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

RODRIGO GASPARONI SANTOS

Evolução de Documentos XML com Tempo e Versões

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

Profª. Dr.ª Nina Edelweiss
Orientadora

Profª. Dr.ª Renata de Matos Galante
Co-orientadora

Porto Alegre, maio de 2005

CIP – CATALOGAÇÃO NA PUBLICAÇÃO

Santos, Rodrigo Gasparoni

Evolução de Documentos XML com Tempo e Versões / Rodrigo Gasparoni Santos – Porto Alegre: Programa de Pós-Graduação em Computação, 2005.

92 f.: il.

Dissertação (mestrado) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação, Porto Alegre, BR-RS, 2005. Orientadora: Nina Edelweiss; Co-orientadora: Renata de Matos Galante.

1. XML. 2. Modelos de Dados Bitemporais. 3. Versionamento. I. Edelweiss, Nina. II. Galante, Renata de Matos. III. Título.

AGRADECIMENTOS

A todos os meus queridos familiares, em especial a meus pais, Odilson e Carmen, por sempre me incentivarem a investir nos estudos. A todos os colegas e amigos que, seja com conselhos técnicos ou meramente através de sua presença, contribuíram para que este projeto se completasse. Às minhas orientadoras, Profª. Dr.ª Nina Edelweiss e Profª. Dr.ª Renata de Matos Galante, sempre prontas a me auxiliar com qualquer dúvida que surgisse – em especial as de última hora. Sua experiência foi indispensável para a elaboração deste trabalho. A todos os professores e funcionários do Instituto de Informática da UFRGS, por sua incondicional cooperação durante esses últimos anos. Ao CNPq, pelo auxílio concedido. Ao Sr. João Alberto de Oliveira Lima, pela ajuda indispensável para a realização do Estudo de Caso.

A todos vocês, muito obrigado.

SUMÁRIO

LISTA DE ABREVIATURAS

DML	Data Manipulation Language
DNN	Durable Node Number
DTD	Document Type Definition
HTML	Hypertext Markup Language
RBVM	Reference-Based Version Model
SPaR	Sparse Preorder and Range
TVX	Tempo e Versões em XML
UBCC	Usefulness-Based Change Control
UML	Unified Modeling Language
URL	Uniform Resource Location
W3C	World Wide Web Consortium
XID	Xyleme Identifier
XML	Extensible Markup Language
XSLT	Extensible Stylesheet Language Transformations

LISTA DE FIGURAS

LISTA DE TABELAS

RESUMO

A utilização de conceitos de representação temporal tem sido essencial em diversas aplicações de banco de dados, por permitir o armazenamento e a manipulação dos diferentes estados assumidos pela base de dados ao longo do tempo. Durante a evolução da base de dados, através do conceito de bitemporalidade, obtém-se acesso a informações presentes, passadas e futuras. Já o conceito de versionamento permite a existência de diversas alternativas para a evolução da base de dados, possibilitando um processo de evolução ramificada, em oposição ao usual mecanismo de evolução linear do conteúdo da base.

Com a migração de tais aplicações para um ambiente *Web*, estas passam cada vez mais a utilizar a linguagem XML como formato de representação e intercâmbio de seus dados. Tornam-se necessários, dessa forma, mecanismos para a representação e manipulação da história do conteúdo de um documento XML que sofre modificações com o passar do tempo. Apesar da existência de propostas de extensão temporal de modelos de dados convencionais e de estratégias para o armazenamento de documentos XML em modelos convencionais, a natureza semi-estruturada dos documentos XML faz com que seja necessário definir um novo modelo de dados temporal, capaz de lidar com os conceitos de bitemporalidade e versionamento em um documento semi-estruturado.

O objetivo deste trabalho é definir um modelo que, ao contrário das demais propostas existentes, combine os conceitos de bitemporalidade e de versionamento em uma única abordagem capaz de permitir o tratamento da evolução do conteúdo de documentos XML. O uso conjunto desses dois recursos visa combinar o poder de expressão de cada um, garantindo uma maior flexibilidade na representação do histórico dos documentos XML. O modelo resultante recebeu o nome de *Tempo e Versões em XML*, ou simplesmente TVX, composto por três partes: um modelo para a organização lógica dos dados, uma linguagem de consulta e uma linguagem para promover alterações ao conteúdo dos documentos XML.

Palavras-chave: XML, Modelos de Dados Bitemporais, Versionamento

Evolution of XML Documents with Time and Versions

ABSTRACT

The use of temporal representation concepts assumed an essential role in several database applications, for allowing the storage and manipulation of the different states the data assumes throughout time. Due to the evolution of the data, the concept of bitemporality allows the access to present, past and future information. On the other hand, the versioning concept allows the storage of different alternatives of an object's history, thus creating the possibility for a branched timeline, as opposed to an usual linear timeline.

With the migration of such applications to a Web environment, the use of the XML language as a standard format for representation and exchange of its internal data is identified. Due to this fact, there is the need for methods for managing the history of the content of a XML document that goes through modifications during the course of time. In spite of the existence of many proposals for temporal extensions of conventional data models and of strategies for storing XML documents in conventional databases, the semi-structured nature of such documents creates the need to define a new data model, capable of handling the concepts of bitemporality and versioning in a semi-structured document.

The goal of this work is to define a model that, unlike other existent proposals, combines the bitemporality and versioning concepts into a single approach able to manage the evolution of the content of XML documents. The united use of those concepts aims to combine their expressive power, guaranteeing a greater flexibility in the representation of the history of XML documents. The resulting model was called *Time and Versions in XML*, or TVX for short. Its presentation is divided in three parts: a model for the logical organization of the data, a query language and a data manipulation language that operates on the content of temporal XML documents.

Keywords: XML, Bitemporal Data Models, Versioning

1 INTRODUÇÃO

1.1 Motivação

Desde sua criação, em 1998, a linguagem XML (W3C, 1998) vem gradativamente se tornando um padrão para representação e intercâmbio de dados em meio eletrônico. Acredita-se, inclusive, que em um futuro próximo a linguagem XML substituirá a linguagem HTML como meio predominante de representação de informações na Internet. Em função da natureza dinâmica das informações presentes na Web, um problema já amplamente estudado para bases de dados convencionais ganha nova importância: a necessidade da associação de informações de cunho temporal ao conteúdo da base de dados. Em outras palavras, tornam-se necessários mecanismos para a representação e manipulação da história do conteúdo de um documento que sofre modificações com o passar do tempo. Existe uma ampla variedade de propostas que abordam essa questão para bases relacionais – por exemplo, o *Historical Relational Data Model* (CLIFFORD; CROKER, 1987) e o *Temporal Relational MODEL* (NAVATHE; AHMED, 1989) – e orientadas a objeto – tais como o *T-CHIMERA* (BERTINO; FERRARI; GUERRINI, 1996) e o *TVM* (MORO, 2001) – mas poucas que lidam com a categoria na qual se encaixam os documentos XML: dados semi-estruturados.

Como exemplo da necessidade de uma solução específica para XML, basta considerar a grande quantidade de aplicações que a cada dia migram de um ambiente tradicional para uma plataforma Web. Para todas as aplicações nas quais a necessidade do tratamento da informação temporal já se fazia presente ao lidar com os modelos de dados tradicionais, será claramente necessária uma nova solução capaz de oferecer os mesmos recursos já criados para os ambientes tradicionais, porém dentro do contexto da Internet. Um exemplo que se encaixa nessa categoria é a edição cooperativa de documentos, onde dois ou mais agentes alteram o conteúdo de um mesmo objeto em um ambiente distribuído. Tal cenário é um exemplo clássico de uma aplicação que depende do gerenciamento das diversas versões de um mesmo objeto; conforme aplicações dessa natureza migram para um ambiente Web, elas naturalmente adotam XML como formato para representação de informações, passando a buscar por soluções adequadas ao tratamento das múltiplas versões em XML.

Além de aplicações já existentes que migram para a Web, novas aplicações surgem em função da Web. Por exemplo, uma aplicação importante para os mecanismos de busca da Internet é a permanência de elos (*links)* em páginas da Web. Basicamente, qualquer URL que se torne inválida causa problemas para todas as páginas que apontem para ela. O problema é particularmente grave para os mecanismos de busca, pois, se constantemente direcionarem milhões de usuários para páginas que já não existem,

perdem sua credibilidade. Apenas substituir a versão antiga da página por uma nova, no mesmo local, não resolve o problema completamente, pois a nova versão pode não conter mais as palavras-chave usadas na busca. A solução ideal consiste em suportar múltiplas versões de um mesmo documento, evitando o armazenamento redundante de seus segmentos em comum.

Há aplicações cujo propósito é tornar acessível o histórico de modificações de um documento. Aplicações desta classe são responsáveis pelas alterações às quais o documento é submetido, tornando-as visíveis ao público. Um exemplo de uma aplicação desta categoria é tratado em detalhes no estudo de caso contido neste trabalho, apresentado no capítulo 6. Em contrapartida, há também aplicações de monitoramento, cujo objetivo é acompanhar a evolução de um documento modificado por terceiros. Um exemplo de aplicação nesta segunda categoria é o projeto Xyleme (MARIAN et al., 2001), cujo propósito é registrar as modificações que ocorrem em páginas buscadas da Web.

Por fim, uma última classe de aplicações que necessita de mecanismos para o gerenciamento temporal de documentos XML consiste em montar visões XML da evolução de uma base relacional. Em outras palavras, ao invés de se utilizar um modelo de dados relacional temporal para a base que se tem em mãos, registra-se na base relacional apenas o estado atual dos dados, sendo que seu histórico fica armazenado em um arquivo XML à parte. Este método é estudado em detalhes em uma série de trabalhos – (WANG; ZANIOLO, 2002), (WANG; ZANIOLO, 2003a), (WANG; ZANIOLO, 2003b), (WANG; ZANIOLO, 2003c) e (WANG; ZANIOLO, 2004) – abordando desde as diferentes estratégias para o mapeamento das tabelas relacionais para documentos XML até questões referentes ao modelo físico de armazenamento dos dados. A justificativa apresentada pelos autores é que a estrutura hierárquica (ou "temporalmente agrupada") dos arquivos XML é mais expressiva e acomoda mais naturalmente a incorporação de informações temporais, quando comparada com a estrutura plana (ou "temporalmente desagrupada") das tabelas relacionais.

Como há extensões do modelo relacional para lidar com o aspecto temporal, e como há estratégias para armazenar um documento XML em um banco de dados relacional, a princípio pode-se pensar que para oferecer o suporte temporal a documentos XML basta armazená-los em uma base relacional temporal. Porém, um dado XML possui uma natureza diferente de um dado convencional de um BD, uma vez que um dado de BD é totalmente estruturado enquanto que um dado XML é um dado semi-estruturado – isto é, seu esquema de representação é bastante irregular. O fato de um documento XML ter uma organização semi-estruturada define duas categorias de documentos XML (BOURRET, 2004):

- Documentos XML Orientados a Dados – documentos fracamente semi-estruturados, ou seja, possuem uma estrutura mais regular e repetitiva. Apresentam pouco ou nenhum conteúdo misto, isto é, intercalação de trechos de texto e de sub-elementos dentro de um mesmo elemento. Tipicamente, documentos usados para transferência de dados convencionais entre aplicações são os representantes desta classe de documentos. Visões XML de bases relacionais usualmente caem nesta categoria;

- Documentos XML Orientados a Documentos – documentos fortemente semi-estruturados, ou seja, possuem uma estrutura mais irregular e particular. Tipicamente apresentam conteúdo misto em grande quantidade, sendo normalmente usados para o relato da linguagem natural, com destaque para

alguns dados delimitados dentro do seu conteúdo. Livros e anúncios classificados são alguns exemplos.

O modo como os documentos XML são gerenciados está diretamente relacionado a esta classificação. Aplicações que lidam com documentos orientados a dados geralmente utilizam bases relacionais, uma vez que os dados XML são fortemente estruturados, sendo facilmente mapeáveis para tabelas relacionais. Exemplos de tais estratégias de mapeamento podem ser encontrados em (MELLO, 2003), sendo discutidas em maior detalhe em (FLORESCU; KOSSMANN, 1999). Já aplicações que lidam com documentos orientados a documentos utilizam normalmente bases XML nativas, pois são mais efetivas para a representação e o acesso a dados em formato XML do que as bases relacionais. Isto ocorre porque a estrutura destes documentos é muito irregular, tornando muito complexo seu armazenamento e gerenciamento em uma base relacional, ao passo que bases XML nativas já são projetadas para lidar com documentos semi-estruturados.

Assim sendo, fica estabelecido que o mapeamento de um documento XML para um modelo de dados relacional temporal não é uma solução apropriada para oferecer suporte ao gerenciamento temporal de documentos XML genéricos, sendo portanto necessário definir uma extensão ao modelo XML convencional capaz de incorporar as características temporais.

1.2 Objetivos

O objetivo da presente dissertação é definir um modelo que empregue os conceitos de bitemporalidade e de versionamento para permitir o armazenamento da evolução do conteúdo de documentos XML. Este modelo foi batizado de *Tempo e Versões em XML*, ou simplesmente TVX.

O modelo TVX difere das demais propostas existentes justamente por combinar em uma única abordagem as características de bitemporalidade e versionamento, as quais não aparecem simultaneamente nos trabalhos estudados. O uso conjunto desses dois recursos visa combinar o poder de expressão de cada um, garantindo uma maior flexibilidade na representação do histórico dos documentos XML. Embora a importância da evolução de esquemas seja inegável, a mesma não será abordada em detalhes neste trabalho, sendo contudo brevemente discutida no capítulo 7.

Embora seja mostrado um mapeamento do diagrama de classes do modelo TVX para uma implementação utilizando a própria linguagem XML, não há a necessidade do desenvolvimento de um sistema gerenciador de banco de dados específico para este modelo. Em outras palavras, a organização lógica dos dados dentro do modelo TVX não requer um tipo específico de organização física, podendo ser implementado, por exemplo, tanto sobre uma base XML nativa quanto sobre uma base relacional.

1.3 Organização do Texto

O restante do texto está organizado como segue:

- o capítulo 2, "Revisão Bibliográfica", revisa conceitos e idéias fundamentais à compreensão do restante do trabalho, dividindo-os em dois grupos: os conceitos básicos de bitemporalidade e versionamento, empregados em diversos modelos temporais existentes, e a apresentação e análise de diversos modelos e propostas com objetivos similares aos do modelo TVX;

- o capítulo 3, "O Modelo TVX", apresenta detalhadamente o modelo proposto para evolução do conteúdo de documentos XML. Partindo da noção convencional de um documento XML, o capítulo gradualmente mostra como esta estrutura pode ser estendida para incorporar as noções de bitemporalidade e versionamento apresentadas no capítulo 2. Em seguida, é mostrado como o modelo pode ser implementado em XML Schema, e como interpretar as informações nele contidas;

- o capítulo 4, "Linguagem de Consulta", apresenta uma extensão da linguagem XQuery e do modelo XPath capaz de processar consultas a um documento XML dentro do modelo TVX. As consultas são escritas sobre a visão original que o usuário tem dos dados, e não sobre sua codificação interna, tornando-a portanto transparente para o usuário final;

- o capítulo 5, "Linguagem de Manipulação de Dados", apresenta também uma linguagem concebida através de extensões de XQuery e de XPath, porém com o objetivo de fornecer um mecanismo através do qual o usuário possa expressar modificações ao conteúdo de um documento XML dentro do modelo TVX;

- o capítulo 6, "Estudo de Caso", analisa a utilização do modelo TVX em uma aplicação real, discutindo a adequação do modelo proposto aos requisitos da aplicação alvo e comparando os resultados obtidos com a solução adotada atualmente;

- por fim, o capítulo 7, "Conclusões", apresenta uma revisão do trabalho realizado, identificando as principais contribuições e as possibilidades para trabalhos futuros.

Este trabalho também inclui um anexo, denominado "Descrição em XML Schema dos Documentos XML do Estudo de Caso". Como o próprio nome indica, este anexo descreve a estrutura dos documentos XML que serviram de base para a confecção do estudo de caso apresentado no capítulo 6.

2 REVISÃO BIBLIOGRÁFICA

O objetivo deste capítulo é apresentar sumariamente alguns conceitos sobre representação de informações temporais que serão necessários para a compreensão do restante do trabalho, bem como ilustrar algumas propostas presentes na literatura com objetivos similares aos do modelo que será apresentado no próximo capítulo. A seção 2.1 mostra as definições essenciais aos modelos de dados temporais, tais como os tipos de rótulos temporais empregados e como estes devem ser interpretados. A seção 2.2, por sua vez, apresenta o conceito de versões, o qual permite a existência de linhas alternativas na evolução de um documento. A seguir, na seção 2.3, serão apresentados diversos modelos temporais encontrados na literatura, os quais empregam esses conceitos para tentar resolver o problema de oferecer suporte ao gerenciamento do conteúdo de documentos XML.

2.1 Conceitos de Representação Temporal

A representação de dados temporais é feita através do acréscimo de uma nova dimensão a uma base de dados convencional. Esta nova dimensão é usada para representar a passagem do tempo, e é usualmente representada através de rótulos temporais (*timestamps*). Tipicamente, tais rótulos caem em uma de três categorias:

- Rótulos de Tempo de Transação – indicam o momento em que a informação foi registrada na base de dados. Quando presentes, permitem uma operação denominada *rollback*, que consiste em retornar a base de dados ao estado em que estava em um determinado momento passado;
- Rótulos de Tempo de Validade – indicam o tempo durante o qual a informação registrada representa adequadamente a realidade modelada. Quando presentes, permitem a correção nas informações armazenadas, bem como previsões de momentos futuros;
- Rótulos definidos pelo usuário – estas são propriedades temporais definidas explicitamente pelo usuário, cuja manipulação cabe exclusivamente aos programas de aplicação.

Tempo de transação e tempo de validade são propriedades ortogonais, e devem ser usadas em conjunto para se ter a informação completa. Bases de dados que não oferecem qualquer suporte temporal são ditas instantâneas ou de *snapshot*. Conforme o suporte que oferecem ao emprego destes tipos, as bases de dados temporais podem ser classificadas em bases de tempo de transação (quando permitem somente o uso de rótulos de tempo de transação), bases de tempo de validade (analogamente, são as que permitem apenas o uso de rótulos de tempo de validade), bases bitemporais (estas

possibilitam o uso simultâneo dos dois tempos) e bases multitemporais. Este último tipo é o mais flexível, pois mescla dados estáticos com dados temporais, permitindo inclusive que certos dados sejam temporalizados apenas com tempo de transação, outros apenas com tempo de validade e outros com ambos.

Além do tempo que utilizam, os rótulos temporais podem ainda ser classificados quanto:

- à ordenação temporal – em sua forma mais simples, os rótulos temporais provêm uma ordenação total entre quaisquer dois pontos no tempo. Neste caso, são ditos de *tempo linearmente ordenado* ou simplesmente de *tempo linear*. Podem também ser representativos de *tempo circular*, quando modelam eventos e processos que se repetem ao longo do tempo, fechando um ciclo (por exemplo, numa situação em que se deva considerar apenas o dia da semana em que os eventos modelados ocorrem). Por fim, existe também o *tempo ramificado*, através do qual dois ou mais pontos diferentes podem ser antecessores ou sucessores imediatos de um determinado ponto no tempo. O tempo ramificado pode aparecer combinado com o tempo linear; por exemplo, pode haver uma situação na qual, por uma questão de incerteza quanto a eventos passados, estes sejam representados com tempo ramificado, e eventos presentes ou futuros com tempo linear. A Figura 2.1 mostra as diferentes possibilidades de ordenação temporal;

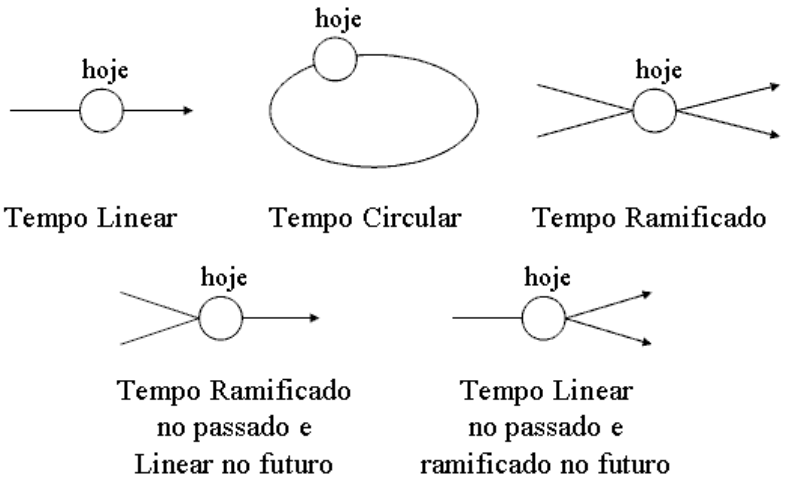


Figura 2.1: Tipos de ordenação temporal

- à variação temporal – os rótulos temporais podem apresentar variação contínua ou discreta. Embora o tempo seja contínuo por natureza, a utilização de tempo discreto simplifica a implementação dos modelos de dados. Uma linha de tempo discreta é composta por uma seqüência de intervalos temporais consecutivos e indecomponíveis, denominados *chronons*. A duração de um *chronon*, também chamada de granularidade, varia conforme a aplicação, podendo ser, por exemplo uma data no formato "dia/mês/ano" ou apresentar um nível mais detalhado, incluindo por exemplo "hora:minuto:segundo";

- ao tipo de rótulo temporal – os rótulos temporais podem indicar um *ponto no tempo*, um *intervalo temporal* ou um *elemento temporal*. Um ponto no tempo é indicado por um único *chronon*, como, por exemplo, "15/02/04". Um intervalo temporal é delimitado por um par de pontos temporais e inclui todos os demais

pontos contidos entre eles na ordenação temporal; por exemplo, um intervalo referente ao mês de março de 2004 pode ser representado por [01/03/04, 31/03/04]. O termo especial *now* pode ser usado para representar o instante atual, movendo-se constantemente ao longo do eixo temporal, separando o passado do futuro. Elementos temporais são compostos através da união finita de intervalos temporais. Independente de qual deles seja usado, o tempo de existência de um objeto é denominado tempo de vida ou *lifespan*.

Como exemplo da aplicação destes conceitos, considere as Tabelas 2.1 e 2.2, que mostram a evolução do valor do salário de um empregado ao longo do tempo. O mecanismo empregado para atualização dos rótulos temporais é o definido em (ELMASRI; NAVATHE, 2000). Os rótulos são representados através de intervalos temporais, com ordenação linear e variação temporal discreta cuja granularidade foi ajustada para dia/mês/ano. A Tabela 2.1 mostra os valores de um empregado que foi registrado na base de dados em 25/07/04 – esta data corresponde, portanto, ao *tempo de transação inicial* da tupla (coluna TTI). O *tempo de transação final* (coluna TTF) está em aberto, o que é indicado pelo termo *now*, significando que não há previsão para que o conteúdo desta tupla deixe de ser válido. O contrato do empregado inicia em 01/08/04 e termina em 30/11/04, o que fica registrado nos atributos *tempo de validade inicial* (coluna TVI) e *tempo de validade final* (coluna TVF).

Tabela 2.1: Propriedades do empregado antes da modificação no salário

Código do Empregado	Salário	TVI	TVF	TTI	TTF
105	300,00	01/08/04	30/11/04	25/07/04	now

Tabela 2.2: Propriedades do empregado após a modificação no salário

Código do Empregado	Salário	TVI	TVF	TTI	TTF
105	300,00	01/08/04	30/11/04	25/07/04	30/08/04
105	500,00	01/08/04	31/12/04	01/08/04	now

Prosseguindo com o exemplo, no dia 01/08/04 o salário do empregado é reajustado e seu contrato é estendido por um mês. A tupla anterior é sobrescrita, substituindo o tempo de transação final pela data 30/08/04, indicando que o conhecimento de que o salário do empregado era de R$ 300,00 e de que seu contrato ia até o fim de novembro era válido somente até esta data. É também adicionada uma nova tupla, com os valores do salário e do tempo de validade final devidamente atualizados (Tabela 2.2). O tempo de transação inicial para esta nova tupla corresponde à data em que a modificação foi feita, e como não há previsão de novas modificações, o tempo de transação final é novamente ajustado para o termo *now*. Futuras modificações podem vir a modificar esta coluna, fechando o intervalo temporal do qual faz parte e inserindo novas tuplas. Maiores detalhes sobre os mecanismos de representação através de rótulos temporais podem ser encontrados em (EDELWEISS; OLIVEIRA, 1994), (JENSEN et al, 1998), (JENSEN, 1999), (SNODGRASS, 2000) e (TANSEL et al, 1993).

