme especificação do administrador do banco de dados;

2. o módulo de detecção de diferenças consulta o *cache* do X-Spread e identifica quais esquemas devem ser monitorados;

3. o módulo de detecção de diferenças consulta o módulo de armazenamento, em busca das versões mais recentes dos esquemas presentes no *cache* do X-Spread;

4. o módulo de detecção de diferenças acessa a versão mais recente dos esquemas e as compara com as versões armazenadas na base de dados do X-Spread;

5. o módulo de detecção de diferenças envia ao módulo de armazenamento os esquemas que possuem uma nova versão disponível, bem como as modificações realizadas, para registro na base de dados do X-Spread;

6. o módulo de armazenamento submete modificações desestabilizadoras realizadas em um esquema ao módulo de revalidação;

7. o módulo de revalidação consulta o *cache* do X-Spread e identifica quais documentos devem ser revalidados;

8. o módulo de revalidação consulta documentos XML especificados no *cache* do X-Spread e os processa;

9. o módulo de revalidação submente documentos inválidos ao processamento do módulo de adaptação;

10. o módulo de adaptação grava documentos adaptados nos seus respectivos sistemas de arquivos, disponibilizando a versão adaptada destes artefatos ao restante dos usuários da base de dados.

O passo 1 refere-se à preparação do ambiente de execução do X-Spread, com atualização de seu *cache*. As demais ações, compreendidas pelos passos 2 a 4 são executados periodicamente: caso nenhuma versão de esquema tenha sido criada, nenhuma ação adicional será executada pelo X-Spread. Caso uma nova versão de esquema tenha sido publicada, o passo 5 é executado. Caso alguma modificação desestabilizadora tenha sido realizada no esquema, os passos 6 a 8 são executados. Por fim, caso algum documento necessite de adaptação, os passos 9 e 10 são executados, a fim de devolver a base de dados a um estado íntegro.

O X-Spread possui fluxos alternativos de execução, que aplicam-se à revalidação e adaptação de mensagens XML. Nestes casos, a mensagem XML é interceptada na rede de dados e os passos 8 e 9 do processo de revalidação e adaptação de documentos são aplicados às mensagens interceptadas. A revalidação de mensagens XML considera as modificações desestabilizadoras que deram origem à versão atual do esquema.

5.3 Experimentos

Esta seção apresenta os experimentos realizados com o X-Spred com o objetivo de avaliar o desempenho deste em relação a pacotes de revalidação de documentos XML existentes. Os experimentos simulam seqüências de revalidações de diferentes arquivos

XML associados a um mesmo esquema. Através da realização de cinco experimentos, busca-se identificar os cenários onde o X-Spread possui o melhor desempenho, bem como as características destes arquivos que influenciam o desempenho do X-Spread. O X-Spread ainda é submetido a um teste de desempenho do seu processo de adaptação de documentos inválidos. Os demais pacotes de revalidação de documentos XML avaliados possuem as seguintes características:

- DOM (W3C, 2005) - Definido pela W3C, o DOM (*Document Object Model*) consiste em um modelo de objetos para representação de documentos semiestruturados. Neste modelo, o documento deve ser carregado em memória antes que algum processamento seja efetuado. Os testes realizados consideram a implementação padrão da versão 5 do Sun Java para o DOM;

- SAX (PROJECT, 2004) - SAX (*Simple API for XML*) é uma alternativa ao modelo DOM. Resultado do trabalho de um conjunto de desenvolvedores e usuários, SAX difere do DOM na medida em que um documento não precisa estar carregado totalmente na memória para que algum processamento seja iniciado. Isto se deve ao fato de que SAX opera com eventos durante a análise dos documentos: o processamento do início de um elemento é um evento, assim como o processamento do término de um elemento é um novo evento. Os testes realizados com SAX utilizam a implementação da versão 5 do Sun Java;

- API Java 5 (HAROLD, 2006) - A API para validação de documentos XML do Java 5 suporta a validação de documentos em relação a diferentes formatos de esquema. Dentre os formatos suportados, encontram-se o DTD e o XML Schema.

5.3.1 Parâmetros utilizados

A avaliação de desempenho considera na análise do resultado final as variações de tamanho e número de nodos dos diferentes documentos XML submetidos ao processamento destes pacotes. O pacote utilizado como parâmetro na avaliação de desempenho é o SAX. Conforme detalhado no restante desta seção, este pacote apresenta o melhor desempenho na maioria dos casos de testes, quando considerados apenas pacotes de revalidação já existentes na literatura.

Os pacotes foram submetidos a diferentes testes e em cada um destes testes os documentos XML apresentavam variações em sua estrutura, de maneira a gerar diferentes cenários para avaliação do desempenho dos pacotes de revalidação. Estes testes são assim definidos:

1. revalidação de um conjunto de documentos compatíveis com um esquema, após a alteração de cardinalidade de um elemento no esquema, de opcional para obrigatório;

2. revalidação de um conjunto de documentos incompatíveis com um esquema, em função da inclusão de um elemento de cardinalidade obrigatória no esquema;

3. revalidação de um conjunto de documentos incompatíveis com um esquema, em função da exclusão de um elemento do esquema;

4. revalidação de um conjunto de documentos incompatíveis com um esquema, em função da inclusão de um novo atributo de cardinalidade obrigatória no esquema;

5. revalidação de um conjunto de documentos incompatíveis com um esquema, em função da combinação das modificações de esquema descritas nos itens 2, 3 e 4;

6. adaptação do conjunto de documentos utilizados nos testes 2, 3, 4 e 5.