Aqui, cabe uma observação importante. No decorrer deste trabalho, os resultados de modificações como a que acaba de ser mostrada são chamadas de *estados*. Em outras palavras, as tuplas que representam os salários de R$ 300,00 e R$ 500,00 correspondem a dois diferentes *estados* de uma mesma entidade no mundo real. Em outros modelos, como os que serão abordados na próxima seção, é comum usar o termo "versões" com o mesmo significado com que aqui se emprega o termo "estados". No contexto deste

trabalho, contudo, ao termo "versão" será reservado outro significado, o qual será apresentado a seguir.

2.2 Conceitos de Representação de Versões

Enquanto o uso de rótulos temporais permite a evolução linear da informação, o versionamento é uma técnica que possibilita a existência de linhas paralelas na evolução de um objeto, representantes de *diferentes alternativas* para a evolução do mesmo. Cada versão é independente das demais, e embora compartilhem a mesma estrutura, cada uma possui o seu próprio histórico. Ao contrário dos rótulos temporais, que são manipulados automaticamente pela base de dados temporal, a criação de versões é um evento que ocorre somente ao comando do usuário. Com exceção da versão inicial, novas versões são criadas através de um processo chamado de *derivação* de versões existentes, onde uma versão serve de base para a criação de outra. Considerando a derivação como uma relação do tipo pai-filho, cria-se uma hierarquia envolvendo o conjunto de versões definidas de acordo com este processo (Figura 2.2); dependendo do modelo de dados, esta hierarquia pode tomar a forma de uma árvore ou de um grafo acíclico. O versionamento é abordado com mais profundidade em (GOLENDZINER, 1995).

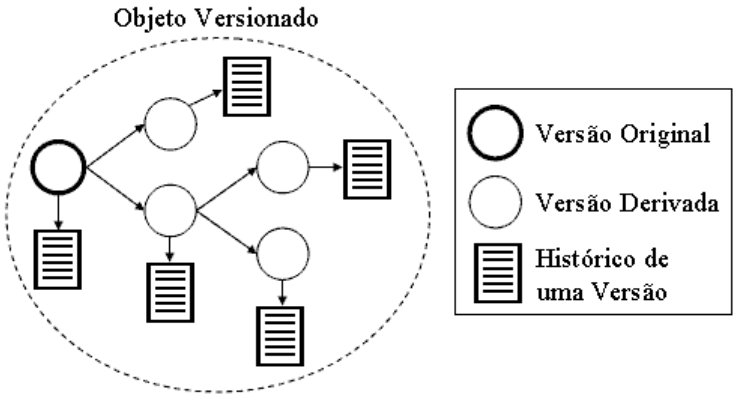


Figura 2.2: Versionamento

2.3 Modelos de Dados Temporais

Muitos são os modelos temporais existentes que estendem modelos tradicionais de dados. Como extensões do modelo relacional, têm-se, por exemplo, o *Historical Relational Data Model* (CLIFFORD; CROKER, 1987), o *Historical SQL* (SARDA, 1990) e o *Temporal Relational Model* (NAVATHE; AHMED, 1989); para o modelo entidade-relacionamento, existem, entre outros, o *Entity Relationship Time Model* (THEODOULIDIS; LOUCOPOULOS; WANGLER, 1991), o *Temporal Entity-Relationship Model* (TAUZOVICH, 1991), o *Temporal Enhanced Entity-Relationship Model* (ELMASRI; WUU; KOURAMAJIAN, 1993) e o *TempER* (ANTUNES, 1997); por fim, como exemplos de modelos orientados a objetos temporais, pode-se citar o *T-CHIMERA* (BERTINO; FERRARI; GUERRINI, 1996), o *TF-ORM* (EDELWEISS; OLIVEIRA; PERNICI, 1993), o *TIGUKAT* (ÖZSU et al, 1993) e o *TVM* (MORO, 2001). O foco desta dissertação, contudo, está nos documentos XML, que caem noutra categoria: dados semi-estruturados. Nesta seção, serão apresentadas propostas que buscam oferecer aos modelos de dados XML o mesmo suporte temporal já existente para os modelos tradicionais.

Há duas abordagens principais que podem ser empregadas para o armazenamento de registros históricos de documentos semi-estruturados que evoluem linearmente. A mais simples é conhecida como *snapshot collection* (CHAWATHE; ABITEBOUL; WIDOW, 1998), e consiste em armazenar integralmente todos os estados assumidos pelo documento. Em outras palavras, armazena-se o estado original do documento e, quando o mesmo sofre uma modificação, é gerada uma nova cópia de todo o documento, idêntica à cópia anterior com exceção da modificação promovida – isto é, a cópia contém tanto o fragmento do documento que foi alterado, quanto os trechos que permaneceram os mesmos. Cada alteração gera, portanto, uma réplica parcial do documento. A essas diferentes cópias, representativas de diferentes estados do mesmo documento, podem ser associados rótulos temporais indicando o período de validade de cada estado. Esta abordagem otimiza o tempo de recuperação de um estado qualquer, pois nenhum estado precisa ser reconstruído, já que todos são armazenadas em sua totalidade, mas justamente por isso apresenta um alto custo em espaço de armazenamento.

Para mostrar a segunda abordagem, é necessário antes apresentar o conceito de *scripts* de conversão. Estes *scripts* (também conhecidos como *deltas*) são listas de operações que, quando aplicadas a um estado de um documento, transformam-no gerando um novo estado. Operações deste tipo incluem, mas não se limitam à inclusão e exclusão do conteúdo de sub-árvores em um documento semi-estruturado. É fácil ver que essas duas operações são suficientes para transformar qualquer documento em qualquer outro, bastando para isso aplicar a operação de exclusão para apagar completamente o documento original, e em seguida inserir todo o conteúdo do documento modificado. Muitas vezes, contudo, utiliza-se um conjunto mais amplo de operações (incluindo, por exemplo, atualização do conteúdo de uma sub-árvore ou movimentação de uma sub-árvore para outra posição dentro do documento). Quanto mais operações forem definidas, maior a semântica dada a um *script* (pois cada operação possui uma interpretação particular) e menores os *scripts* se tornam (um par de operações de inclusão e exclusão, por exemplo, pode ser substituído por uma única operação de modificação). Por fim, os *scripts* podem ser de três tipos: progressivos, quando permitem obter um novo estado a partir de um estado antigo; regressivos, quando permitem voltar de um estado mais recente a um estado anterior; ou completos, quando incluem informação suficiente para servirem tanto como *scripts* progressivos quanto como regressivos.

A segunda abordagem básica, conhecida como *snapshot delta* (CHAWATHE; ABITEBOUL; WIDOW, 1998), consiste em armazenar um único estado – o mais recente, por exemplo – em conjunto com uma coleção de regras de transformação (*scripts* de conversão) para, a partir dele, gerar os demais estados. Esta abordagem otimiza o espaço consumido para armazenar o conjunto de estados, levando em consideração o fato de que, usualmente, um estado cujo conteúdo está armazenado explicitamente ocupa muito mais espaço do que um *script* de conversão – isto depende, é claro, da fração do documento que sofreu alterações. Em casos extremos, um *script* pode ser muito maior do que os estados sobre os quais foi construído, mas esta é uma situação atípica. Em compensação, este método requer uma série de cálculos para reconstruir estados passados, maximizando o tempo de recuperação de estados.

Resumidamente, estas duas propostas representam os extremos do clássico compromisso entre tempo e espaço. A seguir, serão apresentados alguns modelos que buscam atingir um equilíbrio entre estes dois objetivos; muitos destes modelos, como será visto, são fortemente baseados em uma ou em ambas as abordagens.

2.3.1 Modelo TXPath

Em (AMAGASA; YOSHIKAWA; UEMURA, 2000), é apresentado um modelo de dados baseado na extensão temporal do modelo XPath (W3C, 1999). Esta extensão consiste em assinalar rótulos temporais de tempo de validade aos objetos básicos (elementos, atributos e nodos de texto) em um documento XML, de forma a embutir a informação temporal no próprio documento.

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE document SYSTEM "document.dtd">
<document>
 <title>XML</title>
 <authors>
  <author email="taro@db.aist-nara.ac.jp">Taro</author>
  <author>Jiro</author>
 </authors>
 <chapter title="Introduction">
  <section>
   XML stands for eXtensible Markup Language.
  </section>
 </chapter>
</document>
```

Figura 2.3: Documento XML sem rótulos temporais

A Figura 2.3 mostra um documento XML de exemplo, em sua forma atemporal. São apresentadas duas codificações possíveis para incluir rótulos temporais neste documento: uma codificação completa, que inclui uma série de *tags* e atributos especiais para guardar a informação temporal (Figura 2.4), e uma codificação simplificada, que utiliza apenas um atributo *valid* para denotar o tempo de vida de cada elemento (Figura 2.5). A codificação simplificada é mais próxima da forma original do documento, porém é redundante, pois duplica todo o conteúdo de um elemento que sofre alguma alteração – mesmo as partes que se mantiveram inalteradas são duplicadas. Isto não ocorre com a codificação completa, pois esta separa o conteúdo do documento em diversas unidades, cuja evolução é controlada separadamente das demais.

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE document SYSTEM "document.dtd">
<document xmlns:time="http://db-www.aist-nara.ac.jp/time"
          xmlns="http://db-www.aist-nara.ac.jp/document" time:valid="2000-01-01, now">
 <title time:valid="2000-01-01, now">
  <time:stringvalue time:valid="2000-01-01, 2000-01-09">
   XML
  </time:stringvalue>
  <time:stringvalue time:valid="2000-01-10, now">
   Introduction to XML
  </time:stringvalue>
 </title>
 <authors time:valid="2000-01-10, now">
  <author time:valid="2000-01-01, now">
   <time:stringvalue time:valid="2000-01-10, now">
    Taro
   </time:stringvalue>
   <time:attribute name="email" time:valid="2000-01-10, now">
    taro@db.aist-nara.ac.jp
   </time:attribute>
  </author>
  ...
```

Figura 2.4: Fragmento da codificação completa usando rótulos temporais

Para recuperar o conteúdo do documento em um dado ponto no tempo, basta considerar apenas os elementos cujo atributo *valid* contenha o momento desejado. Com a codificação completa, é necessário convertê-los novamente para o formato original; já com a codificação simplificada, que armazena o documento de forma praticamente idêntica à codificação atemporal, basta remover o atributo *valid* de todos os elementos

selecionados. Um mapeamento do modelo TXPath para um banco de dados relacional temporal é apresentado em (AMAGASA; YOSHIKAWA; UEMURA, 2001).

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE document SYSTEM "document.dtd">
<document xmlns:time="http://db-www.aist-nara.ac.jp/Time"
          xmlns="http://db-www.aist-nara.ac.jp/Document" time:valid="2000-01-01, now">
 <title time:valid="2000-01-01, 2000-01-09">XML</title>
 <title time:valid="2000-01-10, now">Introduction to XML</title>
 <authors time:valid="2000-01-01, now">
  <author email="taro@db.aist-nara.ac.jp" time:valid="2000-01-01, now">
   Taro
  </author>
  <author time:valid="2000-01-01, 2000-01-14">Jiro</author>
  <author email="jiro@db.aist-nara.ac.jp" time:valid="2000-01-15, now">
   Jiro
  </author>
 </authors>
 ...
```

Figura 2.5: Fragmento da codificação simplificada usando rótulos temporais

2.3.2 *Usefulness-Based Change Control*

Esquemas baseados em *scripts* de transformação de estados possuem a desvantagem de poder recuperar fragmentos de dados que não são mais válidos durante o processo de reconstrução de um estado. Esquemas dessa natureza também não preservam a estrutura lógica do documento semi-estruturado original – pode ser necessário reconstruir um estado inteiro para recuperar um campo específico.

O esquema proposto em (CHIEN; TSOTRAS; ZANIOLO, 2001-b), o UBCC (*Usefulness-Based Change Control*), busca reduzir os custos de entrada e saída na recuperação de estados através do agrupamento físico de todos os objetos válidos de um dado estado em algumas poucas páginas de dados. Dessa forma, é necessário realizar menos operações de entrada e saída, pois cada página carrega uma grande porção de informação relativa ao estado desejado. Os autores batizaram esta técnica de *page-usefulness*, ou "utilidade de página". Quando, para um dado estado, o número de objetos válidos em uma página cai abaixo de um certo limiar, os objetos ainda válidos são copiados para uma nova página. Quanto maior esse limite, melhor o desempenho obtido, mas também será maior o espaço ocupado em disco. Na reconstrução de um estado, são acessadas apenas as páginas "úteis" para esse estado, ou seja, aquelas com um alto *page-usefulness* para aquele estado específico.

O UBCC possui duas variantes: o UBCC baseado em *scripts* de edição (*Edit-based UBCC*) e o UBCC baseado em identificação e cópia de segmentos comuns (*Copy-based UBCC*). Para ambos, a reconstrução de um estado começa do próprio estado que se quer reconstruir, seguindo recursivamente para os estados aos quais ele faz referência.

2.3.2.1 Edit-based UBCC

Na introdução da seção 2.3, foram brevemente mencionadas duas abordagens elementares para a representação da evolução de documentos: o armazenamento completo de todos os estados (*snapshot collection*) e o armazenamento de um único estado, bem como de um conjunto de regras para a partir deste obter os demais estados (*snapshot delta*). A escolha de uma ou de outra depende da variação entre dois estados consecutivos: em situações onde os estados variam muito, o primeiro método é o mais apropriado; já quando a variabilidade é pequena, o segundo tende a apresentar melhor desempenho.

O UBCC foi projetado para funcionar como um híbrido dos dois métodos, ora funcionando como um, ora como outro, de forma a se adaptar às características dos

documentos temporais. Para isso, se vale do conceito de *page-usefulness* apresentado anteriormente. Além de uma relação de quais páginas são consideradas úteis para cada estado, são armazenados também, para cada par de estados consecutivos, os *scripts* de transformação de um para outro, para que se possa reconstruir a ordenação correta dos fragmentos de dados. Estes *scripts* consistem apenas de operações de inserção e remoção. No caso de replicação de fragmentos, os *scripts* são modificados para incluir operações de inserção desses fragmentos.

O procedimento para recuperar um estado qualquer E_i é como segue. Inicialmente, recupera-se o *script* de transição de E_{i-1} para E_i, e suas operações são analisadas em ordem. Supondo que a primeira operação seja *inserir(OBJ, 4)*, significa que o objeto *OBJ* deverá constar no resultado na posição 4. O objeto *OBJ* será buscado nas páginas que precisaram ser alocadas na criação de E_i, porém antes é necessário preencher as posições 1 a 3. Como o *script* é processado em ordem, significa que estas posições mantiveram-se inalteradas em relação ao estado E_{i-1}. Logo, o processo segue recursivamente para reconstruir o trecho das posições 1 a 3 no estado E_{i-1}, buscando informações nas páginas que são úteis tanto para E_i quanto para E_{i-1}. Para reconstruir o trecho requerido de E_{i-1}, pode ser necessário acessar E_{i-2}, e assim sucessivamente.

2.3.2.2 Copy-based UBCC

O esquema baseado em cópia trabalha com listas de referências para reaproveitar segmentos comuns entre os estados novos e antigos. Em relação ao estado anterior, atingem-se otimizações em situações de movimentação e repetição de fragmentos, pois no esquema *edit-based* tais fragmentos seriam replicados no novo estado, enquanto que no *copy-based* seriam representados através de referências para os fragmentos equivalentes no estado antigo. Usualmente referências ocupam menos espaço em memória do que blocos de dados. Com esse esquema, não é mais necessário armazenar os *scripts*, mas sim uma relação, para cada estado, de quais páginas contêm dados concretos e quais contêm referências a outros estados.

A reconstrução de estados se dá de forma similar ao *Edit-based UBCC*: para reconstruir o estado E_i, recuperam-se as páginas que contém objetos e referências de E_i, e os blocos são processados em ordem. Ao se encontrar uma referência ao estado E_{i-1}, é feita uma chamada recursiva para recuperar aquele trecho específico de E_{i-1}, que por sua vez pode conter referências a E_{i-2}, e assim por diante.

2.3.2.3 Análise de Desempenho

Em (CHIEN; TSOTRAS; ZANIOLO, 2001-b) e (CHIEN; TSOTRAS; ZANIOLO, 2002), estão incluídos experimentos que comparam o desempenho das duas variantes do UBCC entre si e contra as abordagens elementares *snapshot collection* e *snapshot delta*. Como era de se esperar, armazenar todos os estados propicia o menor custo de recuperação de estados e o maior custo de armazenamento, e o armazenamento de *scripts* se comporta da maneira inversa.

Ambas as variantes do UBCC apresentam desempenho e custo de armazenamento intermediários em relação aos extremos e muito similares quando comparados um ao outro. O *Copy-based* UBCC apresenta maior irregularidade nas suas curvas, ora superando e ora perdendo para o *Edit-based* UBCC, porém oferece maior generalidade e flexibilidade para a escrita de consultas, com praticamente o mesmo desempenho.

2.3.3 *Reference-Based Version Model*

O *Reference-Based Version Model* (RBVM) é proposto em (CHIEN; TSOTRAS; ZANIOLO, 2001-a), pelos mesmos autores do artigo que propõe o UBCC, e apresenta para com este muitas similaridades. O RBVM apresenta as seguintes propriedades:

- preserva a estrutura lógica do documento, permitindo recuperação eficiente de estados e consultas baseadas em conteúdo;
- a história de evolução de um documento XML pode ser representada como outro documento XML;
- utiliza técnicas para reduzir o custo de armazenamento da informação, como esquemas baseados em *page-usefulness*.

Ao contrário do UBCC, entretanto, que se trata de um esquema genérico para dados semi-estruturados, o RBVM é um estudo específico sobre árvores XML. Além dos elementos típicos de XML, uma árvore RBVM inclui também nodos de *estado* (que servem como raízes das sub-árvores que identificam cada um dos estados de um documento) e nodos de *referência* (que apontam para sub-árvores em comum entre dois estados distintos). Para implementar as referências, cada nodo da árvore XML recebe um identificador único, assim como ocorre com outras técnicas estudadas.

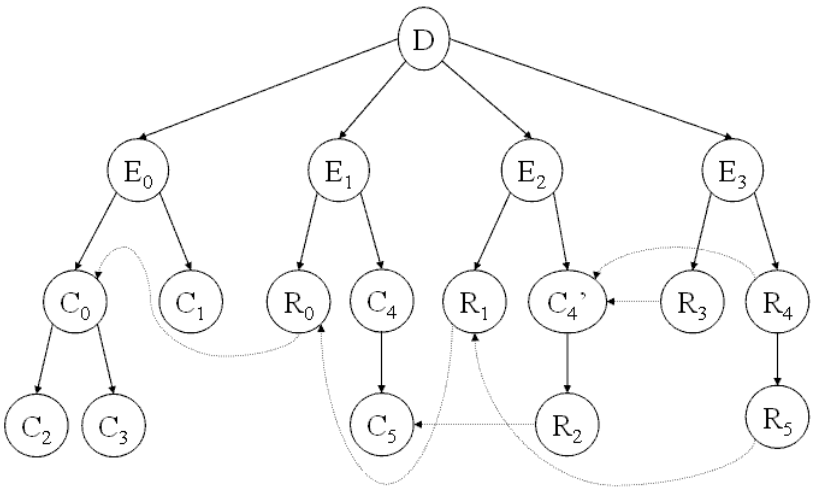


Figura 2.6: Estrutura XML criada pelo RBVM

O primeiro estado é sempre integralmente materializado. A partir do segundo estado, sub-árvores que não são alteradas são mantidas através de nodos de referência, sub-árvores que são excluídas simplesmente não são referenciadas e sub-árvores que são incluídas são materializadas no novo estado. A cópia de sub-árvores pode ser simulada pela existência de várias referências para um mesmo nodo, e a atualização de conteúdo corresponde a uma combinação de remoção e inserção. Nodos de referência sempre apontam de um estado para seu antecessor imediato, de modo que podem apontar para outra referência e assim por diante caso se trate de uma sub-árvore que permaneceu inalterada por diversos estados.

Essa estrutura pode ser vista na Figura 2.6. O símbolo *D* representa a raiz do documento, que possui diversos estados *E*. No estado E_0, todos os nodos de conteúdo *C* (elementos, atributos e texto) estão materializados. No estado E_1, a sub-árvore com raiz em C_0 do estado anterior foi mantida, através de um nodo de referência. Já a sub-árvore com raiz em C_1 foi excluída, pois não há referência que aponte para ela. Ocorreu

também a inclusão de uma nova sub-árvore (nodos C_4 e C_5), que aparece no documento XML explicitamente. Do estado E_1 para E_2, a sub-árvore de C_0 foi novamente mantida; note que a referência R_1 aponta para outra referência, e não para o conteúdo original. Uma alteração no conteúdo de C_4, criando C_4', é feita através de nova materialização do conteúdo, sem se referir ao valor antigo. Finalmente, no estado E_3, é feita uma cópia da sub-árvore de C_4', através de uma segunda referência, e a movimentação da sub-árvore C_0 para outro ponto na árvore é feita também através de uma referência.

Assim como no UBCC, a reconstrução de um estado começa a partir do próprio estado que se quer reconstruir, recuperando a informação que ali se encontra materializada e seguindo recursivamente através das referências para os estados anteriores. O RBVM utiliza também uma metodologia de agrupamento de dados que lembra o *page-usefulness* do UBCC.

Experimentos similares aos do UBCC foram realizados para o RBVM, incluindo também os dois casos básicos (*snapshot collection* e *snapshot delta*), que se comportaram da mesma forma, e outros dois esquemas para o gerenciamento de estados: árvores-B multiversionadas e listas parcialmente persistentes. Os três esquemas mais elaborados apresentaram, novamente, desempenhos intermediários em comparação aos extremos delineados pelos casos básicos, porém o RBVM foi o que apresentou melhor desempenho médio para recuperação e menor demanda de espaço.

2.3.4 Método SPaR

Apresentado em (CHIEN et al, 2001), o método SPaR (*Sparse Preorder and Range*) é de funcionamento semelhante ao UBCC e ao RBVM, trabalhando também exclusivamente com documentos XML. Neste método, contudo, os modelos *edit-based* e *reference-based* são substituídos por um esquema de numeração persistente, responsável por atribuir identificadores (*Durable Node Numbers*, ou simplesmente *DNN*'s) a cada objeto da árvore XML. O nome do método vem da filosofia de atribuição de identificadores empregada, a qual associa a cada objeto um intervalo numérico esparso do qual são retirados os identificadores de seus descendentes, atribuídos segundo uma travessia em pré-ordem da árvore XML (daí a expressão "*Sparse Preorder and Range*").

Ao contrário do que ocorre em outras propostas que também atribuem identificadores aos nodos da árvore, como, por exemplo, o Xyleme (MARIAN et al, 2001), neste método a operação de exclusão de um objeto pode liberar seu DNN, juntamente com todo o intervalo associado, para ser usado em outro objeto. Além dos limites do intervalo, a cada objeto é associado também um par de rótulos temporais que indicam seu intervalo de vida. Ao contrário do UBCC e do RBVM, não é armazenado qualquer tipo de *script* ou de nodo de referência a estados anteriores; a estrutura da árvore pode ser reconstruída tendo-se apenas os DNN's e os intervalos de cada objeto.

Os elementos do documento são armazenados em páginas por ordem crescente de seu DNN – isso é útil na hora de percorrer as páginas para reconstruir a árvore. A idéia do *usefulness-based clustering* é aplicada sobre estas páginas também. A relação entre os estados e suas respectivas páginas úteis é registrada em um índice. Experimentos comparando este método com o UBCC e com os métodos básicos mostraram que SPaR e UBCC ficam mais uma vez situados entre os extremos, porém o SPaR é mais eficiente tanto no tempo de recuperação de informações quanto no espaço requerido. Um estudo sobre alternativas de implementação do método SPaR pode ser encontrado em (CHIEN et al, 2002).

2.3.5 Linguagem TXML

Proposta em (MANUKYAN; KALINICHENKO, 2001), esta é uma linguagem com um forte embasamento matemático. Batizada de TXML, a linguagem é baseada em cálculo lambda, e permite a temporalização de todas as estruturas XML: elementos, atributos, entidades, DTD, etc., podendo envolver tanto tempo de transação quanto tempo de validade. Todas estas estruturas são codificadas como expressões em cálculo lambda, e o documento é portanto visto como uma expressão matemática, chegando inclusive a utilizar adaptações de codificações XML desenvolvidas para o armazenamento de expressões deste tipo, como OPENMath (CAPROTTI; CARLISE; COHEN, 1999).

A notação supõe o acesso da aplicação TXML a arquivos externos chamados *Dicionários de Conteúdo*, que contém o significado das diferentes estruturas utilizadas na representação da informação temporal. Apesar de seu alto poder de expressão, em comparação com as outras abordagens mostradas neste capítulo, esta é altamente complexa e produz codificações extensas mesmo para pequenos repositórios de dados.

2.3.6 Método Xyleme

O tópico desta subseção é um sistema de gerenciamento de documentos XML em evolução que faz parte do projeto Xyleme (MARIAN et al, 2001) de estudo e implementação de bancos de dados XML dinâmicos. Na arquitetura do Xyleme (Figura 2.7), novos estados de documentos são adquiridos da Web e comparados com os estados mais recentes na base. Os dois servem de entrada a um algoritmo de detecção de diferenças denominado *XyDiff* (COBÉNA; ABITEBOUL; MARIAN, 2002), que computa um *script* a ser incluído à base de dados – representado na Figura 2.7 por $\Delta(E_n, E_{n-1})$. O estado mais recente é atualizado com aquele buscado na Web, e o estado anterior é removido. O sistema trabalha com milhões de documentos por dia, logo eficiência é um fator chave, bem como baixo consumo de memória.

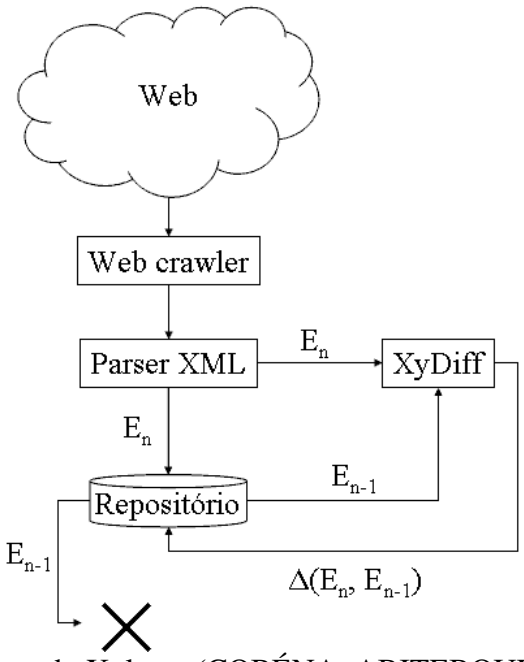


Figura 2.7: Arquitetura do Xyleme (COBÉNA; ABITEBOUL; MARIAN, 2002)

A representação lógica das diferenças é baseada em *deltas* com as seguintes características:

- identificadores persistentes para cada nodo XML, aqui denominados de XID's (Xyleme ID's), assinalados conforme os resultados do algoritmo de cálculo de diferenças. XID's permitem registrar modificações nos dados através de seus identificadores, ao invés de utilizar sua posição na árvore de elementos, como é feito em outros sistemas, como o UBCC;
- utiliza *deltas* completos ao invés de *deltas* simples. *Deltas* completos possuem as facilidades adicionais de permitirem a navegação nos dois sentidos (ou seja, permitem obter o estado mais novo com base no mais antigo e vice-versa – isto também é conhecido como inversão de *deltas*) e da possibilidade de composição das suas operações. Estas operações podem também ser implementadas em sistemas baseados em *deltas* simples, mas nestes podem incorrer em custosas reconstruções de estados.

A política de armazenamento físico empregada consiste no armazenamento do estado corrente, de uma filosofia de atribuição de identificadores inteiros denominada *XID-map* e de um único arquivo XML contendo todos os *forward completed deltas* (descrevem as alterações do estado antigo para o novo, mas também podem ser utilizados para o sentido inverso). Com o XID-map pode-se acessar facilmente, através do XID, todo o histórico de evolução de um dado nodo. Uma vantagem desta política de armazenamento é que a inclusão de um novo estado praticamente não incorre em modificações dos objetos já armazenados; uma desvantagem é que *deltas* completos implicam em informação redundante. Isto pode ser contornado através de técnicas de compressão do arquivo de *deltas*, como identificação de grandes fragmentos que se repetem e substituição destes por ponteiros.

Como se pode perceber, há uma grande similaridade com o método *snapshot delta*. O Xyleme armazena também o estado mais recente e um conjunto de *deltas*, mas ao contrário do outro método, utiliza *deltas* completos ao invés de deltas simples, devido às facilidades de inversão e composição de *deltas* já mencionadas, as quais possuem grande importância no contexto do projeto Xyleme. As prioridades do sistema são inserir novos estados e computar composição de *deltas* de forma eficiente; recuperar estados antigos na íntegra, no contexto deste projeto, não possui muita importância, logo se pode sacrificar um pouco de eficiência neste ponto para favorecer outros.

A representação utilizada para se visualizar os documentos e suas alterações é a de uma árvore XML. A transformação de um estado em outro é vista como a aplicação de operações básicas de modificação sobre o conteúdo dessa árvore, as quais incluem, tipicamente, inserção, remoção, atualização e movimentação de dados (são possíveis também operações mais especializadas, como renomeação de elementos, incremento de valores inteiros, etc.).

O XID-map assinala identificadores inteiros seguindo a ordem de uma travessia pré-fixada nos elementos da árvore XML. Não é necessário atribuir XID's a atributos, pois pela própria definição de atributos em XML estes são únicos e não-ordenados dentro de um elemento; logo o XID do elemento pai junto ao nome do atributo serve para identificá-lo. Seria possível utilizar o próprio conceito de ID's em XML, mas estes não garantiriam a persistência que se obtém com XID's.

2.3.7 Método de Wong & Lam

Em (WONG; LAM, 2003), Wong e Lam apresentam uma técnica de armazenamento que combina características de muitas das propostas já vistas neste capítulo. O resultado é muito similar ao Xyleme, da seção 2.3.6, e em muitos aspectos pode ser visto como

uma evolução do mesmo. Resumidamente, o sistema proposto utiliza identificadores persistentes para referenciar os nodos XML, e armazena a informação através de um conjunto de *deltas* completos e de *diversos* estados integralmente materializados. Isto significa que, ao contrário do Xyleme, não é armazenado apenas um único estado, que serve como ponto de partida para toda e qualquer consulta; há diversos potenciais pontos de partida, e recuperar um estado se traduz em identificar o ponto de partida mais "próximo" (em termos de um custo que será definido a seguir) e utilizar *forwards* ou *backwards deltas* (dependendo se o ponto de partida é um estado anterior ou posterior ao que se quer recuperar) para calcular o resultado.

A intuição por trás do armazenamento integral de diversos estados vem da seguinte situação: imagine, no sistema Xyleme, um documento para o qual existem centenas ou até milhares de estados. Para recuperar um dos estados iniciais do documento, seria necessário partir do mais recente (que é o único ponto de partida possível, pois é o único estado armazenado explicitamente) e retroceder até o estado desejado, computando centenas ou milhares de *deltas* reversos. Naturalmente, seria mais eficiente poder começar de um estado que pertença ao início do sistema, dessa forma menos operações serão executadas para transformá-lo no estado que se quer obter. Observe que modificar o Xyleme para armazenar o primeiro estado ao invés do último solucionaria este problema em particular, mas criaria outro equivalente quando se quisesse recuperar estados mais recentes. Logo, a utilização de múltiplos pontos de partida é a solução óbvia. Em algumas situações, essa abordagem pode ser vantajosa também em termos de espaço consumido: caso um estado mude muito em relação ao seu anterior, o *script* necessário para descrever as alterações pode ocupar mais espaço do que simplesmente armazenar o novo estado em sua totalidade. Nesse caso, poder optar por não armazenar o *script*, mas sim o estado, também representaria uma vantagem.

As operações suportadas neste modelo são: inserção, remoção, atualização, movimentação e cópia. As três últimas poderiam ser descartadas sem perda de poder de representação, porém sua inclusão provê mais significado aos *scripts* de transformação de estados. O conhecimento do conjunto de operações de alteração pode vir do usuário, através do fornecimento direto do *script*, ou pode ser inferido pelo sistema, através de um algoritmo de detecção de diferenças, como o *X-Diff* (WANG; DEWITT; Cai, 2003) ou o *xDiff* (WONG; LAM, 2002). O custo de um *script* é seu tamanho em número de operações. Durante a recuperação de estados, varre-se uma estrutura de dados que contém, para cada estado armazenado, um *flag* que informa se está materializado integralmente e o tamanho do *script* que se deve aplicar sobre seu antecessor imediato para obtê-lo. O estado mais próximo do que se quer recuperar é aquele que incorre no menor número de operações.

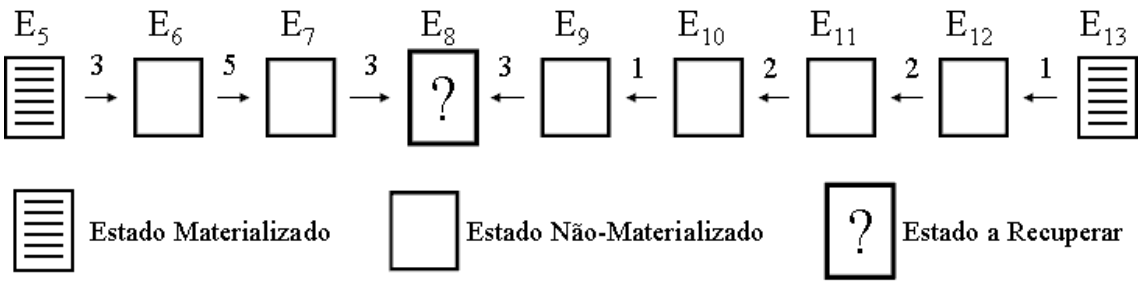


Figura 2.8: Exemplo de Reconstrução de Estados

Como exemplo, observe a situação ilustrada na Figura 2.8. Digamos que se queira recuperar o estado de número 8, tendo-se que os estados armazenados integralmente mais próximos são os de número 5 e 13. Intuitivamente, poder-se-ia pensar que o estado

de número 5 está mais próximo, porém isso não é necessariamente verdade. Na figura, o número de operações necessárias para ir do estado *i* para o estado *j* está indicado acima da seta entre os estados, que por sua vez indica o sentido da transformação. Dessa forma, ir do quinto para o sexto estado, por exemplo, requer três operações, enquanto que para ir do 13º. para o 12°., basta uma operação. Assim sendo, partindo do número 5 e avançando em direção ao número 8, seriam aplicadas 3 + 5 + 3 = 11 operações no total, enquanto que regredindo a partir do 13º. seriam necessárias apenas 1 + 2 + 2 + 1 + 3 = 9 operações.

Na verdade, o exemplo acima é simplista: existem mais parâmetros envolvidos nesta estimativa de custos. Para aplicar os *deltas* regressivos, por exemplo, é necessário invertê-los antes. Ademais, no lugar de numerar seqüencialmente os estados, como é mostrado no exemplo, na verdade cada estado possui um *timestamp* de tempo de transação, indicando o momento em que foi incluído na base de dados. Porém, a idéia por trás do exemplo se mantém: analisa-se os "vizinhos" à esquerda e à direita do estado que se quer recuperar, estima-se os custos de cada um deles e escolhe-se o menor.

Foi mencionado que, ao se inserir um novo estado, pode-se optar por armazenar apenas o *delta* em relação ao último ou, se for mais eficiente em termos de espaço, armazenar o próprio estado. Mesmo que este não seja o caso, há outro fator que pode decidir a favor de se armazenar um estado em sua totalidade – uma estimativa de custos futuros. O método proposto utiliza regressão não-linear para, com base em estatísticas sobre os estados anteriores (tamanhos de estados, tamanhos de *deltas* e taxa de modificações de um estado para outro), estimar se, no futuro, provavelmente seria mais vantajoso recuperar diretamente o estado que está sendo inserido, ou através de *deltas* progressivos a partir de um estado mais antigo ou ainda através de *deltas* reversos a partir de outro estado que ainda não existe no sistema. Dependendo do resultado desta estimativa, armazena-se um *delta* em relação ao anterior ou o estado como um todo.

Para a realização de consultas, utiliza-se a linguagem XQuery, com o auxílio de funções definidas pelo usuário (*user-defined functions*, ou simplesmente *UDF*'s) para implementar as operações temporais. São propostas duas estruturas de índices para agilizar consultas: uma tabela *hash* cujas chaves são as *tags* do documento XML e um índice de caminhos sob a forma de árvore, com correspondência 1-pra-1 entre os nodos da árvore índice e os nodos da árvore XML. Há também um índice de *tags* estendido que indexa cada palavra das *strings*, para melhor suportar consultas ao conteúdo das *tags*. Todos os índices são armazenados também como documentos XML.

2.4 Considerações Finais

Este capítulo revisou conceitos e idéias fundamentais à compreensão do modelo TVX, proposto neste trabalho. O conteúdo apresentado pode ser dividida em dois grandes tópicos: a revisão dos diversos conceitos de representação temporal e de versões e a apresentação de propostas existentes para modelos de dados temporais orientados a documentos semi-estruturados – em sua maioria, mais especificamente, a documentos XML.

A Tabela 2.3 mostra, resumidamente, uma comparação entre os diversos modelos apresentados na seção 2.3. Para cada modelo, é apresentado que tipo de rótulos temporais utiliza – se utiliza tempo de transação (coluna TT) e/ou tempo de validade (coluna TV), e de que tipo é o rótulo utilizado. Note que apenas a notação TXML utiliza tanto tempo de transação quanto tempo de validade, e que ambas as versões do UBCC, bem como o RBVM, não utilizam qualquer tipo de rótulo temporal, simplesmente numerando seqüencialmente os diversos estados assumidos pelos documentos.

A Tabela 2.3 também mostra que nenhuma das propostas estudadas utiliza o conceito de versionamento apresentado na seção 2.2 (embora utilizem o termo "versões" para se referir aos diversos estados de um documento). A coluna "Foco" separa os modelos entre focados em dados e focados em operações – aqueles focados em dados registram a história armazenando os diversos estados assumidos pelo documento, enquanto que aqueles focados em operações optam por armazenar a história através do registro das operações que transformam um estado em outro. As demais colunas mostram se o modelo possui um protótipo implementado e se é um estudo específico para XML, podendo tirar proveito de características próprias da linguagem, ou se é orientado e documentos semi-estruturados genéricos.

Tabela 2.3: Comparação entre os modelos estudados

Modelo	Rótulos			Versões	Foco	Protótipo	Orientado a XML
	TT	TV	Tipo				
TXPath	Não	Sim	Intervalo Temporal	Não	Dados	Não	Sim
Edit-Based UBCC	Não	Não	-	Não	Operações	Sim	Não
Copy-Based UBCC	Não	Não	-	Não	Dados	Sim	Não
RBVM	Não	Não	-	Não	Dados	Sim	Sim
SPaR	Não	Sim	Intervalo Temporal	Não	Dados	Sim	Sim
TXML	Sim	Sim	Intervalo Temporal	Não	Dados	Não	Sim
Xyleme	Sim	Não	Ponto Temporal	Não	Operações	Sim	Sim
Método de Wong & Lam	Sim	Não	Ponto Temporal	Não	Operações	Sim	Sim

3 O MODELO TVX

Este capítulo descreve o modelo TVX (Tempo e Versões em XML), cujo objetivo é oferecer mecanismos de gerenciamento para a evolução do conteúdo de documentos XML, unificando as dimensões de tempo de validade, tempo de transação e de criação de novas versões. O uso combinado destes conceitos, tratados separadamente nos trabalhos apresentados no capítulo anterior, provê grande flexibilidade no tratamento da evolução destes documentos. Apesar da importância da evolução de esquemas ser inegável, a mesma não será tratada no presente trabalho; é uma das metas, contudo, estabelecer uma base para a evolução do conteúdo, de forma que o modelo possa ser estendido em trabalhos futuros para incorporar evolução estrutural.

O capítulo é organizado da seguinte forma: na seção 3.1, será apresentada uma definição formal de um documento XML de acordo com o escopo deste trabalho. Nas seções 3.2 e 3.3, esta definição será estendida para incorporar rótulos temporais e informações de versionamento diretamente no documento XML. Na seção 3.4 será demonstrado através de um diagrama de classes de alto nível como estas extensões afetam os objetos XML e, na seção 3.5, como podem ser implementadas nesse formato. Por fim, a seção 3.6 mostra as etapas necessárias para transformar um documento no formato proposto para um formato não temporal, de maneira a extrair seu conteúdo em um dado instante.

3.1 Conceito de Documentos XML

A definição de documento XML que será usada é uma simplificação da proposta original que exclui os conceitos de *namespaces*, instruções de processamento e comentários; da mesma maneira, não será dado qualquer tratamento especial para atributos dos tipos *ID*, *IDREF* e *IDREFS*. Essas construções foram excluídas para simplificar o modelo e focalizá-lo no conteúdo dos documentos; deve-se ressaltar, contudo, que o modelo pode ser estendido para incluir tais características. Ou seja, os objetos da linguagem XML que serão considerados são: elementos, atributos e nodos de texto.

Mais formalmente, considere um documento XML como uma 6-tupla $\langle El, A, T, S, r, Ed \rangle$, onde:

- *El* é um conjunto de elementos, que devem apresentar um nome;
- *A* é um conjunto de atributos, os quais também possuem nome;
- *T* é um conjunto de nodos de texto;
- *S* é um conjunto de valores do tipo cadeias de caracteres (*strings*);
- *r* é um objeto distinto que aponta para a raiz do documento;

- *Ed* é um conjunto de arcos que conectam os diversos objetos XML de tal forma que o grafo resultante apresente a forma de uma árvore.

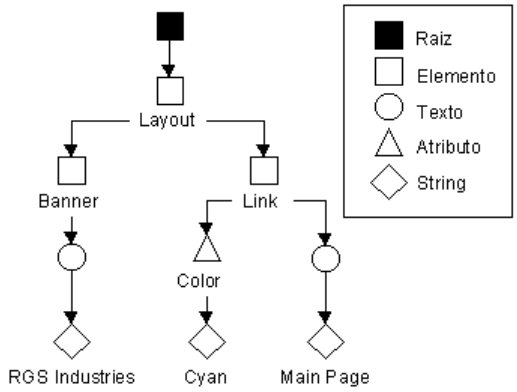
Cada arco é uma 2-tupla $\langle p, c \rangle$, onde p é o nodo pai e c o nodo filho. O elemento raiz r não pode aparecer como filho em nenhum arco, e deve aparecer como pai apenas uma vez. As combinações válidas para nodos pais e filhos são as seguintes:

- $p = r, c \in El$
- $p \in El, c \in El$
- $p \in El, c \in A$
- $p \in El, c \in T$
- $p \in A, c \in S$
- $p \in T, c \in S$

A nomeação de atributos deve obedecer à restrição de que não pode haver dois ou mais atributos filhos do mesmo elemento e com o mesmo nome. Não há restrição para repetição de nomes de atributos filhos de elementos distintos, nem tampouco há qualquer restrição à repetição de nomes de elementos – podendo inclusive haver dois elementos filhos do mesmo pai com nomes idênticos. Por fim, para cada elemento, deve existir uma ordenação total entre seus nodos filhos dos tipos elemento e texto, de tal maneira que dois nodos de texto não apareçam consecutivamente. Os nodos do tipo atributo não apresentam qualquer relação de ordem entre si ou com os demais nodos.

```
<Layout>
 <Banner>
  RGS Industries
 </Banner>
 <Link Color="Cyan">
  Main Page
 </Link>
</Layout>
```

(a)



(b)

Figura 3.1: Documento XML em forma de texto (a) e em forma de árvore (b)

A Figura 3.1a apresenta um pequeno documento XML de exemplo que corresponde à codificação de um *layout* de uma página da Web. A árvore vista na Figura 3.1b, por sua vez, é uma representação gráfica do mesmo documento – baseada na representação utilizada em (AMAGASA; YOSHIKAWA; UEMURA, 2000) – de acordo com a definição de documento XML apresentada nesta seção. Em um primeiro momento, pode-se ter a impressão de que os nodos *string* sejam desnecessários, pois é possível, sem perda de informação, incorporar o valor que contêm diretamente ao nodo de texto ou atributo ao qual estão subjugados. De fato, é a abordagem predominante nas propostas estudadas, porém há uma razão para manter esses nodos separados, a qual ficará aparente na próxima seção, quando esta estrutura for modificada para incorporar rótulos temporais.

3.2 Modelagem de Rótulos Temporais

A definição de documento XML que foi apresentada será estendida para associar a cada um de seus objetos uma série de rótulos temporais para registrar os tempos de validade inicial e final – delimitando o período no qual os objetos modelam a realidade adequadamente – bem como tempos de transação inicial e final – indicando o período no qual a informação foi registrada na base de dados. Como mencionado na seção 2.1, estes rótulos são necessários para permitir ao usuário retornar a base de dados a estados prévios, através do tempo de transação, e registrar o tempo de vida dos objetos no mundo real, por meio do tempo de validade. Ambos devem estar presentes para permitir uma navegação completa através do histórico evolutivo de um documento.

O modelo proposto é, portanto, bitemporal – todos os objetos receberão rótulos de tempo de transação e de tempo de validade. A extensão para um modelo multitemporal, capaz de manipular também objetos com apenas um destes tipos de rótulos e de mesclá-los com dados estáticos será deixada para trabalhos posteriores. Será utilizada ordenação temporal linear (pois também será incluído o versionamento, o qual pode desempenhar o papel da ordenação ramificada) com variação temporal discreta – como já foi dito na seção 2.1, embora o tempo seja contínuo por natureza, a utilização de tempo discreto simplifica a implementação dos modelos de dados. A granularidade dos *chronons* será deixada a cargo do usuário. Por fim, os rótulos temporais terão a forma de intervalos temporais, por serem mais abrangentes que pontos temporais e de gerenciamento mais simples que elementos temporais.

Além dos rótulos temporais, nodos de elemento e de texto recebem também identificadores globais e persistentes, para indicar a correspondência entre estados diferentes do mesmo objeto ao longo da evolução do documento. Não é necessária a associação de identificadores a nodos de atributo pois, como já foi dito, não há repetição de nomes entre os atributos filhos de um mesmo elemento; assim sendo, a combinação do nome do atributo com o identificador de seu elemento pai é suficiente para identificar univocamente um nodo de atributo qualquer dentro do documento. A necessidade dos identificadores será mais bem explorada na próxima seção, quando for incluído o versionamento.

Neste ponto, a razão para a existência dos nodos *string* pode ser apresentada. Imagine, por exemplo, um elemento *Pessoa* que possui um atributo opcional *Telefone*. O elemento *Pessoa* tem seu próprio tempo de vida, que não necessariamente será igual ao tempo de vida de *Telefone*, porque este é um atributo opcional. Em outras palavras, em um determinado momento o elemento *Pessoa* pode existir sem que haja um atributo *Telefone* a ele associado. De maneira similar, como o número de telefone de uma pessoa pode mudar, cada valor que o atributo *Telefone* assume deve apresentar seu próprio tempo de vida – é possível ter um atributo *Telefone* cujo tempo de vida varia de t_1 a t_3, e que assume dois valores diferentes durante sua existência, um variando de t_1 a t_2 e o outro de t_2 a t_3. Isso significa que deve haver rótulos de tempo de validade separados para nodos de texto e atributo e para seus respectivos valores.

Como exemplo, reconsidere o documento mostrado na Figura 3.1a, e assuma que o mesmo foi criado no dia 01/06/04, sendo válido imediatamente, com exceção do atributo *Color*, que começa a valer apenas no dia 05/06/04 (no decorrer deste capítulo, a granularidade usada para os rótulos temporais será de datas no formato DD/MM/AA). A Figura 3.2 usa este exemplo para mostrar como a estrutura XML anteriormente apresentada pode ser estendida para incorporar tempos de validade e transação, representados como rótulos nos arcos que unem dois objetos, onde os intervalos se referem ao objeto filho daquele arco. Os números contidos nas formas geométricas

representam os identificadores globais para os elemento e nodos de texto. Cada rótulo é uma série de pares $\langle V_i, T_i \rangle$, onde V_i é um intervalo de validade e T_i é o intervalo de transação correspondente. Por exemplo, os rótulos para o elemento *Layout* devem ser interpretados como segue:

- o tempo de transação inicial apresenta o valor 01/06/04, indicando que esta é a data na qual o objeto foi registrado na base de dados. Todos os elementos apresentam o mesmo valor para esta propriedade porque foram todos criados simultaneamente – mesmo o atributo *Color*, que ainda não era válido no momento de sua criação;
- assim que foi inserido, o objeto já foi considerado válido (em outras palavras, já representava adequadamente a realidade modelada), pois o tempo de validade inicial coincide com o tempo de transação inicial. O atributo *Color*, por sua vez, começa a valer apenas no dia 05/06/04, conforme indicado por seu intervalo de tempo de validade;
- como o tempo de validade final está em aberto, o objeto permanece válido até que modificações sejam feitas;
- esta série de rótulos faz parte da visão atual do documento, pois o tempo de transação final está em aberto.

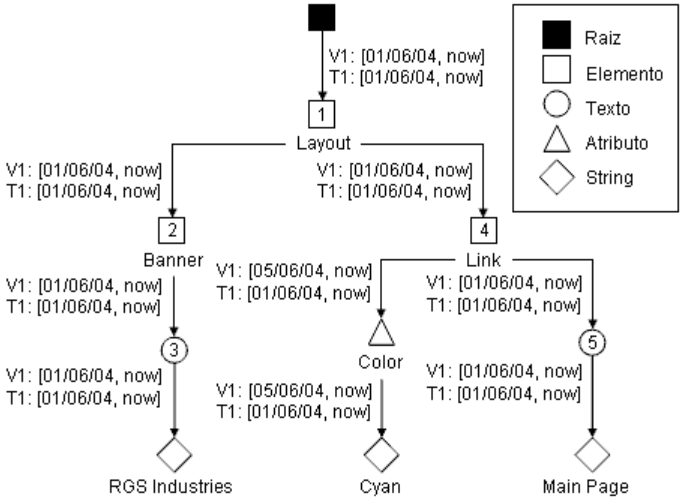


Figura 3.2: Documento original com identificadores e rótulos temporais

Como a estrutura XML está sendo estendida para incluir novas informações, por ora não será possível representá-las em forma textual, tal como foi feito na Figura 3.1a. Uma vez que tenham sido concluídas as extensões ao modelo XML convencional, a seção 3.5 mostrará como este documento estendido pode ser codificado como um documento XML tradicional.