Cada conjunto de documentos utilizado nos testes 1 a 5 é composto por cinco documentos que apresentam variações de tamanho e número de nodos. O esquema utilizado nos testes possui 183 nodos, com tamanho físico de 4,76 Kbytes. Este esquema é uma versão adaptada do esquema utilizado para validação de artigos da enciclopédia *online* Wikipedia (http://en.wikipedia.org). A adaptação realizada no esquema original consiste na alteração de referências a esquemas disponíveis na *web*: nos testes, tais referências consideram esquemas disponíveis localmente. Esta modificação visa desconsiderar atrasos de rede referentes a acessos a esquemas *online* durante o teste dos diferentes pacotes. À exceção do X-Spread, que acessa o esquema e o armazena em sua base de dados interna para uso durante o processo de revalidação, os demais pacotes necessitam emitir uma consulta a estes esquemas para cada documento XML revalidado. A Tabela 5.1 detalha o número de nodos e tamanho físico dos documentos utilizados nos testes. Estes documentos correspondem a diferentes artigos disponíveis na Wikipedia.

Tabela 5.1: Características dos arquivos utilizados no teste de desempenho dos algoritmos de revalidação de documentos XML

Experimento / Arquivo	Nodos	Tamanho físico (em KBytes)
1 / 1	37	14.1
1 / 2	117	8.6
1 / 3	322	71.2
1 / 4	532	191.6
1 / 5	822	562.2
2 / 1	101	11.4
2 / 2	133	47.3
2 / 3	197	80
2 / 4	395	206.7
2 / 5	806	491.8
3 / 1	90	17.9
3 / 2	130	32.9
3 / 3	177	52.6
3 / 4	208	68.6
3 / 5	231	81.2
4 / 1	488	508
4 / 2	595	686.6
4 / 3	657	791.9
4 / 4	702	884.6
4 / 5	797	1024
5 / 1	38	4
5 / 2	38	13.5
5 / 3	38	48.7
5 / 4	38	107
5 / 5	38	189.9

Cada pacote avaliado foi executado um total de 31875 vezes, em diferentes combinações de ordem de execução. Essas diferentes combinações visam evitar a introdução de discrepâncias no resultado dos testes em função de atrasos de *hardware*, por exemplo. Cada pacote de revalidação foi aplicado 1275 vezes a cada um dos 25 documentos pertencente ao conjunto de testes 1 a 5 a fim de identificar a média do tempo de execução do processo de revalidação. Todos os pacotes avaliados possuem uma fase de configuração, que foi desconsiderada do cômputo do resultado final, pois é realizada apenas no

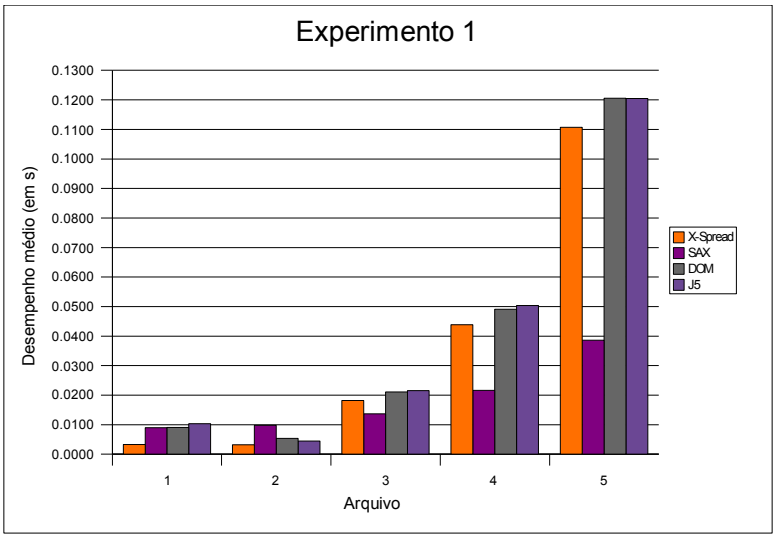

Figura 5.9: Resultados do experimento 1

início dos testes e ocorre em tempos não significantes, quando comparados aos tempos de execução dos pacotes.

É importante ressaltar que testes preliminares com a versão não modificada do esquema disponibilizado pela Wikipedia introduzem atrasos naturalmente variáveis em função do tempo de acesso à rede. Estes atrasos devem-se a dois fatores: o próprio esquema referencia um esquema disponível na *web*, além de ele próprio também encontrar-se na *web*. Entretanto, por menores que sejam, estes atrasos para acesso a esquemas disponíveis na *web* tornam o X-Spread uma opção de desempenho superior quando comparado aos demais pacotes de revalidação. Isto decorre do fato de que o X-Spread não demanda um acesso ao esquema XML a fim de revalidar cada documento, tal como ocorre com os demais pacotes avaliados.

5.3.2 Experimento 1

Neste experimento os pacotes de revalidação processam documentos XML compatíveis com o esquema XML, cujo tamanho físico e número de nodos é maior a cada documento processado. O número de nodos de um documento é em média 40% superior ao número de nodos do documento anterior. O tamanho físico do documento não segue a mesma razão de crescimento, na medida em que o segundo documento do experimento é menor que o primeiro.

A Figura 5.9 apresenta graficamente os resultados deste experimento. Através da análise deste gráfico e da Tabela 5.1, é possível identificar que o X-Spread apresenta os melhores resultados dentre os pacotes avaliados quando os documentos processados possuem pequeno tamanho físico. Assim, o X-Spread apresenta o melhor desempenho quando processa o primeiro e o segundo documento deste experimento, sendo que o SAX apresenta melhor desempenho no restante dos casos.