```
<Layout>
 <Banner> RGS Industries
 </Banner>
 <Link Color="Cyan">
  Main Page
 </Link>
</Layout>
```

(a) Documento em 01/06/04

```
<Layout>
 <Link> About Us </Link>
 <Link> Contact </Link>
 <Banner>
  RGS Industries
 </Banner>
</Layout>
```

(b) Documento em 15/06/04

Figura 3.3: Exemplo de estados de um documento XML

Prosseguindo com o exemplo, a Figura 3.3 mostra dois estados diferentes de um mesmo documento (o documento visto em 3.3a é o mesmo visto em 3.1a). Apesar da ordenação dos filhos de um elemento ser relevante, por ora será desconsiderado o fato do elemento *Banner* ter sido movido da posição de primeiro filho do elemento *Layout* para a posição de último filho do mesmo; o mecanismo para modelar esta mudança será explicado mais adiante, na seção 3.5. A Figura 3.3a apresenta o documento no momento de sua criação, em 01/06/04. Mais tarde, em 10/06/04, o documento foi editado para refletir as mudanças mostradas na Figura 3.3b, mas este novo estado só seria válido a partir de 15/06/04. Detalhando as modificações realizadas: o elemento *Link* teve seu atributo *Color* excluído e seu conteúdo alterado de "Main Page" para "About Us"; foi inserido um novo elemento *Link*, contendo o texto "Contact", e o elemento *Banner* foi movido para o fim do documento. A Figura 3.4 mostra como os rótulos temporais podem ser usados para refletir as mudanças aplicadas ao documento de exemplo.

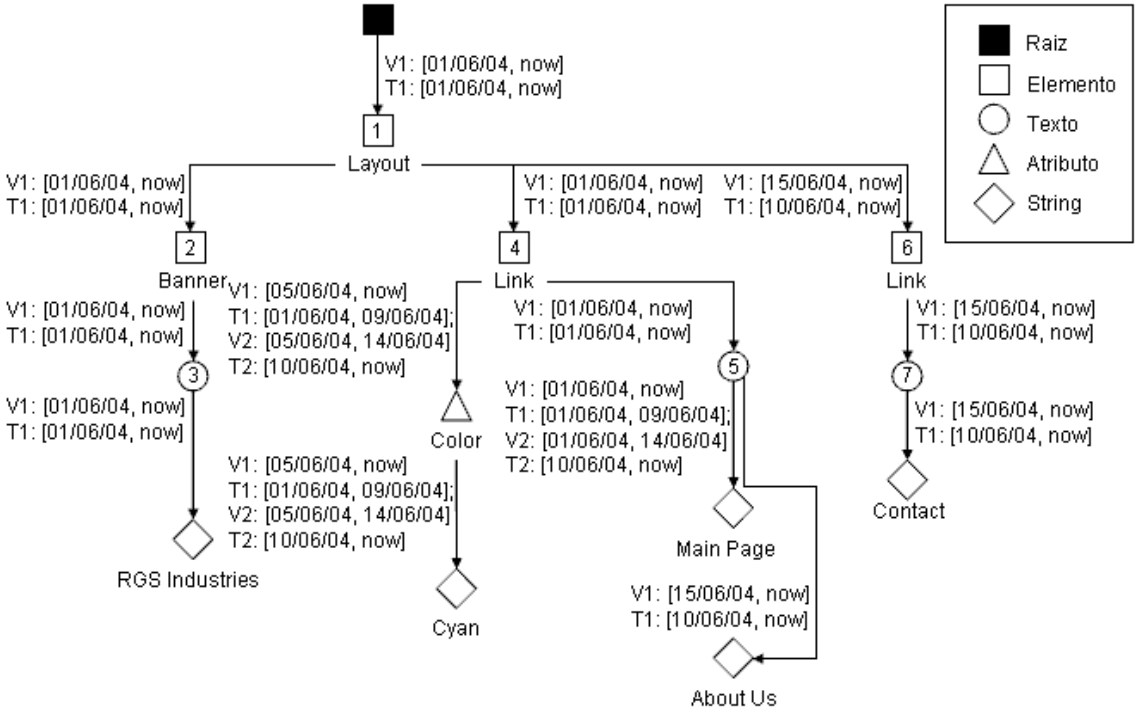


Figura 3.4: Documento modificado com identificadores e rótulos temporais

Conforme mostra a Figura 3.4, o atributo *Color* do primeiro elemento *Link* já não faz mais parte da visão atual do documento. O par prévio de rótulos, $\langle V_1, T_1 \rangle$, representativo do tempo durante o qual se acreditava que este objeto seria indefinidamente válido, é preciso só até o dia 09/06/04 (conforme indicado pelo tempo de transação final em T_1.). A visão corrente, que teve início em 10/06/04, representada pelos rótulos $\langle V_2, T_2 \rangle$, é de que este objeto só foi válido de 05/06/04 a 14/06/04. Da mesma maneira, o valor antigo para o objeto de texto contido neste mesmo elemento *Link* foi substituído por uma nova *string* ("About Us"); esta alteração começa a representar a realidade no dia 15/06/04 (conforme indicado pelo tempo de validade inicial), e já é conhecida pela base de dados no dia 10/06/04 (conforme mostrado pelo tempo de transação inicial). Embora o valor deste objeto de texto tenha sido modificado, o intervalo de validade para o nodo de texto em si permanece o mesmo, pois o fato de seu conteúdo ter sido ou não alterado é irrelevante para o fato de que existe um objeto de texto naquela posição. A presença de nodos do tipo *string* ficou assim clara, pois

deve haver espaço para rótulos que se referem aos nodos de texto e atributos e para outros rótulos que se referem aos valores propriamente ditos.

Note que com o uso conjunto de rótulos de tempo de transação e validade, é possível identificar quatro estados diferentes de conhecimento:

- de 01/06/04 a 04/06/04 – a visão atual do documento é a da Figura 3.3a, com exceção do atributo *Color*, que apesar de já fazer parte da base de dados, ainda não faz parte da visão corrente do documento;
- de 05/06/04 a 09/06/04 – a visão atual do documento é a da Figura 3.3a, e acredita-se que o mesmo permanecerá deste modo;
- de 10/06/04 a 14/06/04 – a visão atual ainda corresponde à Figura 3.3a, mas já é sabido que, a partir de 15/06/04, os valores corretos serão como na Figura 3.3b;
- de 15/06/04 em diante – a visão atual corresponde à Figura 3.3b, porém o conhecimento sobre os estados anteriores não foi perdido.

3.3 Modelagem de Versões

Como motivação para a extensão que será feita para permitir o versionamento, considere, como exemplo, uma empresa que deseja mudar o *layout* de seu *Web site* de acordo com a época do ano (Figura 3.5a). Cada *layout* é armazenado sob a forma de um documento XML, como aquele visto na Figura 3.1a. Há um *layout* principal que é usado durante a maior parte do ano – esta será a versão principal, ou *versão raiz*. Com a chegada do verão, o *layout* muda para um *layout* especial de verão, que é uma versão modificada do *layout* principal, sendo assim uma versão derivada da versão raiz. Durante o verão (para aqueles que vivem no Hemisfério Sul), duas ocasiões especiais acontecem, as quais provocam novas mudanças no *layout*: Natal e Ano Novo; estas serão também novas versões, porém derivadas da versão de Verão. Apesar de qualquer semelhança, cada versão tem seu próprio código que deve evoluir separadamente de outras versões.

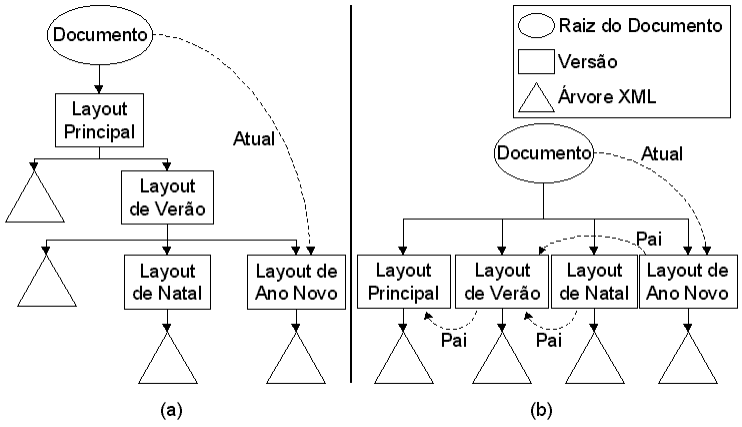


Figura 3.5: Versionamento – estrutura hierárquica (a) e estrutura plana (b)

Além dos rótulos temporais acrescentados para controle de tempo de transação e de tempo de validade, o modelo TVX inclui também a possibilidade de definir versões do documento. A criação de uma versão é determinada pelo usuário, no caso de uma mudança significativa no documento, e é realizada por replicação e modificação dos valores de uma versão existente. A granularidade do versionamento é, portanto, o

documento como um todo. Um documento sempre possui uma versão raiz, da qual podem ser derivadas outras versões. A hierarquia de derivação de versões construída de acordo com estas regras toma a forma de uma árvore: cada versão possui exatamente uma versão pai (com exceção da raiz) e pode possuir uma ou mais versões derivadas, as quais evoluem independentemente umas das outras. A hierarquia em forma de árvore foi escolhida no lugar da hierarquia em forma de grafo porque, apesar de ser mais restrita, encaixa-se naturalmente na estrutura de um documento XML, pelo fato deste ser também estruturado em forma de árvore.

Cada nodo de versão aponta para a raiz de sua própria versão do documento XML; o tempo de vida de uma versão é considerado como sendo idêntico ao tempo de vida do elemento raiz para o qual aponta, pois uma versão não pode existir desvinculada de conteúdo, e vice-versa. Há também a noção de uma versão atual (ou *versão corrente*), a qual simplesmente é o alvo padrão de expressões de atualização e consulta quando nenhuma versão particular é especificada. Isto significa que todas as versões podem ser consultadas e/ou atualizadas a qualquer momento, ao invés de somente a atual.

Os identificadores globais de elementos e nodos de texto, mostrados na seção anterior, indicam a correspondência de fragmentos de informação dentro do histórico de uma versão e entre versões diferentes. Pode haver objetos comuns a muitas versões – por exemplo, um *banner* no topo da página com o nome da empresa. Estes podem ser modelados através de elementos repetidos nas versões envolvidas, todos com o mesmo identificador global. Quando objetos em diferentes versões possuem o mesmo identificador global, significa que representam a mesma entidade no mundo real, mesmo que apresentem conteúdos diferentes. Dois elementos que venham a apresentar o mesmo nome e o mesmo conteúdo, mas com identificadores globais distintos, são considerados objetos diferentes.

A Figura 3.5 mostra duas alternativas para modelar as relações de derivação entre versões: através de uma estrutura hierárquica e por meio de uma estrutura plana com ponteiros. Quando codificadas sob a forma de documentos XML, a estrutura hierárquica se traduz em versões aninhadas, enquanto que na estrutura plana, na qual a hierarquia é modelada por ponteiros, cada versão fica completamente isolada das demais dentro de seu próprio fragmento de XML. Na estrutura plana, para localizar uma determinada versão dentro do documento XML através de seu identificador, é necessário realizar uma busca linear, varrendo potencialmente o arquivo inteiro. Com a estrutura hierárquica, é possível atribuir identificadores às versões de forma a acelerar o processo de localização para uma busca de custo logarítmico, além de poder testar em tempo constante, sem pesquisar a estrutura em árvore, se uma dada versão descende de outra qualquer. Um estudo sobre diferentes esquemas de numeração com essas propriedades pode ser encontrado em (COHEN; KAPLAN; MILO, 2002). Seria possível, por exemplo, atribuir ao *layout* de Ano Novo o identificador 1.1.2, indicando que é o segundo filho do primeiro filho da versão raiz, permitindo assim localizá-lo simplesmente navegando pelos arcos a partir da versão raiz. Por esta razão, neste trabalho optou-se por usar a estrutura hierárquica.

3.4 Visão Completa do Modelo TVX

Esta seção apresenta o modelo TVX através de uma abordagem *bottom-up*. As subseções que seguem descrevem o modelo desmembrando-o nas diversas classes e relacionamentos que o compõe, fazendo um paralelo com as entidades XML correspondentes, para no final apresentar a visão completa do modelo. Os diagramas são apresentados usando a notação UML (OMG, 2004).

3.4.1 Modelagem do Documento e de suas Versões

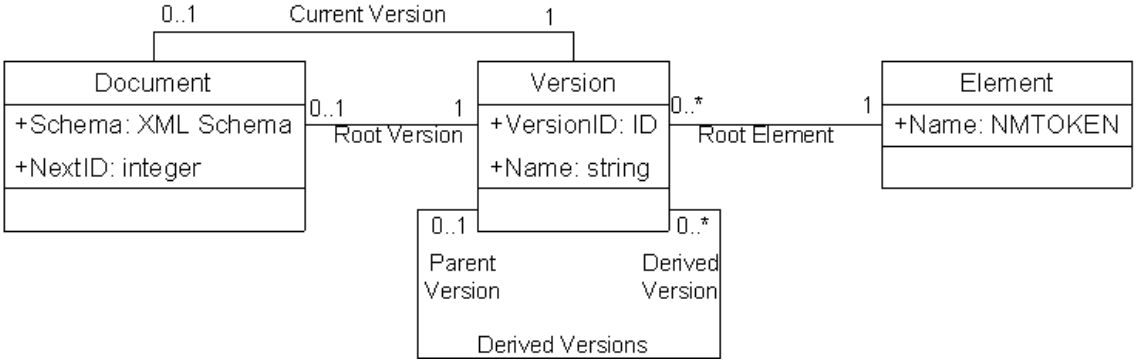


Figura 3.6: Modelagem do documento e de suas versões

Cada documento (classe *Document*) pode ter um esquema associado (Figura 3.6); este esquema, se presente, deve ser definido quando o documento é criado, e uma vez definido permanece para sempre inalterado. O esquema é, portanto, imutável e igual para todas as versões (classe *Version*) que vierem a ser associadas àquele documento. Cada modificação que vier a ocorrer deve manter todas as versões, em qualquer instante, coerentes com o esquema. Se nenhum esquema for definido, então não há restrição quanto às operações de atualização que podem ser aplicadas ao documento modelado. O formato escolhido para representação do esquema é *XML Schema* (W3C, 2000), por três razões básicas: amplo uso e aceitação, flexibilidade superior em comparação a DTD's e principalmente por ser um dialeto XML, o que torna possível incorporá-lo diretamente ao documento.

Cada documento possui uma versão inicial (também chamada de versão raiz, identificada através do relacionamento *Root Version*) da qual é possível derivar outras versões, através do relacionamento *Derived Versions*, cujas estruturas seguirão o mesmo esquema definido para a primeira. Cada versão apresenta um identificador e um nome para diferenciá-la das demais, bem como um elemento distinto que corresponde à raiz do documento XML que representa, indicado pelo relacionamento *Root Element* com a classe *Element*. Há também um relacionamento *Current Version* que indica qual a versão corrente, cujo funcionamento foi explicado na seção 3.3. Por fim, uma propriedade dentro da classe *Document* registra qual o próximo identificador livre, o qual será associado ao próximo elemento ou nodo de texto que venha a ser criado.

3.4.2 Modelagem de Objetos XML

Todos os objetos XML foram modelados como descendentes da classe *XMLObject*, vista na Figura 3.7, e herdam o relacionamento *Validity* que a mesma apresenta com a classe *TimeStamps*; dessa forma, elementos, atributos, nodos de texto e de *string* possuem uma série de rótulos temporais que indicam seu tempo de vida. As propriedades *IVT*, *FVT*, *ITT* e *FTT* correspondem, respectivamente, aos tempos de validade inicial e final e aos tempos de transação inicial e final. Os rótulos temporais dos elementos que são raízes de documentos também se aplicam às versões às quais são subordinados. Cada um destes objetos apresenta a restrição de que seu intervalo de vida deve estar contido no intervalo de vida de seu superior imediato na hierarquia.

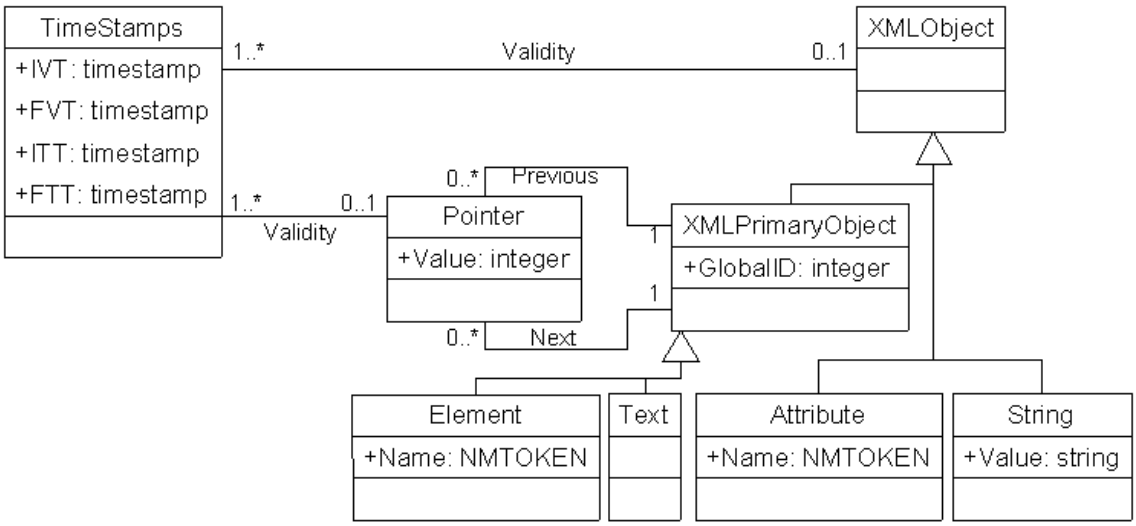


Figura 3.7: Modelagem de objetos XML

Adicionalmente, foi definida a classe *XMLPrimaryObject*, que indica os objetos que podem existir desvinculados de outros na árvore XML – a saber, elementos (classe *Element*) e nodos de texto (classe *Text*). Atributos (classe *Attribute*) sempre aparecem vinculados a um elemento, e *strings* (classe *String*) existem apenas em conjunto com um atributo ou um nodo de texto, portanto estes são considerados objetos secundários. Os objetos primários são, portanto, aqueles que requerem um identificador global, representado na figura pela propriedade *GlobalID*; conforme mencionado no início deste capítulo, na seção 3.1, são também aqueles dentre os quais existe uma relação de ordem, modelado na figura através dos relacionamentos *Previous* e *Next* e da classe *Pointer*, que em conjunto formam uma estrutura de lista duplamente encadeada. A figura não mostra, contudo, a restrição de que dois objetos do tipo *Text* não podem aparecer consecutivamente na ordenação. Não só o conteúdo destes objetos como também sua posição dentro da árvore XML está sujeita a modificações, portanto a relação de ordem também varia com o tempo, o que é modelado através de outro relacionamento com a classe *TimeStamps*. As classes *XMLObject* e *XMLPrimaryObject* são não-instanciáveis, servindo apenas para identificar propriedades e relacionamentos comuns às suas classes especializadas.

3.4.2.1 Modelagem de Elementos

Dentre os objetos XML modelados, os elementos (apresentados na Figura 3.8) são os mais complexos, pois são os que apresentam maior número de relacionamentos com outros objetos. Na Figura 3.8, correspondem à classe *Element*. Além do atributo *GlobalID*, herdado da classe *XMLPrimaryObject*, e do relacionamento *Validity*, herdado de *XMLObject*, possuem uma propriedade que indica seu nome dentro do documento XML.

Para cada *Element* há também um conjunto (possivelmente vazio) de atributos que correspondem àquele elemento. Cada atributo é modelado por uma instância da classe *Attribute*; os atributos também possuem nome, assim como os elementos, e este nome não deve se repetir para os atributos que são descendentes imediatos de um mesmo elemento. Cada elemento pode conter também sub-elementos e nodos de texto; estes são modelados pela relação *Content*. Repare que a descrição através do diagrama de classes não é suficiente para mostrar a restrição de que os objetos relacionados através das relações *Previous* e *Next* devem relacionar-se ao mesmo elemento na relação *Content*.

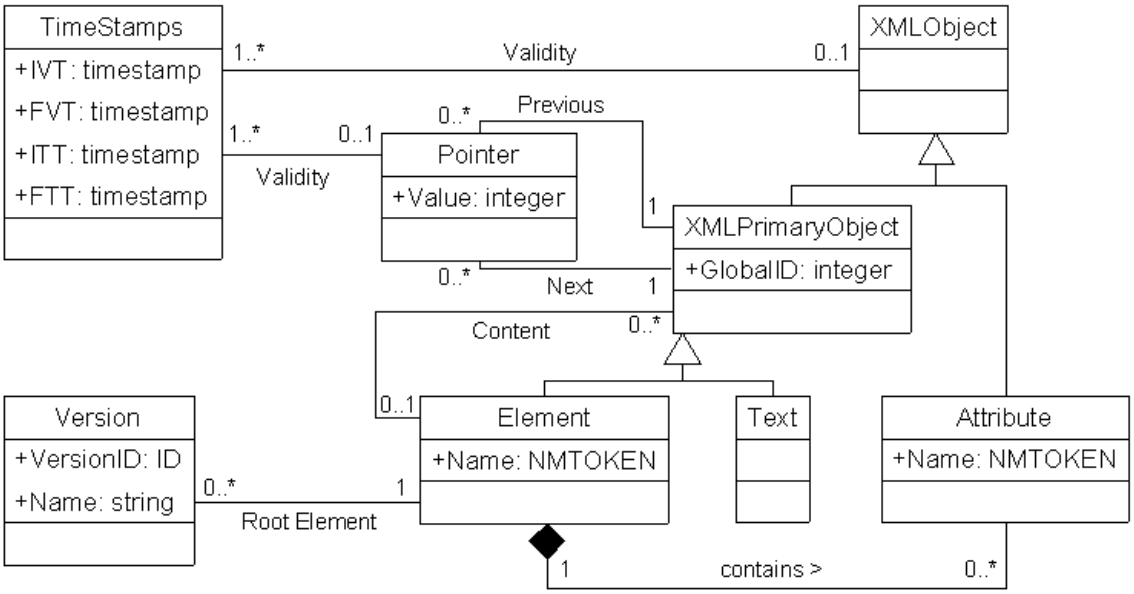


Figura 3.8: Modelagem de elementos

3.4.2.2 Modelagem de atributos, nodos de texto e strings

A Figura 3.9 apresenta em detalhe a representação de nodos de atributos, texto e *strings* no modelo TVX. Ao longo de sua existência, atributos podem assumir diversos valores, cada um com seu próprio intervalo de validade (recorde o exemplo envolvendo o elemento *Pessoa* e o atributo *Telefone*, apresentado na seção 3.2); esses são registrados pela classe *String*. Cada *String* possui uma propriedade que identifica seu valor, bem como *timestamps* para registrar os intervalos temporais associados a cada um dos valores assumidos ao longo de tempo. Nodos de texto também possuem rótulos temporais para o próprio objeto e para cada um seus valores, construídos da mesma maneira das outras propriedades.

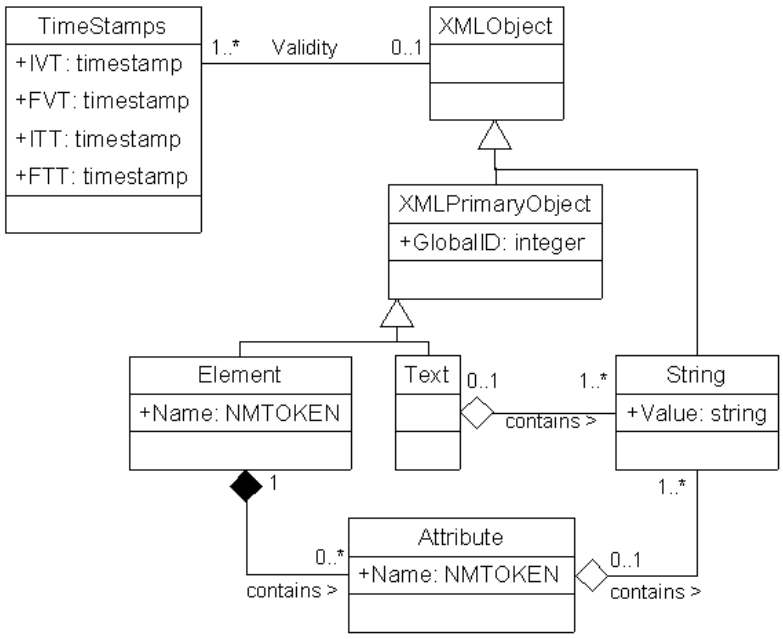


Figura 3.9: Modelagem de atributos, nodos de texto e *strings*

3.4.3 Diagrama de Classes do Modelo TVX

O modelo TVX completo pode ser descrito pelo diagrama de classes mostrado na Figura 3.10. Note que o modelo não estipula a granularidade dos rótulos temporais, permitindo que a mesma varie de acordo com a aplicação.

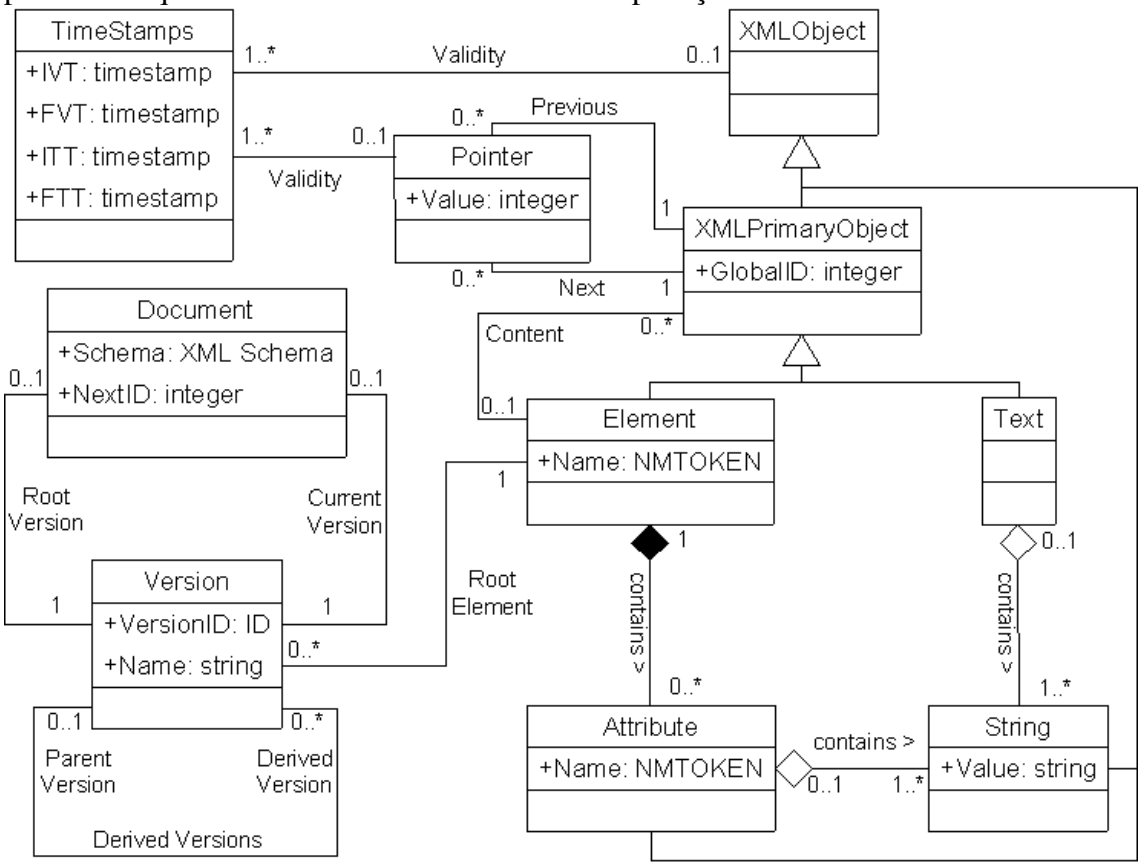


Figura 3.10: Diagrama de classes do modelo TVX

3.5 Implementação do Modelo TVX em XML

Como já foi dito na seção 3.4.1, o formato escolhido para representação do modelo TVX em XML foi XML Schema. Esta seção apresenta gradualmente como traduzir o diagrama de classes da seção anterior para as construções desta linguagem, de forma a armazenar todo o histórico de um documento em evolução em um único arquivo XML.

```
<element name="Document">
 <complexType>
  <sequence>
   <element name="Schema" type="any" minOccurs="0"/>
   <element name="Version" type="VersionType"/>
  </sequence>
  <attribute name="Current" type="IDREF" use="required"/>
  <attribute name="NextID" type="integer" use="required"/>
 </complexType>
</element>
```

Figura 3.11: Codificação da classe *Document* em XML Schema

O armazenamento de um documento começa pela classe *Document*, logo começaremos a implementação por esta classe. A mesma será implementada em XML através de um elemento de mesmo nome, o qual será de tipo complexo, pois incluirá várias informações. Primeiramente, deve incluir espaço para o esquema do documento,

o qual ficará armazenado em um sub-elemento de nome *Schema*. Este é opcional, o que é indicado pela propriedade *minOccurs* ajustada para zero. Também a partir de *Document* se tem acesso à versão raiz do documento, contida no primeiro elemento *Version*. O detalhamento da estrutura das versões será dado mais adiante. Por fim, dois atributos *Current* e *NextID* indicam qual a versão corrente e qual o próximo valor livre para os identificadores globais. A codificação em XML Schema para a classe *Document* pode ser vista na Figura 3.11.

```
<complexType name="VersionType">
 <sequence>
  <element name="Element" type="ElementType"/>
  <element name="DerivedVersions">
   <complexType>
    <element name="Version" type="VersionType" minOccurs="0"
             maxOccurs="unbounded"/>
   </complexType>
  </element>
 </sequence>
 <attribute name="VersionID" type="ID" use="required"/>
 <attribute name="Name" type="string" use="required"/>
</complexType>
```

Figura 3.12: Codificação da classe *Version* em XML Schema

A codificação de versões em XML Schema é mostrada na Figura 3.12. Versões também são tipos complexos, pois além de um atributo que lhes serve de identificador – *VersionID* – e de outro para guardar seu nome, também possuem dois sub-elementos, um para indicar o elemento raiz ao qual estão vinculados, e outro para suas versões derivadas. Estas versões derivadas são opcionais (*minOccurs* ajustado para zero) e ilimitadas (*maxOccurs* ajustado para "*unbounded*"), e por serem também do tipo *VersionType*, possuem cada uma a mesma estrutura mostrada na Figura 3.12.

```
<complexType name="ElementType">
 <sequence>
  <element name="Validity" type="ValidityType"/>
  <element name="Attributes">
   <complexType>
    <element name="Attribute" type="AttributeType" minOccurs="0"
             maxOccurs="unbounded"/>
   </complexType>
  </element>
  <element name="Content" type="ContentType"/>
  <element name="Previous">
   <complexType>
    <element name="Pointer" type="PointerType" maxOccurs="unbounded"/>
   </complexType>
  </element>
  <element name="Next">
   <complexType>
    <element name="Pointer" type="PointerType" maxOccurs="unbounded"/>
   </complexType>
  </element>
 </sequence>
 <attribute name="Name" type="NMTOKEN" use="required"/>
 <attribute name="GlobalID" type="integer" use="required"/>
</complexType>
```

Figura 3.13: Codificação da classe *Element* em XML Schema

A estrutura de todos os nodos do tipo elemento, incluindo os elementos raízes de cada versão, pode ser visto na Figura 3.13. Em primeiro lugar, vê-se que elementos também possuem identificadores, como as versões, porém ao contrário destas seu identificador não é do tipo *ID*, pois um mesmo elemento pode aparecer replicado em múltiplas versões, e um campo do tipo *ID* não pode repetir valores dentro de um mesmo documento XML. Elementos possuem também um nome, o qual por ser uma simples

cadeia de caracteres é guardado dentro de um atributo *Name*. O tempo de vida do elemento é guardado no sub-elemento *Validity*; seus atributos, se existirem, no sub-elemento *Attributes*, que pode conter um número qualquer de elementos do tipo *Attribute*; e outros elementos e nodos de texto que sejam seus filhos, no sub-elemento *Content*. Todos estes são tipos especiais de dados que serão detalhados no decorrer desta seção. Os últimos elementos contidos dentro de um *Element* são ponteiros denominados *Previous* e *Next*; estes são associados a todos os elementos (e também a todos os nodos de texto, como veremos mais adiante), inclusive ao elemento raiz, e servem para registrar a relação de ordem existente entre os descendentes de um mesmo elemento na árvore XML.

O elemento *Validity* mostrado anteriormente, o qual é a materialização do relacionamento de mesmo nome existente entre as classes *TimeStamps* e *XMLObject* (subseção 3.4.2), é simplesmente uma coleção de elementos *TS*, representando os *timestamps* de tempo de validade e de tempo de transação que registram o tempo de vida do objeto ao qual estão associados. Cada um é composto por quatro campos, *IVT*, *FVT*, *ITT*, e *FTT*, correspondendo às propriedades da classe *TimeStamps* mostrada na seção anterior. A codificação para estes elementos é mostrada na Figura 3.14.

```
<complexType name="ValidityType">
 <element name="TS" type="TSType" maxOccurs="unbounded"/>
</complexType>
<complexType name="TSType">
 <attribute name="IVT" type="string" use="required"/>
 <attribute name="FVT" type="string" use="required"/>
 <attribute name="ITT" type="string" use="required"/>
 <attribute name="FTT" type="string" use="required"/>
</complexType>
```

Figura 3.14: Codificação da relação *Validity* e da classe *TimeStamps* em XML Schema

Atributos são codificados através de elementos de nome *Attribute*, cuja descrição é mostrada na Figura 3.15. Tal como *Element*, também possuem um elemento *Validity* para indicar seu tempo de vida (recordando da seção anterior, tanto elementos quanto atributos herdavam este relacionamento da classe *XMLObject*), bem como um atributo contendo seu nome. O conjunto de valores assumidos pelos atributos fica contido em elementos *String*, os quais apresentam, para cada valor, um conjunto de rótulos temporais denotando o intervalo durante o qual são válidos.

```
<complexType name="AttributeType">
 <sequence>
  <element name="Validity" type="ValidityType"/>
  <element name="String" type="StringType" maxOccurs="unbounded"/>
 </sequence>
 <attribute name="Name" type="NMTOKEN" use="required"/>
</complexType>
<complexType name="StringType">
 <sequence>
  <element name="Value" type="string"/>
  <element name="TS" type="TSType" maxOccurs="unbounded"/>
 </sequence>
</complexType>
```

Figura 3.15: Codificação das classes *Attribute* e *String* em XML Schema

O elemento *Content* (Figura 3.16) pode abrigar tanto elementos do tipo *Element* quanto do tipo *Text*, correspondendo ao relacionamento entre as classes *Element* e *XMLPrimaryObject* da seção anterior. Elementos do tipo *Text* também apresentam identificador global e sub-elementos dos tipos *Validity*, *Previous* e *Next*, os quais são exatamente como aqueles mostrados para *Element*. Adicionalmente, também podem

conter valores textuais, assim como atributos, e por isso apresentam a mesma relação que estes para com sub-elementos do tipo *String*.

```
<complexType name="ContentType">
 <choice minOccurs="0" maxOccurs="unbounded">
  <element name="Element" type="ElementType"/>
  <element name="Text" type="TextType"/>
 </choice>
</complexType>
<complexType name="TextType">
 <sequence>
  <element name="Validity" type="ValidityType"/>
  <element name="String" type="StringType" maxOccurs="unbounded"/>
  <element name="Previous">
   <complexType>
    <element name="Pointer" type="PointerType" maxOccurs="unbounded"/>
   </complexType>
  </element>
  <element name="Next">
   <complexType>
    <element name="Pointer" type="PointerType" maxOccurs="unbounded"/>
   </complexType>
  </element>
 </sequence>
 <attribute name="GlobalID" type="string" use="required"/>
</complexType>
```

Figura 3.16: Codificação da relação *Content* e da classe *Text* em XML Schema

Por fim, resta explicar como funcionam os ponteiros *Previous* e *Next* que aparecem em elementos *Element* e *Text*. Cada ponteiro possui um atributo que corresponde a uma referência ao identificador global do elemento ou nodo de texto para o qual apontam; como a atribuição de identificadores começa pelo número 1 (um), o valor zero pode ser usado para indicar um ponteiro com valor nulo, como o ponteiro *Previous* do primeiro filho de um elemento ou o ponteiro *Next* do último filho. Como a relação de ordem pode mudar, pode haver diversos valores associados a cada um desses ponteiros, cada qual com seu intervalo de validade (vide Figura 3.17).

```
<complexType name="PointerType">
 <element name="TS" type="TSType" maxOccurs="unbounded"/>
 <attribute name="Value" type="integer" use="required"/>
</complexType>
```

Figura 3.17: Codificação dos ponteiros da classe *Pointer* em XML Schema

A Figura 3.18 combina os diversos fragmentos de código mostrados nesta seção, apresentando a descrição completa em XML Schema da implementação do modelo TVX em XML. Para concluir o exemplo, as Figuras 3.19 e 3.20 mostram como o documento em evolução da Figura 3.3 pode ser codificado com a abordagem proposta. Neste exemplo, contudo, o versionamento não é mostrado (3.3a e 3.3b são dois estados distintos de uma mesma versão), mas uma versão nova seria representada simplesmente por outro elemento *Version* com uma cópia modificada do documento, a qual teria a mesma estrutura vista nas figuras que seguem.

```
<schema
xmlns="http://www.w3.org/2001/XMLSchema">
 <element name="Document">
  <complexType>
   <sequence>
    <element name="Schema" type="any"
             minOccurs="0"/>
    <element name="Version"
             type="VersionType"/>
   </sequence>
   <attribute name="Current" type="IDREF"
              use="required"/>
   <attribute name="NextID" type="integer"
              use="required"/>
  </complexType>
 </element>
 <complexType name="VersionType">
  <sequence>
   <element name="Element"
            type="ElementType"/>
   <element name="DerivedVersions">
    <complexType>
     <element name="Version"
              type="VersionType"
              minOccurs="0"
              maxOccurs="unbounded"/>
    </complexType>
   </element>
  </sequence>
  <attribute name="VersionID" type="ID"
             use="required"/>
  <attribute name="Name" type="string"
             use="required"/>
 </complexType>
 <complexType name="ElementType">
  <sequence>
   <element name="Validity"
            type="ValidityType"/>
   <element name="Attributes">
    <complexType>
     <element name="Attribute"
              type="AttributeType"
              minOccurs="0"
              maxOccurs="unbounded"/>
    </complexType>
   </element>
   <element name="Content"
            type="ContentType"/>
   <element name="Previous">
    <complexType>
     <element name="Pointer"
              type="PointerType"
              maxOccurs="unbounded"/>
    </complexType>
   </element>
   <element name="Next">
    <complexType>
     <element name="Pointer"
              type="PointerType"
              maxOccurs="unbounded"/>
    </complexType>
   </element>
  </sequence>
  <attribute name="Name" type="NMTOKEN"
             use="required"/>
  <attribute name="GlobalID"
             type="integer"
             use="required"/>
 </complexType>
 <complexType name="ValidityType">
  <element name="TS" type="TSType"
           maxOccurs="unbounded"/>
 </complexType>
 <complexType name="TSType">
  <attribute name="IVT" type="string"
             use="required"/>
  <attribute name="FVT" type="string"
             use="required"/>
  <attribute name="ITT" type="string"
             use="required"/>
  <attribute name="FTT" type="string"
             use="required"/>
 </complexType>
 <complexType name="AttributeType">
  <sequence>
   <element name="Validity"
            type="ValidityType"/>
   <element name="String" type="StringType"
            maxOccurs="unbounded"/>
  </sequence>
  <attribute name="Name" type="NMTOKEN"
             use="required"/>
 </complexType>
 <complexType name="StringType">
  <sequence>
   <element name="Value" type="string"/>
   <element name="TS" type="TSType"
            maxOccurs="unbounded"/>
  </sequence>
 </complexType>
 <complexType name="ContentType">
  <choice minOccurs="0"
          maxOccurs="unbounded">
   <element name="Element"
            type="ElementType"/>
   <element name="Text" type="TextType"/>
  </choice>
 </complexType>
 <complexType name="TextType">
  <sequence>
   <element name="Validity"
            type="ValidityType"/>
   <element name="String" type="StringType"
            maxOccurs="unbounded"/>
   <element name="Previous">
    <complexType>
     <element name="Pointer"
              type="PointerType"
              maxOccurs="unbounded"/>
    </complexType>
   </element>
   <element name="Next">
    <complexType>
     <element name="Pointer"
              type="PointerType"
              maxOccurs="unbounded"/>
    </complexType>
   </element>
  </sequence>
  <attribute name="GlobalID" type="string"
             use="required"/>
 </complexType>
 <complexType name="PointerType">
  <element name="TS" type="TSType"
           maxOccurs="unbounded"/>
  <attribute name="Value" type="integer"
             use="required"/>
 </complexType>
</schema>
```

Figura 3.18: Implementação completa do modelo TVX em XML Schema

```
<Document Current="1" NextID="8">
 <Schema>
  <schema
xmlns="http://www.w3.org/2001/XMLSchema">
   <element name="Layout">
    <complexType>
     <choice maxOccurs="unbounded">
      <element name="Banner"
               type="string"/>
      <element name="Link">
       <complexType>
        <simpleContent>
         <extension base="xs:string">
          <attribute name="Color"
                     type="string"/>
         </extension>
        </simpleContent>
       </complexType>
      </element>
     </choice>
    </complexType>
   </element>
  </schema>
 </Schema>
 <Version VersionID="1" Name="Principal">
  <Element Name="Layout" GlobalID="1">
   <Validity>
    <TS IVT="01/06/04" FVT="now"
        ITT="01/06/04" FTT="now"/>
   </Validity>
   <Attributes/>
   <Content>
    <Element Name="Banner" GlobalID="2">
     <Validity>
      <TS IVT="01/06/04" FVT="now"
          ITT="01/06/04" FTT="now"/>
     </Validity>
     <Attributes/>
     <Content>
      <Text GlobalID="3">
       <Validity>
        <TS IVT="01/06/04" FVT="now"
            ITT="01/06/04" FTT="now"/>
       </Validity>
       <String>
        <Value> RGS Industries </Value>
        <TS IVT="01/06/04" FVT="now"
            ITT="01/06/04" FTT="now"/>
       </String>
       <Previous>
        <Pointer Value="0">
         <TS IVT="01/06/04" FVT="now"
             ITT="01/06/04" FTT="now"/>
        </Pointer>
       </Previous>
       <Next>
        <Pointer Value="0">
         <TS IVT="01/06/04" FVT="now"
             ITT="01/06/04" FTT="now"/>
        </Pointer>
       </Next>
      </Text>
     </Content>
     <Previous>
      <Pointer Value="0">
       <TS IVT="01/06/04" FVT="now"
           ITT="01/06/04" FTT="09/06/04"/>
       <TS IVT="01/06/04" FVT="14/06/04"
           ITT="10/06/04" FTT="now"/>
      </Pointer>
      <Pointer Value="6">
       <TS IVT="15/06/04" FVT="now"
           ITT="10/06/04" FTT="now"/>
      </Pointer>
     </Previous>
     <Next>
      <Pointer Value="4">
       <TS IVT="01/06/04" FVT="now"
           ITT="01/06/04" FTT="09/06/04"/>
       <TS IVT="01/06/04" FVT="14/06/04"
           ITT="10/06/04" FTT="now"/>
      </Pointer>
      <Pointer Value="0">
       <TS IVT="15/06/04" FVT="now"
           ITT="10/06/04" FTT="now"/>
      </Pointer>
     </Next>
    </Element>
    <Element Name="Link" GlobalID="4">
     <Validity>
      <TS IVT="01/06/04" FVT="now"
          ITT="01/06/04" FTT="now"/>
     </Validity>
     <Attributes>
      <Attribute Name="Color">
       <Validity>
        <TS IVT="01/06/04" FVT="now"
            ITT="01/06/04" FTT="09/06/04"/>
        <TS IVT="01/06/04" FVT="14/06/04"
            ITT="10/06/04" FTT="now"/>
       </Validity>
       <String>
        <Value> Cyan </Value>
        <TS IVT="01/06/04" FVT="now"
            ITT="01/06/04" FTT="09/06/04"/>
        <TS IVT="01/06/04" FVT="14/06/04"
            ITT="10/06/04" FTT="now"/>
       </String>
      </Attribute>
     </Attributes>
     <Content>
      <Text GlobalID="5">
       <Validity>
        <TS IVT="01/06/04" FVT="now"
            ITT="01/06/04" FTT="now"/>
       </Validity>
       <String>
        <Value> Main Page </Value>
        <TS IVT="01/06/04" FVT="now"
            ITT="01/06/04" FTT="09/06/04"/>
        <TS IVT="01/06/04" FVT="14/06/04"
            ITT="10/06/04" FTT="now"/>
       </String>
       <String>
        <Value> About Us </Value>
        <TS IVT="15/06/04" FVT="now"
            ITT="10/06/04" FTT="now"/>
       </String>
       <Previous>
        <Pointer Value="0">
         <TS IVT="01/06/04" FVT="now"
             ITT="01/06/04" FTT="now"/>
        </Pointer>
       </Previous>
       <Next>
        <Pointer Value="0">
         <TS IVT="01/06/04" FVT="now"
             ITT="01/06/04" FTT="now"/>
        </Pointer>
       </Next>
      </Text>
     </Content>
     <Previous>
      <Pointer Value="2">
       <TS IVT="01/06/04" FVT="now"
           ITT="01/06/04" FTT="09/06/04"/>
       <TS IVT="01/06/04" FVT="14/06/04"
           ITT="10/06/04" FTT="now"/>
      </Pointer>
      ...
```

Figura 3.19: Codificação do documento de exemplo no modelo TVX