5.3.3 Experimento 2

No experimento 2 os documentos XML não são compatíveis com o esquema XML, pois não definem um elemento de cardinalidade obrigatória. Na seqüência de documentos utilizados neste experimento, cada documento apresenta um incremento médio de 23% no número de nodos em relação ao documento anterior da seqüência, enquanto o incremento

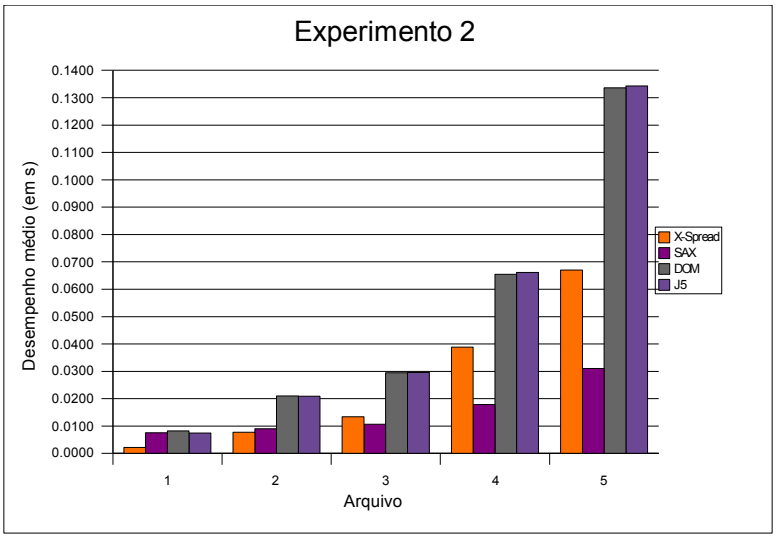

Figura 5.10: Resultados do experimento 2

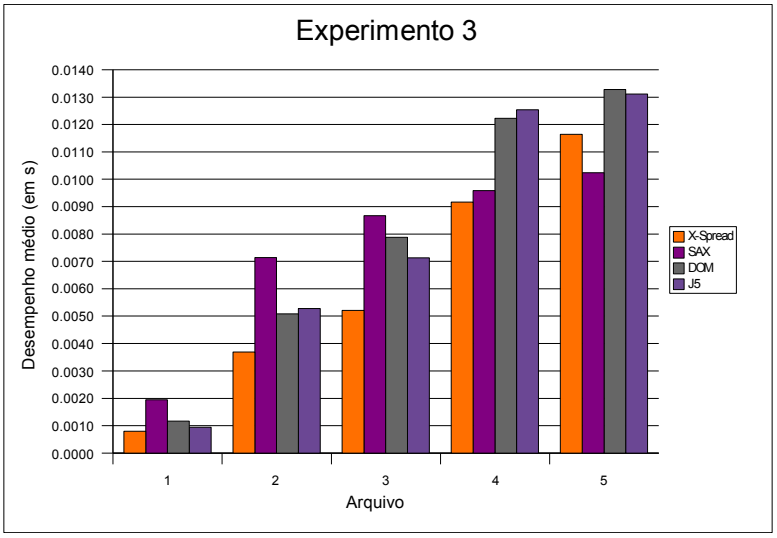

Figura 5.11: Resultados do experimento 3

médio no tamanho do documento é de 40%.

A Figura 5.10 apresenta graficamente os resultados deste experimento. A análise deste gráfico permite identificar a mesma tendência observada no Experimento 1: o X-Spread possui o melhor desempenho dentre os pacotes de revalidação quando processa documentos pequenos. Novamente o SAX apresenta o melhor desempenho no restante dos casos de teste.

5.3.4 Experimento 3

No experimento 3 os documentos XML não são compatíveis com o esquema XML, pois possuem um elemento não definido no esquema. A seqüência de documentos apresenta um incremento médio de 20% em relação ao documento anterior da seqüência no número de nodos. Já o tamanho do documento é incrementado em média em 30%.

A Figura 5.11 apresenta graficamente os resultados deste experimento. Neste experimento o pacote SAX apresenta o melhor desempenho em todos os casos. Neste cenário, o tempo de resposta do X-Spread é aumentado, pois o elemento retirado do esquema

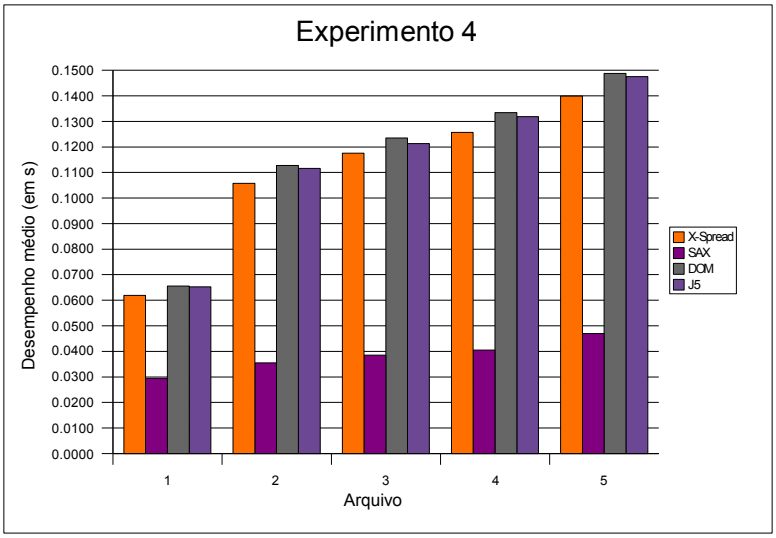

Figura 5.12: Resultados do experimento 4

encontra-se nos documentos XML. Como este elemento encontra-se nos documentos XML, a biblioteca XPath utilizada pelo X-Spread recupera os valores associados a estes elementos, e os disponibiliza em memória para processamento pelo usuário, o que justifica o baixo desempenho.

5.3.5 Experimento 4

No experimento 4 os documentos XML não são compatíveis com o esquema XML, pois não possuem um atributo de cardinalidade obrigatória, conforme definição do esquema. A seqüência de documentos processada neste experimento apresenta incremento médio de 10% no número de nodos e de 15% no tamanho físico.

Os documentos avaliados neste experimento são os maiores documentos analisados nestes testes realizados com o X-Spread. Com estes documentos, objetiva-se verificar o desempenho do X-Spread quando este processa documentos grandes. A Figura 5.12 apresenta graficamente os resultados deste experimento. Conforme observado nos experimentos anteriores, o X-Spread apresenta baixo desempenho quando processa grandes documentos. Neste experimento, o SAX apresentou o melhor desempenho em todos os casos, tendo o X-Spread o segundo melhor desempenho.