```
  ...
  <Pointer Value="0">
   <TS IVT="15/06/04" FVT="now"
       ITT="10/06/04" FTT="now"/>
  </Pointer>
 </Previous>
 <Next>
  <Pointer Value="0">
   <TS IVT="01/06/04" FVT="now"
       ITT="01/06/04" FTT="09/06/04"/>
   <TS IVT="01/06/04" FVT="14/06/04"
       ITT="10/06/04" FTT="now"/>
  </Pointer>
  <Pointer Value="6">
   <TS IVT="15/06/04" FVT="now"
       ITT="10/06/04" FTT="now"/>
  </Pointer>
 </Next>
</Element>
<Element Name="Link" GlobalID="6">
 <Validity>
  <TS IVT="15/06/04" FVT="now"
      ITT="10/06/04" FTT="now"/>
 </Validity>
 <Attributes/>
 <Content>
  <Text GlobalID="7">
   <Validity>
    <TS IVT="15/06/04" FVT="now"
        ITT="10/06/04" FTT="now"/>
   </Validity>
   <String>
    <Value> Contact </Value>
    <TS IVT="15/06/04" FVT="now"
        ITT="10/06/04" FTT="now"/>
   </String>
   <Previous>
    <Pointer Value="0">
     <TS IVT="15/06/04" FVT="now"
         ITT="10/06/04" FTT="now"/>
    </Pointer>
   </Previous>
   <Next>
    <Pointer Value="0">
     <TS IVT="15/06/04" FVT="now"
         ITT="10/06/04" FTT="now"/>
    </Pointer>
   </Next>
  </Text>
 </Content>
 <Previous>
  <Pointer Value="4">
   <TS IVT="15/06/04" FVT="now"
       ITT="10/06/04" FTT="now"/>
  </Pointer>
 </Previous>
 <Next>
  <Pointer Value="2">
   <TS IVT="15/06/04" FVT="now"
       ITT="10/06/04" FTT="now"/>
  </Pointer>
 </Next>
</Element>
</Content>
<Previous>
 <Pointer Value="0">
  <TS IVT="01/06/04" FVT="now"
      ITT="01/06/04" FTT="now"/>
 </Pointer>
</Previous>
<Next>
 <Pointer Value="0">
  <TS IVT="01/06/04" FVT="now"
      ITT="01/06/04" FTT="now"/>
 </Pointer>
</Next>
</Element>
</Version>
</Document>
```

Figura 3.20: Codificação do documento de exemplo no modelo TVX (continuação)

Os significados das *tags* empregadas nas Figuras 3.19 e 3.20 já foram apresentados no decorrer desta seção: elementos, atributos e assim por diante são modelados por *tags* com os mesmos nomes. Os rótulos temporais que apareceram nos arcos de entrada nas Figuras 3.2 e 3.4 são agora *timestamps* registrados dentro de um elemento *Validity*, o qual se faz presente dentro de todas as *tags* principais; a maneira de se interpretar estes rótulos permanece a mesma que foi dada na seção 3.2. Recorde do exemplo que a forma de modelar corretamente a mudança na posição do elemento *Banner* havia sido deixada sem explicação, até que foram introduzidos os ponteiros *Previous* e *Next*. Estas figuras mostram como estes ponteiros funcionam: para o primeiro elemento *Link*, por exemplo, pode-se ver através dos rótulos que o mesmo costumava vir após o elemento *Banner* (aquele que possui um *GlobalID* igual a "2"), mas depois das mudanças promovidas não apresenta nenhum antecessor (o valor do ponteiro *Previous* foi ajustado para zero), sendo assim o primeiro descendente de seu pai. De forma análoga, costumava ser também o último descendente, porém agora é seguido por outro elemento *Link*, indicado pelo ponteiro *Next* com valor "6".

A Figura 3.21 mostra como ficaria o documento caso o mesmo tivesse múltiplas versões, como as mostradas na Figura 3.5. Na Figura 3.21, o esquema foi omitido, bem como grande parte do conteúdo de cada versão, por serem idênticos ao que já foi mostrado nas Figuras 3.19 e 3.20. A versão com *VersionID* igual a 1 representa o *layout* principal; possui apenas uma versão derivada, contida no elemento *DerivedVersions* e com *VersionID* igual a 1.1, correspondente ao *layout* de Verão.

```
<Document Current="1.1.2" NextID="14">
 <Schema> ... </Schema>
 <Version VersionID="1" Name="Principal">
  <Element Name="Layout" GlobalID="1"> ...
  </Element>
  <DerivedVersions>
   <Version VersionID="1.1" Name="Verao">
    <Element Name="Layout" GlobalID="1">
     <Validity> ... </Validity>
     <Attributes/>
     <Content>
      ...
      <Element Name="Link" GlobalID="8">
       <Validity> ... </Validity>
       <Attributes/>
       <Content>
        <Text GlobalID="9">
         <Validity> ... </Validity>
         <String>
          <Value>
          Click here for our Summer schedule
          </Value>
          ...
         </String>
        </Text>
       </Content>
       ...
      </Element>
     </Content>
     ...
    </Element>
    <DerivedVersions>
     <Version VersionID="1.1.1"
              Name="Natal">
      <Element Name="Layout" GlobalID="1">
       <Validity> ... </Validity>
       <Attributes/>
       <Content>
        ...
        <Element Name="Banner"
                 GlobalID="10">
         <Validity> ... </Validity>
         <Attributes/>
         <Content>
          <Text GlobalID="11">
           <Validity> ... </Validity>
           <String>
            <Value>
             We wish you a Merry Christmas!
            </Value>
            ...
           </String>
          </Text>
         </Content>
         ...
        </Element>
       </Content>
       ...
      </Element>
     </Version>
     <Version VersionID="1.1.2"
              Name="Ano Novo">
      <Element Name="Layout" GlobalID="1">
       <Validity> ... </Validity>
       <Attributes/>
       <Content>
        ...
        <Element Name="Banner"
                 GlobalID="12">
         <Validity> ... </Validity>
         <Attributes/>
         <Content>
          <Text GlobalID="13">
           <Validity> ... </Validity>
           <String>
            <Value> Happy New Year!
            </Value>
            ...
           </String>
          </Text>
         </Content>
         ...
        </Element>
       </Content>
       ...
      </Element>
     </Version>
    </DerivedVersions>
   </Version>
  </DerivedVersions>
 </Version>
</Document>
```

Figura 3.21: Versionamento no modelo TVX

Todos os elementos e nodos de texto que foram preservados entre as diferentes versões, como o elemento raiz *Layout*, aparecem nelas com os mesmos identificadores globais. Novos objetos próprios de cada versão, como o *Link* contendo o texto "Click here for our Summer schedule" pertencente apenas à versão de Verão, recebem seus próprios identificadores. As versões de Natal (*VersionID* igual a 1.1.1) e Ano Novo (*VersionID* igual a 1.1.2) receberam novos *banners*, contendo os textos "We wish you a Merry Christmas!" e "Happy New Year!", respectivamente. Conforme indicado pelo atributo *Current* no elemento *Document*, a versão de Ano Novo é a versão corrente.

3.6 Extração da Informação Temporal e Versionada

Esta seção mostra como recuperar o conteúdo em um dado ponto no tempo de um documento XML codificado no modelo TVX. Como o histórico de uma versão está inteiramente contido dentro do elemento *Version* correspondente, o tratamento do aspecto de versionamento na recuperação de informações é trivial: basta considerar apenas o conteúdo da(s) versão(ões) que se deseja pesquisar. Por esta razão, os exemplos que seguem (ainda baseados nos documentos da Figura 3.3) mostrarão apenas como funciona a recuperação dos documentos em relação aos rótulos bitemporais.

Como primeiro exemplo, considere que a propriedade *now* indica o dia 16/06/04 – isto é, este é o momento presente – e que se deseja recuperar o conteúdo atual do documento. O primeiro passo é filtrar pelo tempo de transação apenas os pares de rótulos cujos intervalos de transação contenham o dia 16/06/04. A Figura 3.22 mostra mais uma vez a representação gráfica do documento de exemplo, desta vez indicando em negrito os pares de rótulos que são selecionados como resultado da filtragem pelo tempo de transação. Para não poluir a figura, não foram incluídos os rótulos referentes aos ponteiros que indicam a relação de ordem entre elementos e nodos de texto; o tratamento destes, contudo, é idêntico ao que está sendo demonstrado para os demais objetos, e deve ser feito automaticamente pelo módulo de *software* executor da consulta.

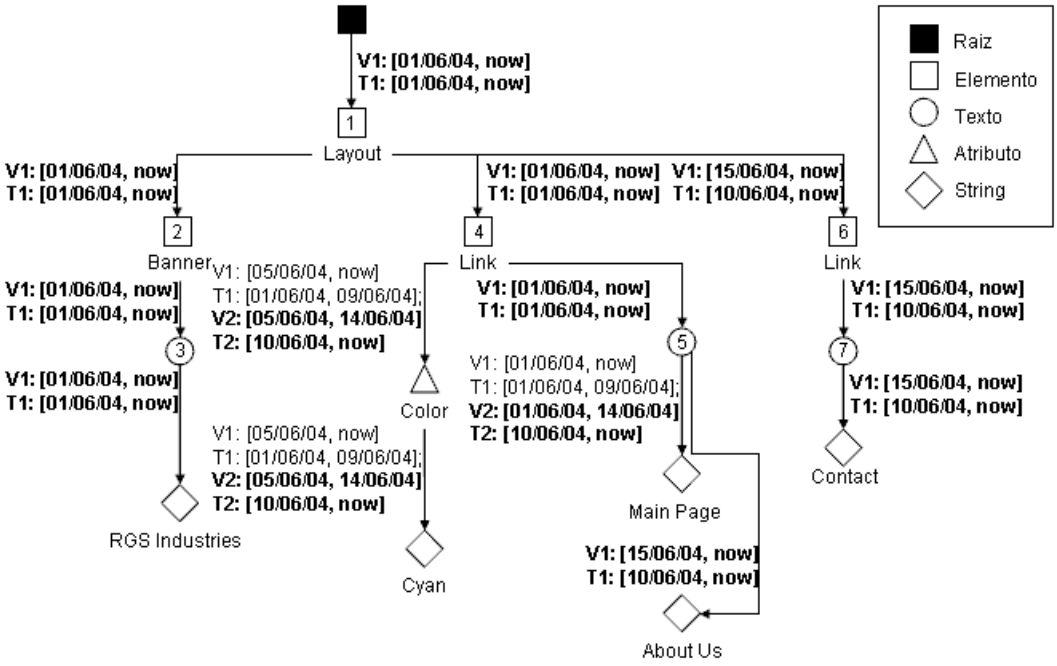


Figura 3.22: Recuperação do documento em relação ao dia 16/06/04

Uma vez selecionados os pares de rótulos que fazem parte da visão do dia 16/06/04, deve-se filtrá-los novamente em relação ao momento que se deseja consultar – mais uma vez, o momento presente – porém desta vez em relação ao tempo de validade. Os objetos que são eliminados por não serem válidos no dia 16/06/04 são o atributo *Color* e seu conteúdo de texto, válidos somente até o dia 14/06/04, e a *string* "Main Page", substituída a partir de 15/06/04 pela string "About Us". A visão do documento em 16/06/04 é, portanto, como mostrado na Figura 3.23a.

```
<Layout>
 <Link>
  About Us
 </Link>
 <Link>
  Contact
 </Link>
 <Banner>
  RGS Industries
 </Banner>
</Layout>
```

(a)

```
<Layout>
 <Banner>
  RGS Industries
 </Banner>
 <Link Color="Cyan">
  Main Page
 </Link>
</Layout>
```

(b)

Figura 3.23: Visão em 16/06/04 sobre o momento presente (a) e sobre o dia 12/06/04 (b)

Se, ao invés de consultar a visão sobre o momento presente, desejar-se consultar a visão atual sobre um momento passado, o procedimento é similar. Para, por exemplo, consultar a visão atual (ainda considerando que o presente é o dia 16/06/04) sobre o dia 12/06/04, a primeira etapa – seleção através do intervalo de transação – é idêntica à do exemplo anterior, resultando novamente na Figura 3.22. Na segunda etapa, dentre os objetos sobreviventes, serão selecionados apenas aqueles cujo intervalo de validade contenha o dia 12/06/04. O elemento *Link* cujo conteúdo é a *string* "Contact" será excluído do resultado, por ser válido apenas a partir de 15/06/04; já o atributo *Color* do outro elemento *Link*, antes excluído porque sua validade ia somente até o dia 14/06/04, agora entra no resultado, juntamente com seu conteúdo. Por fim, a *string* selecionada como conteúdo deste outro elemento *Link* é "Main Page". O resultado desta segunda consulta é mostrado na Figura 3.23b (lembrando, mais uma vez, que a mudança na posição do elemento *Banner* em relação aos demais é decorrente da existência de ponteiros que indicam a ordenação entre os objetos, omitidos da figura por simplicidade, os quais também estão sujeitos a variações com o decorrer do tempo).

As duas consultas mostradas até agora refletiam a visão *atual* sobre um ponto qualquer no tempo. Para consultar visões *passadas* – isto é, retornar ao estado de conhecimento em que se estava em um momento passado, desconsiderando todas as operações executadas posteriormente – é necessário realizar uma operação denominada *rollback*. Esta operação nada mais é do que a seleção de rótulos temporais pelo intervalo de transação feita anteriormente, porém em relação a um momento passado, ao invés do momento presente.

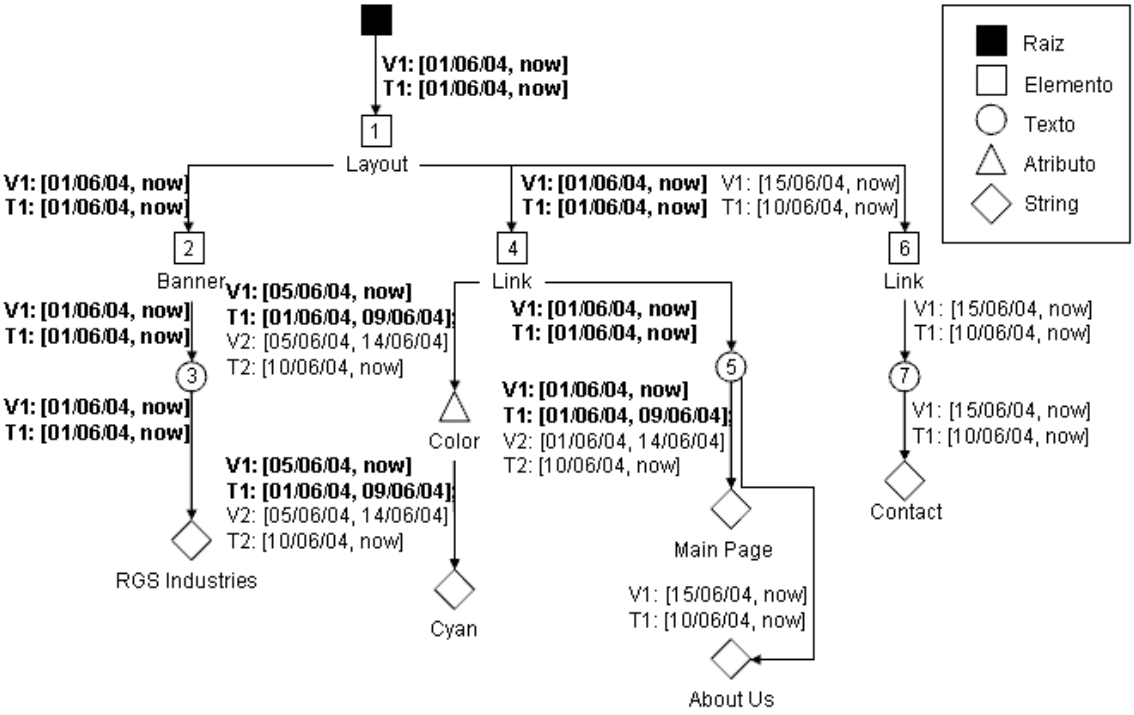


Figura 3.24: Recuperação do documento em relação ao dia 03/06/04

Por exemplo, suponha que se deseje consultar, em 16/06/04, a visão que se tinha do documento em 03/06/04. O primeiro passo é selecionar, através do intervalo de transação, apenas os pares de rótulos temporais que contenham o momento 03/06/04. O resultado é mostrado, em negrito, na Figura 3.24. Em seguida, seleciona-se através dos intervalos de tempo de validade apenas aqueles objetos válidos em 03/06/04. Dentre os objetos selecionados pelo tempo de transação, apenas o atributo *Color*, juntamente com

seu conteúdo, é excluído pela seleção por tempo de validade, pois ambos são válidos apenas a partir de 05/06/04. O resultado desta consulta é mostrado na Figura 3.25a.

```
<Layout>
 <Banner>
  RGS Industries
 </Banner>
 <Link>
  Main Page
 </Link>
</Layout>
```

(a)

```
<Layout>
 <Banner>
  RGS Industries
 </Banner>
 <Link Color="Cyan">
  Main Page
 </Link>
</Layout>
```

(b)

Figura 3.25: Visão em 03/06/04 sobre o dia 03/06/04 (a) e sobre o dia 06/06/04(b)

Assim como se pode consultar a visão atual sobre um ponto qualquer no tempo, também é possível executar um *rollback* para pesquisar a visão passada sobre um momento qualquer. Por exemplo, é possível consultar a visão que se tinha em 03/06/04 sobre como seria o documento no dia 06/06/04. Para tanto, primeiro é feita a seleção através do tempo de transação, tendo como parâmetro o dia 03/06/04; o resultado é, novamente, indicado em negrito na Figura 3.24. Em seguida, faz-se a seleção pelo tempo de validade, tendo como parâmetro a data 06/06/04. Desta vez, o atributo *Color* não é excluído do resultado, pois em 03/06/04 acreditava-se que o mesmo seria válido de 05/06/04 em diante, o que inclui o dia 06/06/04. O resultado deste último exemplo de consulta temporal é mostrado na Figura 3.25b.

Para facilitar a compreensão de seu funcionamento, nos exemplos desta seção foi realizada primeiro a seleção através do tempo de transação e, após, em uma etapa separada, a seleção através do tempo de validade. É importante destacar, contudo, que por serem propriedades independentes, na prática é possível filtrar os nodos da árvore XML por tempo de transação e por tempo de validade simultaneamente.

3.7 Considerações Finais

Este capítulo apresentou o modelo TVX, capaz de gerenciar a evolução do conteúdo de documentos XML através das propriedades de tempo de transação e de tempo de validade e da possibilidade de versionamento dos documentos. Através destes dados, é possível recuperar visões presentes e passadas sobre o documento em qualquer ponto no tempo, bem como consultar diferentes alternativas para a evolução do mesmo. A codificação resultante toma também a forma de um documento XML, apresentando portanto todas as vantagens inerentes a essa linguagem, como por exemplo independência de plataforma.

Apesar da flexibilidade propiciada pelo uso simultâneo de tempo de transação, tempo de validade e versionamento, o modelo TVX possui suas limitações. Em primeiro lugar, não há qualquer suporte ao gerenciamento dos esquemas dos documentos XML, os quais, se definidos, permanecem inalteráveis conforme o conteúdo sofre modificações. Em segundo lugar, a criação de novas versões é realizada sobre o documento como um todo; ao contrário de outros trabalhos, como (GOLENDZINER, 1995) e (MORO, 2001), não é possível definir versões de objetos mais simples (no caso, sub-árvores do documento XML). Embora haja uma classe de aplicações para as quais essas limitações sejam irrelevantes – como a aplicação do estudo de caso que será apresentado no capítulo 6 – uma possível extensão que eliminasse essas limitações tornaria o modelo TVX mais abrangente. O capítulo 7 discute brevemente como e por quê essas limitações poderiam ser eliminadas.

A capacidade de representar o histórico de um documento, contudo, não é suficiente para o gerenciamento da evolução do mesmo. A seguir, os capítulos 4 e 5 apresentam, respectivamente, linguagens para consultar e modificar o conteúdo de documentos XML codificados segundo o formato definido neste capítulo.

4 LINGUAGEM DE CONSULTA

Este capítulo propõe uma linguagem de consulta para ser usada em conjunto com o modelo TVX, apresentado no capítulo anterior. A linguagem resultante baseia-se na linguagem XQuery (W3C, 2001) e conta com extensões das expressões de caminho XPath para acessar informações temporais e versionadas dentro do documento XML. Abordagens similares são apresentadas em (DYRESON, 2001), porém lidando apenas com tempo de transação, e em (ZHANG; DYRESON, 2002), lidando contudo apenas com tempo de validade. O objetivo da linguagem proposta neste capítulo é permitir ao usuário expressar consultas ao conteúdo do documento sobre a estrutura original do mesmo, sem que haja necessidade por parte do usuário final de conhecer a representação interna utilizada pelo modelo TVX.

O restante do capítulo está organizado como segue. A seção 4.1 apresenta as extensões feitas às expressões de caminho da linguagem XPath, detalhando a sintaxe e a semântica das novas construções que delas podem fazer parte. A seção 4.2 descreve as funções definidas para o tratamento das informações relativas à bitemporalidade e ao versionamento, e a seção 4.3 encerra o capítulo apresentando exemplos de expressões de consulta escritas na linguagem proposta.

4.1 Extensão das Expressões de Caminho XPath

Usualmente, para acessar o conteúdo de um documento em uma expressão XQuery, inicia-se a consulta por uma expressão do tipo:

```
FOR $v IN document("doc.xml")/nodoraiz/...
```

A cláusula *document* especifica o nome do documento que está sendo consultado, e em seguida a expressão de caminho seleciona o conjunto de nodos desejado, normalmente partindo do nodo raiz, associando-os à variável especificada junto ao FOR. Uma das características adicionadas pelo modelo TVX é o versionamento; portanto, deve haver um mecanismo capaz de selecionar uma ou mais versões de interesse dentro de uma expressão de caminho. Para tal, utiliza-se a expressão *tvx:version*, seguida de um predicado de seleção entre colchetes. O uso do prefixo *tvx*, seguido de dois pontos, é necessário para evitar ambigüidade, pois é possível que o documento pesquisado apresente elementos com o nome *version* em seu conteúdo. Por exemplo, para selecionar a versão cujo *VersionID* é 1.3, escreve-se:

```
FOR $v IN document("doc.xml")/tvx:version[1.3]/nodoraiz/...
```

É possível utilizar predicados mais elaborados para selecionar quais versões devem fazer parte da pesquisa, utilizando por exemplo as funções que são apresentadas na

seção 4.2. Por exemplo, considerando que as funções `ivt()` e `fvt()` retornam, respectivamente, os instantes de validade inicial e de validade final dos objetos passados como parâmetro, é possível selecionar todas as versões válidas durante o mês de janeiro de 2005 através da expressão `tvx:version[ivt(.) >= '01/01/05' and fvt(.) <= '31/01/05']`.

Há notações abreviadas para selecionar versões através de seus identificadores e de seus nomes. Sabendo, por exemplo, que a função `vid()` retorna o identificador da versão passada como parâmetro, percebe-se que `tvx:version[1.3]` é na verdade uma notação abreviada para `tvx:version[vid(.)='1.3']`. Da mesma forma, ao invés de se escrever `tvx:version[vname(.)='Principal']`, onde `vname()` é uma função que retorna o nome da versão passada como parâmetro, pode-se escrever diretamente `tvx:version['Principal']`. Note que as notações abreviadas são diferenciadas pela presença ou ausência de aspas simples dentro dos colchetes. O uso da cláusula *tvx:version* não é obrigatório; caso não esteja presente, a consulta atuará sobre a versão corrente.

Assim como é possível selecionar um conjunto de versões para fazer parte da consulta, é possível especificar que a consulta deve operar não sobre o estado atual da base de dados, mas sobre algum estado passado, por meio da operação de *rollback* definida sobre os intervalos de tempo de transação. Um exemplo desta operação foi apresentado na seção 3.6. Para tal fim, utiliza-se a expressão *tvx:rollback* em meio à expressão de caminho. A expressão *tvx:rollback* apresenta, entre colchetes, o instante de tempo ao qual se deve retornar. Por exemplo, para retornar a base de dados ao estado em que estava no dia 06/08/04, a expressão de caminho fica:

```
FOR $v IN document("doc.xml")/tvx:rollback['06/08/04']/nodoraiz/...
```

Assim como ocorre com a cláusula *tvx:version*, o uso da cláusula *tvx:rollback* não é obrigatório; caso não esteja presente, não é efetuado *rollback*, e a consulta atuará sobre o estado corrente da base de dados. Ao invés de valores absolutos, também é possível utilizar expressões mais complexas dentro dos colchetes da cláusula *tvx:rollback*. Por exemplo, para retornar ao estado em que a versão corrente se encontrava quando a versão "Verão" foi criada, e considerando que a função `itt()` retorna o tempo de transação inicial associado ao objeto passado como parâmetro, pode-se usar a expressão:

```
FOR $v1 IN document("doc.xml")/tvx:version['Verao'],
    $v2 IN document("doc.xml")/tvx:rollback[itt($v1)]/nodoraiz/...
```

Por fim, é possível incluir na expressão de caminho uma cláusula que selecione os nodos da árvore XML através de seu intervalo de validade. Esta cláusula recebe o nome de *tvx:validity*, e como parâmetro recebe, entre colchetes, um intervalo de validade. São selecionados apenas os objetos que são válidos durante todo o intervalo especificado por *tvx:validity*. Por exemplo, para selecionar os objetos cujo intervalo de validade engloba os meses de julho a dezembro de 2004, utiliza-se a expressão:

```
FOR $v IN document("doc.xml")/tvx:validity['01/07/04', '31/12/04']/...
```

Assim como com as cláusulas *tvx:version* e *tvx:rollback*, é possível construir expressões envolvendo o resultado de funções nos parâmetros da cláusula *tvx:validity*. Se ao invés de um intervalo temporal, desejar-se utilizar um ponto temporal para seleção através do tempo de validade, basta fazer com que os limites do intervalo sejam

iguais. Pode-se também utilizar a palavra-chave *EVER* dentro dos colchetes, para pesquisar todo o histórico do documento. Por fim, a cláusula *tvx:validity* não é obrigatória; quando ausente, são considerados apenas os objetos válidos no momento corrente. Se não foi executado um *rollback* antes da seleção por tempo de validade, o momento corrente é o momento em que a consulta é executada; caso contrário, o instante para o qual foi feito o *rollback* é considerado o momento corrente.

Estas três novas cláusulas, quando presentes, devem aparecer na expressão de caminho após a cláusula *document* e antes do restante da expressão, referente ao conteúdo do documento. As cláusulas podem aparecer em qualquer ordem, porém deve-se tomar cuidado ao escrever a expressão, pois a ordem em que aparecem pode afetar o resultado. Considere, por exemplo, as duas expressões abaixo; embora muito similares, produzem resultados diferentes. A primeira expressão seleciona todas as versões que, *no momento corrente*, são válidas a partir de 1° de janeiro de 2005, para então recuperar o estado em que estas se encontravam em 15 de junho de 2004. Já a segunda expressão executa o *rollback* antes de selecionar as versões, recuperando aquelas que eram válidas a partir de 1° de janeiro de 2005 *conforme a visão do dia 15 de junho de 2004*.

```
FOR $v IN document("doc.xml")/tvx:version[ivt(.) >= '01/01/05']/
tvx:rollback['15/06/04']/...

FOR $v IN document("doc.xml")/tvx:rollback['15/06/04']/
tvx:version[ivt(.) >= '01/01/05']/...
```

Na linguagem XPath padrão, embora seja possível especificar predicados de seleção que atuem sobre atributos e nodos de texto, é permitido às variáveis apontar apenas para elementos dentro do documento XML. Nesta extensão, esta restrição será relaxada; isto é, será permitido que as variáveis apontem para qualquer tipo de objeto dentro da árvore XML. Isto será necessário no próximo capítulo, o qual apresenta uma linguagem para modificação do conteúdo dos documentos, na qual deve ser possível fazer referência a um objeto qualquer como alvo das operações de alteração.

4.2 Funções para Acesso a Informações Temporais e Versionadas

Esta seção apresenta as funções que podem ser empregadas na construção de expressões de caminho, predicados de seleção e na construção do retorno da consulta. As funções relativas ao versionamento são:

- *vid(version)*: retorna o identificador da versão passada como parâmetro;
- *vname(version)*: retorna o nome da versão passada como parâmetro;
- *isroot(version)*: retorna um valor booleano (verdadeiro ou falso) indicando se a versão especificada é ou não a versão raiz do documento;
- *iscurrent(version)*: retorna um valor booleano indicando se a versão passada como parâmetro é ou não a versão corrente;
- *isparent(*$version_1$*,* $version_2$*)*: retorna um valor booleano indicando se a versão $version_1$ é ou não antecessora imediata da versão $version_2$ na hierarquia de derivação de versões;
- *ischild(*$version_1$*,* $version_2$*)*: complementar à função *isparent()*;
- *isascendant(*$version_1$*,* $version_2$*)*: retorna um valor booleano indicando se a versão $version_1$ é ou não antecessora da versão $version_2$ na hierarquia de derivação de versões, não importando o número de versões intermediárias entre elas existentes;

- *isdescendant(version$_1$, version$_2$)*: complementar à função *isascendant()*;
- *current()*: retorna o identificador da versão corrente;
- *root()*: retorna o identificador da versão raiz.

As funções do bloco a seguir, por sua vez, são relativas à manipulação de intervalos e instantes temporais, e foram definidas tomando por base as relações de Allen (ALLEN, 1983). Cada função recebe dois parâmetros, *a* e *b*, onde estes podem ser tanto instantes quanto intervalos. Caso sejam instantes, basta considerá-los como intervalos onde os limites superior e inferior são iguais. Considere, para interpretação da semântica de cada função, que a duração dos intervalos *a* e *b* é dada por [a_1, a_2] e [b_1, b_2], respectivamente.

- *equal(a, b)*: retorna um valor booleano indicando se os intervalo *a* e *b* são iguais, isto é, se $a_1 = b_1$ e $a_2 = b_2$;
- *different(a, b)*: complementar à função *equal()*;
- *precedes(a, b)*: retorna um valor booleano indicando se o intervalo *a* antecede o intervalo *b*, isto é, se $a_2 < b_1$;
- *follows(a, b)*: complementar à função *precedes()*;
- *contains(a, b)*: retorna um valor booleano indicando se o intervalo *a* contém o intervalo *b*, isto é, se $a_1 \leq b_1$ e $a_2 \geq b_2$;
- *into(a, b)*: retorna um valor booleano indicando se o intervalo *a* está contido no intervalo *b*, isto é, se $a_1 \geq b_1$ e $a_2 \leq b_2$;
- *intersect(a, b)*: retorna um valor booleano indicando se a interseção dos intervalos *a* e *b* não é vazia, isto é, se $a_1 \leq b_2$ e $a_2 \geq b_1$;
- *disjoint(a, b)*: complementar à função *intersect()*.

Além destas funções, também é possível utilizar os operadores >, <, >=, <=, = e <> para comparar instantes e intervalos. Neste ponto, cabe interromper a apresentação das funções para discutir uma peculiaridade do modelo TVX. Conforme visto na apresentação do modelo TVX, na seção 3.4, o mesmo não fixa a granularidade dos rótulos temporais, deixando-a variar conforme a necessidade da aplicação. Isto significa que enquanto uma determinada aplicação pode utilizar rótulos do tipo "dia/mês/ano", para outra pode ser necessário utilizar, por exemplo, "dia/mês/ano - hora:minuto". Por desconhecer a granularidade escolhida pelo usuário, o modelo TVX enxerga os rótulos temporais apenas como *strings* genéricas, sendo a princípio incapaz de comparar dois rótulos quaisquer, com exceção do teste de igualdade. Contornar esta situação é, na verdade, muito simples: exige-se do usuário apenas que defina, em XQuery, uma função *UserCompare*, que toma como parâmetro dois rótulos temporais e retorna um valor booleano indicando se o primeiro precede o segundo na lógica da aplicação alvo.

```
define function UserCompare($ts1, $ts2) {
  if number(substring-before($ts1, '/')) < number(substring-before($ts2, '/')
  then true()
  else if number(substring-before($ts1, '/')) = number(substring-before($ts2, '/') and
          number(substring-after($ts1, '/')) < number(substring-after($ts2, '/'))
       then true()
       else false()
}
```

Figura 4.1: Exemplo de definição da função *UserCompare*

Como exemplo, considere uma aplicação referente a uma universidade, onde a granularidade escolhida para os rótulos temporais é de um semestre. O usuário decide

codificar cada semestre no formato AAAA/S, isto é, usando quatro dígitos para informar o ano e, separado por uma barra, mais um dígito para informar se é o primeiro ou o segundo semestre dentro do ano. O ano de 2005, por exemplo, divide-se nos semestres 2005/1 e 2005/2. Para suportar datas neste formato com o modelo TVX, basta que o usuário forneça uma implementação da função *UserCompare* que opere sobre este tipo de rótulo. Tal função pode ser como vista na Figura 4.1. Através deste teste fornecido pelo usuário, do teste de igualdade de *strings* e dos operadores booleanos da linguagem XPath, o modelo TVX consegue implementar todas as funções e operadores de comparação de instantes e intervalos mostrados anteriormente.

Retomando a apresentação das funções definidas para a linguagem de consulta, apresentamos a seguir as funções responsáveis pela extração de informações próprias do modelo, relativas ao conteúdo dos documentos:

- *id(object)*: retorna o identificador global do elemento ou nodo de texto passado como parâmetro;
- *ivt(object)*: retorna o tempo de validade inicial do objeto especificado;
- *fvt(object)*: retorna o tempo de validade final do objeto especificado;
- *itt(object)*: retorna o tempo de transação inicial do objeto especificado;
- *ftt(object)*: retorna o tempo de transação final do objeto especificado;
- *vinterval(object)*: retorna o intervalo de validade do objeto especificado;
- *tinterval(object)*: retorna o intervalo de transação do objeto especificado.

Por fim, as funções apresentadas a seguir caem na categoria de funções auxiliares – não adicionam poder de expressão à linguagem, mas facilitam a escrita de certas expressões:

- *interval(lower, upper)*: recebe dois pontos temporais como parâmetro e devolve um intervalo com limite inferior correspondendo a *lower* e limite superior igual a *upper*;
- *snapshot(root)*: projetada para ser usada dentro da cláusula RETURN, devolve a sub-árvore XML com raiz apontada por *root* em formato atemporal, ou seja, sem os rótulos de tempo de transação e de validade inseridos pelo modelo, retornando o documento à sua estrutura original;
- *now()*: retorna o valor de relógio correspondente ao momento atual. Deve-se tomar cuidado para não confundi-la com o literal ‘now’. Considere as expressões `tvx:validity['01/01/05', 'now']` e `tvx:validity['01/01/05', now()]`; ambas selecionam todos os objetos cujo intervalo de validade inicia em 2005, porém a primeira seleciona apenas aqueles cujo tempo de validade final está em aberto, enquanto que a segunda seleciona aqueles válidos até o momento em que a consulta é executada. Supondo, por exemplo, que o momento corrente seja o dia 03 de maio de 2005, um intervalo do tipo [‘20/03/05’, ‘now’] seria selecionado por ambas as expressões, porém o intervalo [‘20/03/05’, ‘26/04/05’], apenas pela primeira.

Devido à grande variedade de funções que foram definidas, pode haver mais de uma maneira de expressar uma mesma condição temporal. Por exemplo, a expressão do primeiro exemplo da seção 4.1, referente à cláusula *tvx:version*, pode ser rescrita como `tvx:version[contains(vinterval(.), interval('01/01/05', '31/01/05'))]`.

4.3 Exemplos de Expressões de Consulta

Na linguagem XQuery, não é exigido que o retorno da expressão de consulta seja um documento XML com a mesma estrutura do documento consultado; de fato, não é requerido sequer que o retorno da consulta seja um documento XML válido (não há obrigatoriedade, por exemplo, de um elemento raiz englobando todo o resultado da consulta). A maneira mais apropriada de encarar o resultado de uma expressão de consulta em XQuery, na verdade, é como uma seqüência de fragmentos de uma árvore XML.

Isso ocorre porque, da forma como as expressões FOR-LET-WHERE-RETURN são construídas, é dada ao usuário a liberdade de formatar o resultado da consulta da maneira que desejar. O mesmo é válido para a extensão de XQuery feita para o modelo TVX: o resultado da consulta pode vir formatado de acordo com a estrutura original do documento, ou de acordo com a codificação interna do modelo TVX, ou ainda em um outro formato qualquer – variando conforme a necessidade do usuário.

Como exemplos de expressões de consulta, considere em um primeiro momento as seguintes expressões, correspondentes aos exemplos de consultas dados na seção 3.6:

- recuperar a visão corrente sobre o momento corrente (resultado na Figura 3.23a):

```
FOR $layout IN document("layout.xml")/layout
RETURN { snapshot($layout) }
```

- recuperar a visão corrente sobre o dia 12/06/04 (resultado na Figura 3.23b):

```
FOR $layout IN document("layout.xml")/tvx:validity['12/06/04',
               '12/06/04']/layout
RETURN { snapshot($layout) }
```

- recuperar a visão do dia 03/06/04 sobre o momento corrente (resultado na Figura 3.25a):

```
FOR $layout IN document("layout.xml")/tvx:rollback['03/06/04']/layout
RETURN { snapshot($layout) }
```

- recuperar a visão do dia 03/06/04 sobre o dia 06/06/04 (resultado na Figura 3.25b):

```
FOR $layout IN document("layout.xml")/tvx:rollback['03/06/04']/
               tvx:validity['06/06/04', '06/06/04']/layout
RETURN { snapshot($layout) }
```

Para os demais exemplos, considere um documento `empresa.xml`, com a estrutura vista na Figura 4.2. O documento é dividido primariamente nos departamentos que compõe a empresa, e dentro de cada departamento, há os registros dos diversos funcionários que o integram. As consultas que se deseja realizar sobre este documento são:

```
<empresa>
 <departamento nome="Financas">
  <empregado>
   <nome> João </nome>
   <salario> R$ 1.500,00 </salario>
  </empregado>
  ...
 </departamento>
 ...
</empresa>
```

Figura 4.2: Documento dos exemplos de expressões de consulta

- recuperar a visão corrente sobre a história do salário do empregado João, junto com o intervalo de validade para cada valor (um exemplo da estrutura resultante é visto abaixo da consulta):

```
<salarios> {
 FOR $s IN document("empresa.xml")/tvx:validity[EVER]//empregado[nome="João"]/salario
 RETURN <salario De="{ivt($s/text())}" Ate="{fvt($s/text())}">
         { $s/text() }
        </salario>
} </salarios>
```

```
<salarios>
 <salario De="01/03/04" Ate="30/08/04"> R$ 1.500,00 </salario>
 <salario De="01/09/04" Ate="now"> R$ 2.000,00 </salario>
 ...
</salarios>
```

- retornar, conforme a visão corrente em 31/12/03, a quantidade de empregados vinculados a cada departamento:

```
<departamentos> {
 FOR $d IN document("empresa.xml")/tvx:rollback['31/12/03']//departamento
 LET $e := $d/empregado
 RETURN <departamento nome="{$d/@nome}" qtdeemp="{count($e)}"/>
} </departamentos>
```

- na versão raiz do documento encontrar, para cada empregado, todos os departamentos em que trabalhou, e durante qual período:

```
<empregados> {
 FOR $e1 IN distinct-values(document("empresa.xml")/tvx:version[root()]//empregado)
 RETURN <empregado nome="{$e1/nome/text()}">
         { FOR $d IN document("empresa.xml")/tvx:version[root()]//departamento
           LET $e2 := $d/empregado
           WHERE id($e2) = id($e1)
           RETURN <departamento nome="$d/@nome" De="{ivt($e2)}" Ate="{fvt($e2)}"/> }
        </empregado>
} </empregados>
```

- encontrar os dados de todos os empregados que trabalharam no departamento "Recursos Humanos" durante o primeiro semestre de 2002:

```
<empregados> {
FOR $e IN document("empresa.xml")/tvx:validity['01/01/02', '30/06/12']//
          departamento[@nome="Recursos Humanos"]/empregado
RETURN { snapshot($e) }
} </empregados>
```

- descobrir quando os dados do empregado José foram inseridos na base de dados:

```
FOR $e IN document("empresa.xml")/tvx:validity[EVER]//empregado[nome/text()="José"]
RETURN min(itt($e))
```

4.4 Considerações Finais

Neste capítulo, foi definida uma linguagem de consulta para o modelo TVX, apresentado no capítulo 3. A linguagem resultante estende as construções da linguagem XQuery e as expressões de caminho da linguagem XPath para possibilitar o acesso ao conteúdo do documento XML, com relação aos critérios de bitemporalidade e de versionamento. Através da linguagem proposta neste capítulo, a qual torna os detalhes de implementação do modelo TVX transparentes para o usuário, é possível consultar tanto a história presente quanto a história passada de todas as versões da base de dados.

O capítulo seguinte apresenta uma linguagem muito similar àquela desenvolvida neste capítulo, pois se trata também de uma extensão à linguagem XQuery, a qual inclusive reutiliza os conceitos definidos neste capítulo para construção de suas expressões. O objetivo desta próxima linguagem, contudo, não é o de executar leituras ao conteúdo do documento, mas sim escritas, tratando-se portanto de uma linguagem de manipulação do conteúdo.

5 LINGUAGEM DE MANIPULAÇÃO DE DADOS

O capítulo anterior mostrou uma linguagem de consulta para ser usada em conjunto com a modelagem proposta no capítulo 3. A modelagem apresentada baseou-se na linguagem XQuery, e apresentou extensões das expressões de caminho XPath para acessar informações temporais e versionadas dentro do documento XML. Este capítulo apresenta uma linguagem dedicada à alteração do conteúdo dos documentos dentro do modelo TVX, a qual também tem por base a linguagem XQuery. O objetivo desta linguagem é permitir ao usuário expressar operações de modificação ao conteúdo do documento sobre o esquema original do mesmo, tornando a codificação XML adotada pelo modelo TVX transparente para o usuário. Na construção de expressões desta DML são válidas todas as novas funcionalidades apresentadas no capítulo anterior.

O restante do capítulo está dividido da seguinte forma: a seção 5.1 apresenta, sem entrar em detalhes, a sintaxe da linguagem de manipulação de dados que foi definida para se trabalhar com o modelo TVX. Em seguida, as seções 5.2 a 5.4 desmembram esta linguagem em grupos de operações com propósitos comuns, detalhando a sintaxe e a semântica de cada operação. Por fim, a seção 5.5 apresenta exemplos de expressões na linguagem proposta, e a seção 5.6 discute maneiras de verificar se as modificações promovidas por uma expressão de alteração não tornam o documento inválido.

5.1 Sintaxe das Expressões de Atualização

```
UpdateExpression → CreateDocument | VersionManagement | UpdateContent
CreateDocument → CREATE(RootName[, SchemaURI]) [FROM(DocURI, ivt, fvt)]
                        [AS UpdateContent]
VersionManagement → DERIVE(vid, NewName) [AS UpdateContent] |
                    SETCURRENT(vid)
UpdateContent → FOR $binding IN XPath-expr, ...
                LET $binding := XPath-expr, ...
                WHERE predicate, ...
                UPDATE $binding { UpdOp {, UpdOp}* }
UpdOp → InsEl(name, ivt, fvt) | InsElBef($child, name, ivt, fvt) |
        InsElAft($child, name, ivt, fvt) | InsAt(name, str, ivt, fvt) |
        InsTxt(str, ivt, fvt) | InsTxtBef($child, str, ivt, fvt) |
        InsTxtAft($child, str, ivt, fvt) |
        Del($child, ivt) |
        Upd($child, str, ivt) | UpdIVT(ivt) | UpdFVT(fvt) |
        Mov($srcref, ivt) | MovBef($srcref, $child, ivt) | MovAft($srcref, $child, ivt)
        | UpdateContent
```

Figura 5.1: Gramática das expressões de atualização

As expressões da linguagem de atualização de documentos podem ser divididas em três grupos: operações para criação de documentos, operações para gerenciamento da hierarquia de versões e operações para gerenciamento do conteúdo do documento. Estas últimas são as de estrutura mais complexa, seguindo uma sintaxe similar à das expressões FOR-LET-WHERE-RETURN da linguagem XQuery padrão; de fato, a

principal diferença está na substituição da cláusula RETURN por uma cláusula UPDATE, na qual estão contidos os comandos de DML propriamente ditos. Uma visão geral da linguagem de manipulação proposta pode ser vista na Figura 5.1. O funcionamento das cláusulas FOR, LET e WHERE é idêntico ao do padrão XQuery, logo o detalhamento de sua sintaxe é omitido. O significado de cada uma dessas construções será apresentado nas seções que seguem, agrupadas de acordo com o propósito que possuem.

5.2 Operação de Criação de Documentos

A sintaxe da operação de criação de documentos é como vista abaixo:

```
CREATE(RootName[, SchemaURI]) [FROM(DocURI, ivt, fvt)] [AS UpdateContent]
```

Esta operação cria um novo documento XML já ajustado ao formato TVX. O esquema para este documento é a ele associado através do parâmetro opcional *SchemaURI*. Como o próprio nome indica, este parâmetro contém uma *string* que indica a localização de um arquivo contendo a descrição, em XML Schema, do documento sendo criado. Se o parâmetro for omitido, subentende-se que o documento está sendo criado sem vinculação a um esquema pré-definido. Caso o recurso seja localizado e a descrição em XML Schema não contenha erros sintáticos, o esquema nele contido será anexado ao conteúdo do novo documento. Dessa forma, somente durante a criação do documento será necessária a existência de um recurso externo contendo o esquema do mesmo; depois de criado, como o conteúdo do documento incluirá a definição do esquema, o arquivo utilizado para sua criação deixa de ser necessário, podendo ser movido para outra localização ou mesmo apagado sem conseqüências para o documento gerado. Qualquer erro relacionado à localização ou ao conteúdo do documento que contém o esquema deve abortar o processo de criação, e o usuário deve ser alertado sobre o ocorrido.

O primeiro parâmetro, *RootName*, contém a *string* que será associada como nome à versão raiz do documento sendo criado. Como este nome ficará guardado dentro de um atributo na codificação resultante, não pode apresentar aspas em seu conteúdo. O identificador para esta versão raiz será atribuído automaticamente quando da criação do documento.

Através da cláusula opcional FROM, é possível especificar que o estado inicial do documento deve ser gerado a partir da conversão de um documento XML em formato convencional para o formato do modelo TVX. A localização do arquivo fonte é dada pelo parâmetro *DocURI*, e os parâmetros *ivt* e *fvt* recebem os tempos de validade inicial e final que serão atribuídos a todos os nodos da árvore XML após a conversão. Na ausência da cláusula FROM, o estado inicial gerado encontra-se inicialmente vazio. Por fim, a expressão *UpdateContent* corresponde a um conjunto de operações responsáveis por gerar o conteúdo do documento XML – tipicamente, uma seqüência de operações de inserção de elementos, atributos e nodos de texto. Estas operações são responsáveis por preencher inteiramente a árvore XML inicial quando a cláusula FROM está ausente, ou por realizar modificações na árvore original quando a cláusula está presente. A sintaxe e o funcionamento destas operações serão apresentados mais adiante, na seção 5.4.

5.3 Operações de Gerenciamento da Hierarquia de Versões

São duas as operações que trabalham sobre o versionamento: *Derive* e *SetCurrent*. Começando pela mais simples, a sintaxe da operação *SetCurrent* é a seguinte:

```
SETCURRENT(vid)
```

O propósito desta operação é simplesmente definir qual a versão corrente do documento atual. Como foi definido no capítulo 3, a versão corrente serve como alvo implícito das operações de consulta e modificação do documento, quando nenhuma versão específica é requerida. A indicação da versão corrente não restringe, portanto, que a mesma será a única passível de consultas e modificações, mas sim indica que este será o comportamento padrão. Não há, também, qualquer restrição quanto a quais versões dentro da hierarquia podem ser ajustadas como versão corrente. O parâmetro *vid* corresponde ao identificador da versão que está recebendo o *status* de versão corrente. A operação *Derive*, por sua vez, apresenta a seguinte sintaxe:

```
DERIVE(vid, NewName) [AS UpdateContent]
```

A operação *Derive* corresponde à operação de derivação de versões descrita na seção 3.3. Através dela, cria-se uma nova versão derivada a partir da versão indicada pelo parâmetro *vid*. Este parâmetro indica, portanto, o identificador da versão da qual se está derivando, e não o identificador que a nova versão receberá – assim como o que ocorre com a versão raiz, este é atribuído automaticamente durante o processo de derivação. A versão criada pela operação *Derive* receberá o nome indicado pelo parâmetro *NewName*.

Ao realizar a derivação de uma versão, a visão corrente do conteúdo da versão base é reproduzida fielmente na versão derivada, juntamente com os respectivos tempos de validade inicial e final de cada objeto. Os rótulos de tempo de transação inicial para todos os objetos na versão derivada recebem o valor correspondente ao momento em que a derivação é realizada, e os rótulos de tempo de transação final recebem o valor *now*. Embora não seja obrigatório, muito provavelmente a derivação de uma versão é conseqüência de alguma mudança no documento, logo a versão derivada dificilmente deverá aparecer como uma cópia idêntica da versão original. Por essa razão, é possível especificar na cláusula *UpdateContent* um conjunto de operações que modificam o conteúdo reproduzido na versão derivada, visando corrigir as diferenças entre a versão base e a nova versão. As diversas operações que podem aparecer dentro desta cláusula são apresentadas na seção que segue.

5.4 Operações de Gerenciamento do Conteúdo do Documento XML

A sintaxe proposta para as operações de gerenciamento do conteúdo da árvore XML é uma modificação em relação ao padrão XQuery; tal como em XQuery, as cláusulas FOR, LET e WHERE estão presentes e servem para selecionar, através de expressões de caminho e de predicados booleanos, um conjunto de nodos de interesse dentro da árvore XML. Ao contrário de XQuery, no entanto, onde a expressão é encerrada por uma cláusula RETURN que devolve o conteúdo dos nodos selecionados, nesta modificação a expressão é encerrada por uma cláusula UPDATE, a qual contém uma série de operações primitivas que agem sobre os nodos selecionados, modificando seu conteúdo (Figura 5.2).