5.3.6 Experimento 5

No experimento 5 os documentos avaliados são incompatíveis com o esquema XML, pois não possuem elementos e atributos definidos como obrigatórios no esquema. Ao mesmo tempo, estes documentos definem elementos não presentes no esquema. A seqüência de documentos processados neste experimento não apresenta variação no número de nodos, enquanto o tamanho físico dos documentos varia em média em 15%.

O uso de documentos com número fixo de nodos visa confirmar o comportamento do X-Spread em relação à variação do tamanho dos documentos avaliados. A Figura 5.13 apresenta graficamente os resultados deste experimento. Neste experimento o X-Spread apresentou o melhor desempenho em 3 dos 5 documentos avaliados. No restante dos casos, o SAX apresentou o melhor desempenho.

Considerando o desempenho do pacote SAX, este pode ser empregado pelo X-Spread quando a revalidação completa de documentos se faz necessária, ou mesmo em determi-

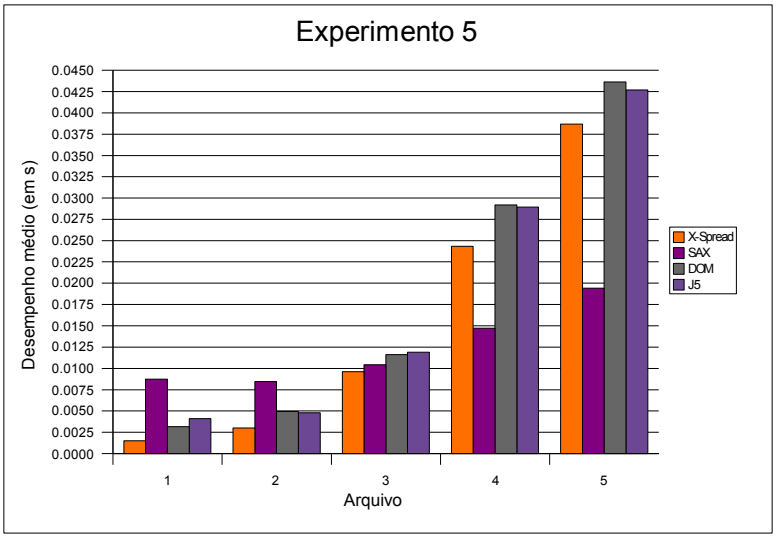

Figura 5.13: Resultados do experimento 5

nados cenários da revalidação parcial. O emprego do SAX na revalidação de documentos depende de diferentes fatores: o tamanho do documento em questão e o tempo de acesso ao esquema referenciado pelo documento a ser revalidado. O tamanho do documento influencia a escolha do algoritmo de revalidação na medida em que o X-Spread possui melhor desempenho quando aplicado a documentos menores, enquanto o SAX processa mais rapidamente documentos maiores. Já o tempo de acesso ao esquema influencia no desempenho do SAX, uma vez que este pacote necessita da definição do esquema a fim de realizar a validação do documento. Como o esquema pode estar disponível em uma rede de dados de baixo desempenho, nestes casos, o X-Spread pode ser empregado com melhor desempenho na revalidação de documentos.

5.3.7 Experimento 6

O protótipo do X-Spread ainda é submetido a um teste adicional, que corresponde ao cômputo do tempo médio utilizado para adaptação de documentos incompatíveis com o esquema modificado e armazenamento destes documentos no sistema de arquivos. A Figura 5.11 ilustra os tempos de revalidação, adaptação e armazenamento dos mesmos arquivos utilizados nos testes de revalidação. Como a primeira seqüência de documentos não necessita de adaptação, esta não se encontra na Figura 5.11.

O tempo consumido pelo processo de adaptação corresponde à soma do tempo necessário para manipular os documentos XML e armazená-los no sistema de arquivos. A análise do gráfico da Figura 5.11 permite a visualização da proporção de tempos consumidos por estas tarefas. Em mais de metade dos casos de teste o processo de adaptação é realizado em tempo inferior ao tempo de revalidação. A quinta seqüência de testes evidencia que o número de nodos não possui grande influência no desempenho do processo de manipulação do documento. Naturalmente o tempo de armazenamento do documento é alto nestes casos, entretanto, isto decorre do suporte em *hardware* disponível. Cabe observar que documentos grandes, como o último arquivo da quinta seqüência de testes, são adaptados em tempos até três vezes maiores que aquele utilizado na revalidação, quando considerado o somatório do tempo de manipulação do documento e seu armazenamento.

Na maioria dos casos avaliados, a manipulação dos documentos ocorreu em um décimo do tempo necessário para armazenamento dos documentos. De maneira semelhante,

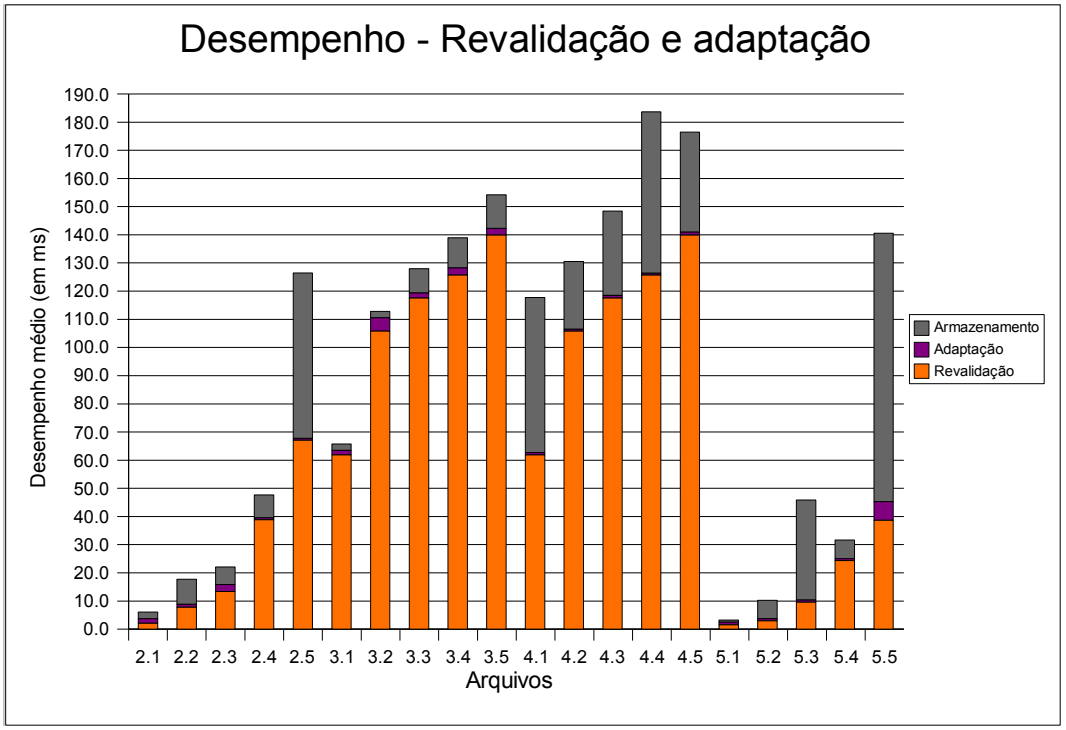

Figura 5.14: Resultados do experimento 6

na maioria dos casos a manipulação dos documentos ocorreu em tempos insignificantes quando comparada à revalidação dos documentos. Por fim, quando o processo de manipulação e armazenamento dos documentos é comparado à revalidação, o primeiro ocorre na metade do tempo necessário para revalidação dos documentos.

5.4 Considerações finais

Este capítulo apresentou a arquitetura da implementação e a descrição do protótipo do X-Spread. A descrição do protótipo é seguida por uma avaliação de desempenho, na qual o X-Spread é comparado ao desempenho de outros pacotes de revalidação de documentos XML, como o SAX, DOM e o pacote de revalidação de documentos XML do Sun Java 5.

A arquitetura do X-Spread define o mecanismo como um sistema observador, que periodicamente monitora a definição de esquemas XML configuráveis pelo administrador do banco de dados. Ao identificar diferenças entre o esquema publicado em um servidor e a versão deste esquema armazenada em sua base de dados interna, o X-Spread inicia o processo de revalidação e adaptação dos documentos XML que referenciam o esquema modificado.

Quando comparado a trabalhos na área da evolução de esquemas e adaptação de documentos XML, o X-Spread atende um maior número de critérios de comparação, definidos na Seção 3.3. As Tabelas 5.2 e 5.3 sumarizam estes critérios e as características das propostas.

O X-Spread suporta a propagação de modificações em esquemas para documentos XML, como a maior parte das propostas analisadas. O processo de adaptação de documentos do X-Spread se diferencia pois permite a adaptação de documentos armazenados em sistemas de arquivos e mensagens XML, sem exigir que estes artefatos sejam armazenados em um banco de dados relacional, por exemplo. A adaptação dos documentos

Tabela 5.2: Comparativo X-Spread e trabalhos relacionados: modificação de esquemas e propagação de mudanças em propostas acadêmicas

Itens / Propostas	X-Spread	Coox	Bertino	Su	Al-Jadir	Guerrini	Bouchou
Propagação de modificações	√	×	×	√	√	√	×
Independência de ambiente para alteração de esquema	√	×	×	×	×	×	×
Adaptação de documentos em sistemas de arquivos	√	NA	×	×	×	√	√
Adaptação sem intervenção do administrador	√	NA	NA	√	√	×	×
Evolução on-line	√	−−	×	√	√	√	−−
Restrições de integridade dos esquemas	√	√	√	√	√	√	NA
Restrições de integridade dos documentos	√	×	√	√	√	√	√
Modelo de dados	√	×	×	×	×	×	×
Taxonomia de modificações	√	×	×	×	×	×	×

Legenda: √ Possui × Não possui −− Não conhecido NA Não Aplicável

Tabela 5.3: Comparativo X-Spread e trabalhos relacionados: modificação de esquemas e propagação de mudanças em implementações de bancos de dados

Itens / Propostas	X-Spread	DB2	Oracle	SQL Server	Tamino
Propagação de modificações	√	×	√	√	×
Independência de ambiente para alteração de esquema	√	×	×	×	×
Adaptação de documentos em sistemas de arquivos	√	×	×	×	×
Adaptação sem intervenção do administrador	√	√	×	×	×
Evolução on-line	√	√	√	√	√
Restrições de integridade dos esquemas	√	√	√	√	√
Restrições de integridade dos documentos	√	√	√	√	√
Modelo de dados	√	×	×	×	×
Taxonomia de modificações	√	×	×	×	×

Legenda: √ Possui × Não possui

ocorre sem intervenção do administrador do banco de dados. Quaisquer intervenções serão efetuadas pelo usuário final da base de dados, uma vez que este possui conhecimento sobre as regras de negócio abordadas pelo banco de dados em questão.

O X-Spread ainda permite que as modificações nos esquemas sejam efetuadas por quaisquer aplicações de edição de esquemas. Logo, ocorre independência entre o ambiente utilizado para modificação do esquema e o X-Spread. O X-Spread realiza o processamento de documentos XML integrantes de uma base de dados enquanto aplicações que acessam esta base encontram-se operacionais. O X-Spread ainda define um modelo de dados para registro do histórico de modificações realizadas em um esquema XML, além de definir uma taxonomia para modificações realizadas nestes esquemas.

Quanto às restrições de integridade de esquemas e documentos implementadas pelo X-Spread, algumas observações devem ser feitas. Como o X-Spread não oferece dispositivos para alteração dos esquemas XML, não é possível garantir que um esquema seja válido. Entretanto, o X-Spread garante que apenas esquemas válidos desencadearão processos de revalidação e adaptação de documentos. Esta medida visa evitar a propagação de definições incorretas em um esquema para as instâncias da base de dados.

Já a restrição de integridade dos documentos XML é parcial na medida em que documentos adaptados pelo X-Spread podem não ser totalmente compatíveis com um esquema. Quando da inclusão de novos elementos obrigatórios em um esquema, esta ope-

ração deve ser refletida nos documentos. Como o X-Spread não especifica o valor destes novos elementos nos documentos, há a possibilidade de que este documento, mesmo adaptado, não seja compatível com a versão mais recente do esquema. A fim de contornar esta possibilidade, ao administrador do banco de dados é oferecida a possibilidade de desabilitar operações de adaptação que levem documentos a estados inconsistentes.

Quanto ao teste de desempenho dos pacotes de revalidação, é possível identificar que o X-Spread possui melhor desempenho na revalidação quando aplicado a documentos de menor tamanho, em torno de 70 Kbytes. Este teste de desempenho também permitiu a identificação do SAX como alternativa a ser empregada quando a revalidação completa de um documento faz-se necessária.

6 CONCLUSÕES

O X-Spread é definido neste trabalho como um mecanismo de suporte aos processos de evolução de esquemas XML e aos processos de revalidação e adaptação de documentos XML. Este suporte à evolução de esquemas consiste na modelagem da evolução do esquema, através da caracterização das possíveis modificações em um esquema e do impacto que estas possuem nos documentos XML. O processo de revalidação especifica quais validações devem ser executadas em documentos XML quando da evolução de um esquema, a fim de identificar se a consistência de um documento em relação a um esquema é mantida. Já o processo de adaptação de documentos especifica quais modificações devem ser realizadas nestes artefatos quando estes passam a não respeitar as novas definições de um esquema submetido a modificações.

A partir das definições propostas para este trabalho, construídas também com base na análise das características das demais propostas existentes para modelagem da evolução de esquemas XML, o X-Spread é assim caracterizado:

- sistema observador - o X-Spread atua como um sistema observador, de maneira que consultas aos esquemas são emitidas a intervalos regulares, a fim de identificar a realização de modificações;

- revalidação parcial - a revalidação dos documentos XML é parcial, sendo que somente as porções dos documentos correspondentes aos elementos modificados no esquema XML são revalidadas após a identificações de alterações no esquema;

- desempenho no processo de revalidação - a identificação de diferentes cenários do processo de revalidação permite a sua otimização, a fim de reduzir o tempo de execução deste processo, que pode ser aplicado a um grande número de documentos;

- adaptação de documentos - o processo de adaptação de documentos XML não requer intervenção do administrador do banco de dados, sendo executado automaticamente quando documentos incompatíveis com um esquema são identificados;

- revalidação e adaptação de mensagens - os processos de revalidação e documentos XML aplicam-se também a mensagens XML, ou seja, documentos trocados por duas aplicações através de uma rede de dados;

- revalidação e adaptação de documentos em sistemas de arquivos - os processos de revalidação e adaptação de documentos XML atuam diretamente nestes artefatos quando estes encontram-se armazenados em sistemas de arquivos, sem exigir que sejam previamente armazenados em bases de dados relacionais.

O diferencial do X-Spread em relação a demais trabalhos na área de evolução de esquemas XML consiste na independência entre o X-Spread e a aplicação utilizada para edição do esquema. É importante destacar a capacidade do X-Spread de revalidar e adaptar documentos e mensagens XML sem que estes sejam armazenados em uma base intermediária, além da especificação do processo de revalidação parcial de documentos. Este processo apresenta ganho de desempenho em relação ao processo de revalidação tradicional de documentos XML, uma vez que atua somente em porções dos documentos correspondentes às porções modificadas no esquema XML.

Uma série de trabalhos acadêmicos recentes especificam soluções para o problema da evolução de esquemas XML e o impacto que esta evolução possui sobre documentos existentes. Dentre estes trabalhos, encontram-se (COOX, 2003), (BERTINO et al., 2002), (SU et al., 2001), (AL-JADIR; EL-MOUKADDEM, 2004), (GUERRINI; MESITI; ROSSI, 2005) e (BOUCHOU et al., 2006). Diferentes implementações de bancos de dados comerciais especificam soluções para este mesmo problema, como é o caso do DB2 (NICOLA; LINDEN, 2005), Oracle (ORACLE, 2004), SQL Server (PAL; FUSSELL; DOLOBOWSKY, 2004) e Tamino (SOFTWAREAG, 2006). Diversos destes trabalhos, como os de Su, Bouchou e Al-Jadir, exigem que os documentos encontrem-se armazenados em bases de dados relacionais quando do início do processo de adaptação. Já trabalhos como o de Guerrini, exigem que interfaces específicas sejam utilizadas para edição do esquema. Tais restrições não fazem parte da definição do X-Spread.

Como produção científica foram publicados os seguintes artigos:

- SILVEIRA, V. N. K. da; GALANTE, R. M. X-Spread - Um mecanismo para propagação da evolução de esquemas para documentos XML. In: V WORKSHOP DE TESES E DISSERTAÇÕES EM BANCO DE DADOS (WTDBD), 2006, Florianópolis, SC. Anais... Sociedade Brasileira de Computação, 2006. p. 26-31.

- PERINI, A. B.; SILVEIRA, V. N. K. da; GALANTE, R. M. XSDelta - Uma ferramenta visual para comparação de esquemas XML. In: SESSÃO DE DEMOS DO SBBD, 2006, Florianópolis, SC. Anais. . . Sociedade Brasileira de Computação, 2006. p.07-12.

Alguns trabalhos foram desenvolvidos em pararelelo com esta dissertação:

- XSDelta - Uma ferramenta visual para comparação de esquemas XML. Trabalho de conclusão de curso em Ciência da Computação de Augusto Belotto Perini. Desenvolvido paralelamente à este trabalho, o XSDelta consiste em uma implementação visual do módulo de detecção de diferenças do X-Spread. Orientado por Renata de Matos Galante, o XSDelta teve co-orientação de Vincent Nelson Kellers da Silveira;

- Estudo Comparativo entre Algoritmos de Detecção de Diferenças entre Esquemas XML. Este estudo, realizado por Rodrigo Scheffer Lumertz, corresponde ao relatório de sua bolsa de Iniciação Científica, com orientação de Renata de Matos Galante e co-orientação de Vincent Nelson Kellers da Silveira;

- Implementação do protótipo do X-Spread, por Rodrigo Scheffer Lumertz, Renata de Matos Galante e Vincent Nelson Kellers da Silveira. Esta implementação encontra-se em andamento.

Ao longo deste trabalho foram mostradas as vantagens das definições do X-Spread, que permitem a manutenção da consistência de uma base de dados semiestruturados

quando da evolução do esquema. Documentos XML distribuídos em diferentes nodos de rede são revalidados e adaptados conforme necessário pelo X-Spread, após a identificação de modificações no esquema, de maneira a garantir a consistência da base de dados. As mesmas definições de revalidação parcial e adaptação de documentos aplicam-se a mensagens XML, documentos trocados entre diferentes aplicações através de uma rede de dados.

Ao longo do desenvolvimento deste trabalho, diversos aspectos do X-Spread foram identificados como passíveis de melhorias ou extensões:

- definição e implementação de um serviço de descoberta de documentos XML que referenciam esquemas monitorados pelo X-Spread. Desta maneira, a base de dados interna do X-Spread pode ser populada automaticamente com a localização de documentos que devem ser revalidados quando da evolução de um esquema XML;

- a especificação dos valores corretos de novos nodos obrigatórios inseridos em documentos XML pode fazer uso de diferentes técnicas para sua implementação. Técnicas de mineração de dados podem ser utilizadas, a fim de realizar a identificação de padrões de valores para novos nodos de documentos XML. Técnicas da área de ontologia podem ser empregadas, a fim de definir os valores de novos nodos a partir do conhecimento adquirido sobre a base de dados semiestruturados em questão;

- extensão da definição da arquitetura do X-Spread para suporte à distribuição do mecanismo, com emprego de uma arquitetura *peer-to-peer*. Esta extensão visa descentralizar as configurações do X-Spread e garantir a presença do mecanismo em diferentes redes de dados, de maneira a aumentar o desempenho dos processos de revalidação e adaptação de documentos, uma vez que idealmente, os documentos a revalidar e adaptar estarão na rede local de um nodo onde o X-Spread é executado;

- extensão da arquitetura do X-Spread para revalidação e adaptação de cláusulas de consulta construídas com base em um esquema XML. Esta extensão visa oferecer suporte a aplicações que definem consultas a partir de um esquema, e que podem ser levadas a um estado inválido quando da modificação de um esquema.

REFERÊNCIAS

AB, M. **MySQL AB**. Disponível em: <http://www.mysql.com/>. Acesso em: jun. 2007.

AL-JADIR, L.; EL-MOUKADDEM, F. F2/XML: managing xml document schema evolution. In: INTERNATIONAL CONFERENCE ON ENTERPRISE INFORMATION SYSTEMS, ICEIS, 6, 2004, Porto. **Proceedings...** Setubal: Escola Superior de Tecnologia de Setubal, 2004. p.251–258.

BARBOSA, D. et al. Efficient Incremental Validation of XML Documents. In: ICDE, 2004. **Proceedings...** [S.l.]: IEEE Computer Society, 2004. p.671–682.

BERTINO, E. et al. Evolving a Set of DTDs According to a Dynamic Set of XML Documents. In: EDBT 2002, WORSHOPS XMLDM, MDDE, AND YRWS, Prague, Czech Republic. **XML-Based Data Management and Multimedia Engineering:** revised papers. Berlin: Springer, 2002. p. 45–66.

BEX, G. J.; NEVEN, F.; BUSSCHE, J. V. den. DTDs versus XML schema: a practical study. In: INTERNATIONAL WORKSHOP ON THE WEB AND DATABASES, WEBDB, 7., 2004, New York, NY, USA. **Proceedings...** New York: ACM Press, 2004. p.79–84.

BOUCHOU, B.; ALVES, M. H. F. Updates and Incremental Validation of XML Documents. In: DBPL, 2003. **Database Programming Languages:** proceedings... Berlin: Springer, 2003. p.216–232. (Lecture Notes in Computer Science, v.2921).

BOUCHOU, B.; CHERIAT, A.; ALVES, M. H. F.; SAVARY, A. XML Document Correction: incremental approach activated by schema validation. In: IDEAS, 2006. **Proceedings...** [S.l.]: IOS Press, 2006. p.228–238.

CAMOLESI, L. J.; TRAINA, C. J. Evolução de Esquemas de Dados: um panorama amplo de aspectos técnicos e gerenciais. In: SIMPÓSIO BRASILEIRO DE BANCO DE DADOS, SBBD, 11., 1996, São Carlos, SP. **Anais...** São Carlos: Universidade Federal de São Carlos, 1996. p.1–19.

COBENA, G et al. Detecting Changes in XML Documents. In: ICDE, 2002. **Proceedings...** [S.l.]: IEEE Computer Society, 2002. p.41–52.

CODEHAUS. **jaxen**: universal jaxa xpath engine. Disponível em: <http://jaxen.org/>. Acesso em: jun. 2007.

COHEN, S. et al. EquiX Easy Querying in XML Databases. In: WEBDB, 1999. **Proceedings...** [S.l.: s.n.], 1999. p.43–48.

COOX, S. V. Axiomatization of the Evolution of XML Database Schema. **Program. Comput. Softw.**, New York, NY, USA, v.29, n.3, p.140–146, 2003.

DYRESON, C. E. Observing Transaction-Time Semantics with TTXPath. In: INTERNATIONAL CONFERENCE ON WEB INFORMATION SYSTEMS ENGINEERING, WISE, 2., 2001. **Proceedings...** [S.l.]: IEEE Computer Society, 2001. v. 1, p.193.

EPSTEIN, D.; CURBERA, F. **XML TreeDiff - A set of Java beans that enable efficient differentiation and updating of DOM trees**. Disponível em: <http://alphaworks.ibm.com/tech/xmltreediff>. Acesso em: jun. 2007.

GALANTE, R. M. **Um Modelo de Evolução de Esquemas Conceituais para Bancos de Dados Orientados a Objetos com o Emprego de Versões**. 2003. Tese (Doutorado em Ciência da Computação) — Instituto de Informática, Universidade Federal do Rio Grande do Sul, Porto Alegre.

GUERRINI, G.; MESITI, M.; ROSSI, D. Impact of XML schema evolution on valid documents. In: ANNUAL ACM INTERNATIONAL WORKSHOP ON WEB INFORMATION AND DATA MANAGEMENT, WIDM, 7., 2005, New York, NY, USA. **Proceedings...** New York: ACM Press, 2005. p.39–44.

HAROLD, E. R. **The Jaxa XML Validation API**. Disponível em: <http://www-128.ibm.com/developerworks/xml/library/x-javaxmlvalidapi.html>. Acesso em: jun. 2007.

JENSEN, C. S. et al. The Consensus Glossary of Temporal Database Concepts - February 1998 Version. In: ETZION, O.; JAJODIA, S.; SRIPADA, S. (Ed.). **Temporal Databases**: research and practice. Berlin: Springer-Verlag, 1998. p.367–405. (Lecture Notes in Computer Science, v.1399).

LEONARDI, E.; BHOWMICK, S. S.; MADRIA, S. K. Xandy: detecting changes on large unordered xml documents using relational databases. In: DASFAA, 2005. **Proceedings...** Berlin: Springer, 2005. p.711–723. (Lecture Notes in Computer Science, v.3453).

MESITI, M. et al. X-Evolution: a system for xml schema evolution and document adaptation. In: EDBT, 2006. **Proceedings...** Berlin: Springer, 2006. p.1143–1146. (Lecture Notes in Computer Science, v.3896).

NICOLA, M.; LINDEN, B. van der. Native XML support in DB2 universal database. In: INTERNATIONAL CONFERENCE ON VERY LARGE DATA BASES, VLDB, 31., 2005. **Proceedings...** New York: ACM, 2005. p.1164–1174.

NOH, S.-Y.; GADIA, S. K. An XML-Based Framework for Temporal Database Implementation. In: INTERNATIONAL SYMPOSIUM ON TEMPORAL REPRESENTATION AND REASONING, TIME, 12., 2005. **Proceedings...** [S.l.]: IEEE Computer Society, 2005. p.180–182.

ORACLE. **Oracle Database Online Online Documentation 10g Release 1 (10.1)**. Disponível em: <http://www.stanford.edu/dept/itss/docs/oracle/10g/index.htm>. Acesso em: jun. 2007.

PAL, S.; FUSSELL, M.; DOLOBOWSKY, I. **XML Support in Microsoft SQL Server 2005**. Disponível em: <http://msdn2.microsoft.com/en-us/library/ms345117.aspx>. Acesso em: jun. 2007.

PAPAKONSTANTINOU, Y.; VIANU, V. Incremental Validation of XML Documents. In: ICDT, 2003. **Proceedings...** Berlin: Springer, 2003. p.47–63. (Lecture Notes in Computer Science, v.2572).

PERINI, A. B.; SILVEIRA, V. N. K. da; GALANTE, R. M. XSDelta - Uma ferramenta visual para comparação de esquemas XML. In: SIMPÓSIO BRASILEIRO DE BANCO DE DADOS, SBBD, 21., 2006, Florianópolis. **Anais da III Sessão de Demos.** Florianópolis: SBC, 2006. p. 7–12.

PROJECT, S. **SAX Project**. Disponível em: <http://www.saxproject.org/>. Acesso em: jun. 2007.

RAGHAVACHARI, M.; SHMUELI, O. Efficient Schema-Based Revalidation of XML. In: EDBT, 2004. **Proceedings...** Berlin: Springer, 2004. p.639–657. (Lecture Notes in Computer Science, v.2992).

SILVEIRA, V. N. K. da. **Estudo sobre evolução de esquemas XML**. 2006. Trabalho Individual (Mestrado em Ciência da Computação) — Instituto de Informática, Universidade Federal do Rio Grande do Sul, Porto Alegre.

SILVEIRA, V. N. K. da; GALANTE, R. M. X-Spread - Um mecanismo para propagação da evolução de esquemas para documentos XML. In: WORKSHOP DE TESES E DISSERTAÇÕES EM BANCO DE DADOS, WTDBD, 5., 2006, Florianópolis, SC. **Anais...** Florianópolis: Sociedade Brasileira de Computação, 2006. p.26–31.

SNODGRASS, R. T. **Developing time-oriented database applications in SQL**. San Francisco, CA, USA: Morgan Kaufmann, 2000.

SOFTWAREAG. **Tamino XML Schema User Guide**. Disponível em: <http://documentation.softwareag.com/crossvision/ins441/tsl/modschema.htm>. Acesso em: jun. 2007.

SU, H. et al. XEM: managing the evolution of xml documents. In: INTERNATIONAL WORKSHOP ON RESEARCH ISSUES IN DATA ENGINEERING, RIDE, 2001. **Proceedings...** [S.l.]: IEEE Computer Society, 2001. p.103.

SUN MICROSYSTEMS. **JDK 5 Documentation**. Disponível em: <http://java.sun.com/j2se/1.5.0/docs/index.html>. Acesso em: jun. 2007.

W3C. **Extensible Markup Language (XML) 1.0**. Disponível em: <http://www.w3.org/TR/1998/REC-xml-19980210/>. Acesso em: jun. 2007.

W3C. **W3C XML Schema**. Disponível em: <http://www.w3.org/XML/Schema>. Acesso em: jun. 2007.

W3C. **W3C Document Object Model**. Disponível em: <http://www.w3.org/DOM/>. Acesso em: jun. 2007.

W3C. **Extensible Markup Language (XML) 1.0 (Fourth Edition)**. Disponível em: <http://www.w3.org/TR/2006/REC-xml-20060816/>. Acesso em: jun. 2007.

WANG, Y.; DEWITT, D. J.; CAI, J. yi. X-Diff: an effective change detection algorithm for xml documents. In: ICDE, 2003. **Proceedings...** [S.l.]: IEEE Computer Society, 2003. p.519–530.