```
FOR $binding IN XPath-expr, ...
LET $binding := XPath-expr, ...
WHERE predicate, ...
UPDATE $binding { UpdOp {, UpdOp}* }
```

Figura 5.2: Estrutura das expressões de modificação de conteúdo

A linguagem resultante é, na verdade, uma modificação de uma proposta existente (TATARINOV, 2001), cujo objetivo é fornecer uma linguagem através da qual seja possível expressar alterações em um documento XML, entretanto sem a preocupação da manutenção de estados antigos. Esta proposta foi, então, estendida para incorporar informações temporais às operações de atualização de conteúdo.

A cláusula UPDATE é seguida por um nome de variável e por uma lista ordenada de operações, contidas entre chaves. O alvo implícito dessas operações é o conjunto de nodos ligados à variável que segue a cláusula UPDATE. Dentro desta lista podem aparecer operações de inserção, de remoção (lógica, e não física), de alteração e de movimentação de sub-árvores do documento XML, bem como uma nova expressão FOR-LET-WHERE-UPDATE – este aninhamento permite escrever uma única expressão de atualização capaz de trabalhar simultaneamente em vários níveis dentro da árvore XML.

5.4.1 Operações de Inserção

As operações de inserção dividem-se em três tipos: inserção de elementos, inserção de atributos e inserção de nodos de texto, identificadas respectivamente pelos prefixos *InsEl*, *InsAt* e *InsTxt*. Estas operações lidam com objetos atômicos; para incluir uma sub-árvore complexa em um documento, é necessário, portanto incluí-la um objeto por vez. É possível, contudo, agrupar todas as operações necessárias dentro da mesma cláusula UPDATE. A Figura 5.3 apresenta as diversas operações de inserção presentes na linguagem. Como pode ser visto na figura, existem variantes para as operações de inserção de elementos e de nodos de texto, cuja diferença será explicada no decorrer desta subseção.

```
InsEl(name, ivt, fvt)
InsElBef($child, name, ivt, fvt)
InsElAft($child, name, ivt, fvt)
InsAt(name, str, ivt, fvt)
InsTxt(str, ivt, fvt)
InsTxtBef($child, str, ivt, fvt)
InsTxtAft($child, str, ivt, fvt)
```

Figura 5.3: Operações de inserção

Apesar de lidarem com objetos diferentes, estas operações possuem certas similaridades. Em primeiro lugar, todas inserem o conteúdo de que tratam como descendente direto do(s) nodo(s) apontado(s) pela variável que segue a cláusula UPDATE. Como qualquer destes tipos de objetos só pode aparecer como descendente de um elemento no documento XML, todas as respectivas operações apresentam a restrição de só serem válidas se houver apenas elementos no conjunto de nodos associados à variável que segue o UPDATE. Todas apresentam, também, dois parâmetros *ivt* e *fvt*, através dos quais se pode especificar, respectivamente, o tempo de validade inicial e o tempo de validade final para o objeto que se deseja inserir. Para o tempo de validade final, é possível utilizar a *string* "now" no lugar de um valor absoluto.

Começando pelas operações de inserção de elementos, todas apresentam um parâmetro *name*, através do qual se deve especificar o nome do elemento que está sendo inserido. Existem duas variantes nas quais se especifica a posição em que o novo

elemento deve ser inserido, como antecessor – através de *InsElBef* – ou sucessor – por meio de *InsElAft* – imediato de um outro objeto que serve como ponto de referência. Este ponto de referência é indicado pela variável *$child*, a qual deve apontar obrigatoriamente para outro nodo de elemento ou de texto que seja descendente imediato do objeto no qual se está tentando inserir o novo elemento – caso contrário, configura-se uma situação de erro e a operação deve ser abortada. Há também uma variante que não especifica posição – *InsEl* – e que portanto não inclui o parâmetro *$child*. Esta variante insere o novo elemento como último filho de seu pai, ou seja, na última posição disponível.

Para a inserção de atributos, não há necessidade de variantes para especificar posição, pois não há ordem dentre os atributos que descendem de um mesmo elemento. Logo há uma única operação deste tipo, a qual também apresenta um parâmetro *name*, indicativo do nome do atributo que está sendo inserido, bem como um parâmetro *str*, onde se deve especificar a *string* de conteúdo do atributo que está sendo criado. A operação de inserção de atributos deve ser abortada caso se tente inserir um atributo com o mesmo nome de outro já presente no elemento alvo.

A inserção de nodos de texto é bastante similar à inserção de elementos, inclusive apresentando operações com nomes e funcionamento similares – a saber, *InsTxt*, *InsTxtBef* e *InsTxtAft*. A grande diferença em relação a seus correspondentes para inserção de elementos é que no lugar do parâmetro *name* há um parâmetro *str*, onde se informa o conteúdo do nodo de texto sendo inserido. A operação de inserção de nodos de texto deve ser abortada se ocasionar a existência de dois nodos de texto contíguos dentre os descendentes diretos de um elemento. Para as operações de inserção de elementos e de nodos de texto, os identificadores globais dos novos objetos são gerados automaticamente no momento de sua criação, portanto não fazem parte da lista de parâmetros. A inserção de atributos não requer a associação destes a identificadores globais, por razões já apresentadas na seção 3.2.

5.4.2 Operação de Remoção

A operação de remoção de conteúdo apresenta a sintaxe mostrada abaixo:

```
Del($child, ivt)
```

O conteúdo apontado por *$child* representa a raiz de uma sub-árvore que deve ser excluída logicamente do documento XML. A raiz desta sub-árvore deve ser um objeto que descenda diretamente da variável que segue a cláusula UPDATE. Não é possível apagar apenas o conteúdo de um atributo, sem apagar o atributo em si, pois um atributo não pode existir sem conteúdo; o mesmo vale para nodos de texto.

Ao contrário das operações de inserção, que lidam com um objeto por vez, a operação de remoção pode apagar do documento um fragmento maior da árvore XML – ou seja, um nodo qualquer juntamente com todos seus descendentes. A operação de remoção não exclui fisicamente os objetos selecionados; na verdade, ela apenas altera os rótulos de tempo de transação final e de tempo de validade final para simular a exclusão do objeto, sem perder o conhecimento sobre o estado em que este fazia parte do documento. Este processo é repetido para o nodo original e recursivamente para todos os seus descendentes. O parâmetro *ivt* serve para especificar o instante de tempo a partir do qual o objeto removido deixa de fazer parte do documento, ou, em outras palavras, o momento a partir do qual a remoção passa a ser válida. Para este parâmetro, pode também ser usado o valor *now* no lugar de um valor absoluto, significando que a remoção tem efeito imediato. Caso se deseje remover um elemento contido entre dois

nodos de texto, o que os deixaria contíguos na árvore XML, o segundo nodo de texto é excluído e seu conteúdo é anexado ao conteúdo do primeiro.

5.4.3 Operações de Atualização

As operações de atualização podem ser de dois tipos: atualização de conteúdo e atualização de rótulos temporais. Todas devem operar sobre descendentes do objeto apontado pela variável que segue o UPDATE. A atualização de conteúdo age sobre as *strings* contidas em atributos e nodos de texto; sua sintaxe é `Upd($child, str, ivt)`. Novamente, esta operação inclui um parâmetro *ivt*, correspondente ao momento em que a modificação começa a valer no mundo real, o que pode ocorrer assim que a transação for processada se for usado o valor *now*. O valor a ser assumido pelo objeto apontado por *$child* é indicado pelo parâmetro *str*.

A atualização de rótulos temporais opera apenas sobre os rótulos de tempo de validade, pois a gerência dos rótulos de tempo de transação é de responsabilidade exclusiva da base de dados. São duas operações nesta categoria, uma para modificar o tempo de validade inicial de um objeto – `UpdIVT(ivt)` – e outra para alterar seu tempo de validade final – `UpdFVT(fvt)`. Para cada uma delas, o tempo de validade do objeto apontado pela variável ligada à cláusula UPDATE é ajustado de acordo com o novo valor indicado pelo parâmetro fornecido. A alteração do tempo de validade de um objeto pode provocar alterações em cascata em seus descendentes, tal como ocorre com a operação de remoção.

5.4.4 Operações de Movimentação

As operações de movimentação deslocam uma sub-árvore inteira em um documento XML para uma nova posição dentro do mesmo documento. Esta nova localização pode ser uma posição diferente dentro do mesmo pai – como a mudança na posição do elemento *Banner* no exemplo do capítulo 3 – ou uma posição onde o ascendente da raiz da sub-árvore sendo movida seria outro. A raiz da sub-árvore deve ser um elemento ou nodo de texto, não podendo, portanto, ser um atributo. As diversas variantes para a operação de movimentação são mostradas na Figura 5.4.

```
Mov($srcref, ivt)
MovBef($srcref, $child, ivt)
MovAft($srcref, $child, ivt)
```

Figura 5.4: Operações de movimentação

As diversas operações de movimentação deslocam a árvore com raiz em *$srcref* para uma nova posição, como descendente do elemento apontado pela variável que segue a cláusula UPDATE. Ou seja, as operações de movimentação especificam apenas a origem da sub-árvore, sendo que o destino está implícito. Como a sub-árvore sendo movida pode ser descendente de um objeto qualquer, não há a restrição de que *$srcref* seja um descendente do objeto especificado pela cláusula UPDATE. Além do usual parâmetro *ivt* para indicar o momento em que a transformação passa a ser válida, as operações podem incluir também um parâmetro *$child* para indicar um ponto de referência no local onde a sub-árvore deve ser inserida, de forma a especificar a nova posição da mesma com relação a este ponto de referência – analogamente ao que ocorre com as variantes da operação de inserção. Caso não seja especificado um ponto de referência, a operação *Mov* posiciona a sub-árvore na última posição disponível, tal como as operações *InsEl* e *InsTxt*.

5.5 Exemplos de Expressões da Linguagem de Modificação de Dados

A seguir, serão vistos alguns exemplos de expressões envolvendo os diversos operadores apresentados nas seções anteriores. Estas expressões serão construídas sobre o mesmo exemplo do capítulo 3. Em primeiro lugar, considere que se deseja criar o documento das Figuras 3.1 e 3.2, e que a descrição em XML Schema para o mesmo se encontra em http://www.inf.ufrgs.br/~rgsantos/layout.xsd. A expressão vista na Figura 5.5 gera o estado inicial do documento, associando-o ao esquema desejado e criando o conteúdo a partir da raiz.

```
CREATE('Principal', 'http://www.inf.ufrgs.br/~rgsantos/layout.xsd') AS
FOR $root IN /
UPDATE $root { InsEl('Layout', '01/06/04', 'now'),
               FOR $layout IN $root/layout
               UPDATE $layout { InsEl('Banner', '01/06/04', 'now'),
                                FOR $banner IN $layout/banner
                                UPDATE $banner { InsTxt('RGS Industries', '01/06/04',
                                                        'now') },
                                InsEl('Link', '01/06/04', 'now'),
                                FOR $link IN $layout/link
                                UPDATE $link { InsAt('Color', 'Cyan', '05/06/04', 'now')
                              }}}
```

Figura 5.5: Exemplo de expressão de criação de um documento

Na expressão acima, o comando CREATE gera um documento inicialmente vazio, e o conteúdo do mesmo é preenchido inteiramente pelas operações de inserção contidas na cláusula UPDATE. Supondo que haja um arquivo XML contendo o estado inicial a ser gerado em formato XML convencional, tal como visto na figura 3.1, e que o mesmo esteja localizado em http://www.inf.ufrgs.br/~rgsantos/layout.xml, é possível utilizar a cláusula FROM para adaptar o documento ao modelo TVX através de conversão de formatos, ao invés de inserir um a um os objetos que o constituem. A expressão que perfaz esta operação é mostrada na Figura 5.6. Todos os objetos do documento resultante apresentarão um intervalo de tempo de validade iniciando em 01/06/04 e com final em aberto, conforme definido na cláusula FROM, com exceção do atributo *Color*, cujo tempo de validade inicial é modificado através da cláusula UPDATE.

```
CREATE('Principal', 'http://www.inf.ufrgs.br/~rgsantos/layout.xsd')
FROM('http://www.inf.ufrgs.br/~rgsantos/layout.xml', '01/06/04', 'now') AS
FOR $color IN /layout/link/@color
UPDATE $color { UpdIVT('05/06/04') }
```

Figura 5.6: Exemplo de expressão de criação de um documento através de conversão

Supondo que à versão criada por qualquer um destes comandos tenha sido atribuído um *VersionID* igual a 1 (o que pode ser descoberto através da linguagem de consulta do capítulo anterior), é possível derivar outras versões a partir dela. Por exemplo, para incluir a versão de Verão mostrada sumariamente na Figura 3.21, a qual possui um elemento *Link* diferenciado, basta usar a expressão mostrada na Figura 5.7.

```
DERIVE('1', 'Verao') AS
FOR $layout IN /layout
UPDATE $layout { InsEl('Link', '20/12/04', '20/03/05'),
                 FOR $link IN $layout/link[last()]
                 UPDATE $link { InsTxt('Click here for our Summer schedule', '20/12/04',
                                       '20/03/05') } }
```

Figura 5.7: Exemplo de derivação de versões

Para ajustar a versão de Ano Novo, cujo identificador é 1.1.2, como versão corrente, basta usar o comando `SETCURRENT('1.1.2')`. Por fim, a expressão na Figura 5.8 executa as transformações ao conteúdo do documento mostradas nas Figuras 3.3 e 3.4.

```
FOR $la IN document("layout.xml")/tvx:version[1]/layout, $b IN $la/banner
UPDATE $la { FOR $li IN $la/link, $c IN $li/@color, $t IN $li/text()
             UPDATE $li { Del($c, '15/06/04'), Upd($t, 'About Us', '15/06/04') },
             InsEl('Link', '15/06/04', 'now'),
             FOR $li IN $la/link[2]
             UPDATE $li { InsTxt('Contact', '15/06/04', 'now') },
             Mov($t, '15/06/04') }
```

Figura 5.8 Exemplo de alteração de conteúdo

5.6 Verificação de Consistência do Documento XML

Em um documento XML, assim como em uma base de dados qualquer, é possível que uma tentativa de modificação do conteúdo acabe por gerar um estado inconsistente, por violar uma ou mais regras de integridade associadas aos dados sendo modificados. No caso de documentos XML convencionais, as verificações de integridade referem-se a tentativas de modificação à estrutura do documento, as quais devem ser limitadas pelo esquema a ele associado. Em se tratando de documentos XML temporais, devem ser levadas em consideração também as restrições que dizem respeito aos rótulos temporais envolvidos. Esta seção discute estes dois tipos de regras no contexto deste trabalho, e no final apresenta métodos para executar a verificação de consistência em virtude da ocorrência de modificações no documento.

5.6.1 Consistência dos Rótulos Temporais

Inserções de objetos e modificações nos valores de rótulos temporais devem obedecer à restrição de que os intervalos temporais associados a um objeto qualquer dentro do documento XML devem estar contidos nos correspondentes intervalos temporais do objeto do qual este descende imediatamente na árvore XML. Sendo assim, definem-se as seguintes regras de integridade temporal, adaptadas de (MORO, 2001):

- o tempo de validade inicial de um objeto deve ser maior ou igual ao tempo de validade inicial de seu objeto pai, isto é, um objeto que é parte de um todo não pode começar a existir sem que o todo já exista;
- o tempo de validade final de um objeto deve ser maior ou igual ao seu tempo de validade inicial (pode ser igual pois o objeto pode ser válido por apenas um *chronon*) e maior ou igual ao tempo de validade final dos objetos nele contidos, isto é, a existência dos objetos que fazem parte de um todo não pode ultrapassar a existência do todo;
- o tempo de transação inicial de um objeto deve ser maior ou igual ao tempo de transação inicial de seu objeto pai, isto é, um objeto não pode ter sido inserido na base de dados antes de se inserir o objeto do qual faz parte;
- o tempo de transação final de um objeto deve ser maior ou igual ao seu tempo de transação inicial (pode ser igual pois o objeto pode ser inserido durante um *chronon* e removido no *chronon* seguinte) e maior ou igual ao tempo de transação final dos objetos nele contidos, isto é, não é possível remover um objeto sem remover todos os seus descendentes;
- para qualquer instante que se execute uma operação de *rollback*, o conjunto de intervalos de tempo de validade correspondentes aos diferentes valores de um

mesmo objeto deve apresentar interseção vazia; isto é, não pode haver dois ou mais valores válidos para o mesmo dado no mesmo instante.

5.6.2 Consistência do Documento XML em Relação ao Esquema

No modelo TVX, a associação do documento XML a um esquema é opcional. Assim sendo, no caso de documentos para os quais não foi definido um esquema, é permitido que a estrutura destes varie livremente, devendo-se apenas levar em consideração a consistência dos rótulos temporais e as regras estruturais convencionais de um documento XML. Estas regras convencionais foram apresentadas juntamente com as próprias operações de modificação, por serem independentes da associação a um esquema, como por exemplo a regra de que não se pode inserir um atributo onde já exista outro de mesmo nome. Quando é definido um esquema para o documento, contudo, deve-se tomar o cuidado para permitir a execução apenas das operações de modificação que mantenham o documento em um estado consistente, de acordo com a estrutura ditada por seu esquema, em qualquer ponto do tempo.

Como exemplo, considere um elemento "Veículo" que apresenta um atributo "Modelo" que foi definido na descrição em XML Schema como sendo obrigatório. Inicialmente, ambos são válidos de t_1 a t_4 (Figura 5.9a). Em geral, uma operação que tente excluir o atributo em um instante t_2 (sendo que t_2 está contido entre t_1 e t_4) não deve ser permitida, pois isto consistiria em uma violação ao esquema. Em outras palavras, de t_2 a t_4 o elemento "Veículo" existiria sem estar vinculado a um atributo "Modelo", o que não é permitido (Figura 5.9b). Considere, contudo, que na mesma cláusula UPDATE a operação de remoção é seguida por uma operação de inserção, a qual insere um novo atributo "Modelo" no mesmo lugar do antigo atributo, válido a partir de t_2 (Figura 5.9c). Neste caso, não há violação no esquema, pois em qualquer ponto durante seu intervalo de vida, o elemento "Veículo" apresenta um atributo "Modelo". Pode ocorrer também uma situação em que o novo atributo não é válido imediatamente após a exclusão do atributo antigo (Figura 5.9d), restando um intervalo que caracteriza a violação do esquema – no caso, de t_2 a t_3.

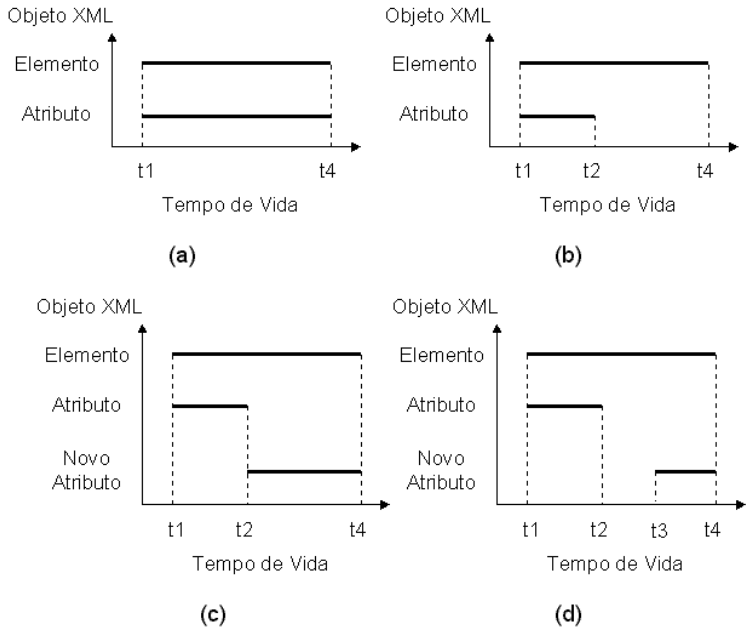


Figura 5.9: Modificações ao conteúdo do documento

Através do exemplo da Figura 5.9c, percebe-se que a verificação de consistência não deve ser feita para cada operação, mas sim para conjuntos de operações. Dessa forma, a cláusula UPDATE deve ser encarada como uma transação, onde se exige apenas que o resultado seja consistente ao final de sua execução completa, e não em passos intermediários. Como conseqüência de considerá-la uma transação, se for verificada uma violação qualquer ao final de sua execução, todas as modificações devem ser desfeitas – mesmo as que não constituem violações de esquema.

Os exemplos anteriores diziam respeito a um tipo de restrição onde deve haver ao menos um objeto de um certo tipo em uma dada posição em qualquer ponto no tempo. Genericamente, as restrições impostas pela descrição em XML Schema determinam quais tipos de objetos e em que quantidade podem aparecer em cada posição na árvore. Um exemplo de restrição ao tipo de objeto que pode aparecer em uma determinada posição é que somente atributos de nome "Modelo" podem aparecer subordinados ao elemento "Veículo"; assim, qualquer tentativa de inserir um atributo de nome diferenciado nesta posição, não importando aqui as relações entre os rótulos temporais, deve ser barrada. Como exemplo de restrição à quantidade máxima de um objeto em uma determinada posição, considere que a propriedade "Modelo" foi modelada como um sub-elemento de "Veículo", ao invés de um atributo, e que o esquema especifica que deve haver no máximo um sub-elemento "Modelo" em um instante qualquer. A Figura 5.10 apresenta uma situação em que se tenta excluir o elemento "Modelo" antigo e substituí-lo por um novo, porém sem ter o devido cuidado com os rótulos temporais. Na situação mostrada na figura a operação deve ser abortada, pois se fosse consolidada existiria um intervalo, de t_2 a t_3, onde o elemento "Veículo" apresentaria dois sub-elementos de nome "Modelo", o que, de acordo com o esquema, não é permitido.

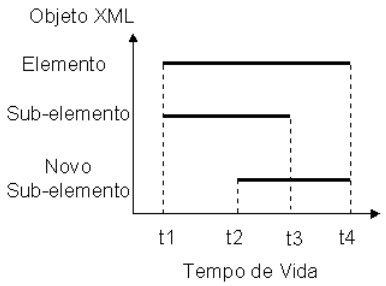


Figura 5.10: Exemplo de violação ao esquema do documento

5.6.3 Estratégias para Verificação de Consistência

Em (KANE; SU; RUNDENSTEINER, 2002) e (SU et al, 2002) são apresentadas duas alternativas para a verificação da consistência de um documento XML após o mesmo sofrer alterações. O primeiro é o método convencional, onde a verificação é postergada até que todas as alterações tenham sido realizadas. Neste método, todas as alterações sobre o documento original são executadas, mas não sobrescrevem o estado original de imediato, gerando ao invés um documento temporário. Este documento é verificado quanto a possíveis violações de consistência, e caso não haja problemas, substitui o documento original.

O método supracitado é potencialmente ineficiente, pois verifica todo o documento, e não somente a parte que foi modificada. Para tentar contornar esta deficiência, os autores propõem um método que verifica a consistência das operações de atualização conforme estas são executadas. Isto é feito através da rescrita das expressões de atualização de conteúdo para que as mesmas incluam em seu código sub-consultas à descrição em XML Schema do documento sendo modificado, para verificar a validade

da modificação sendo executada, abortando-a caso viole o esquema. Caso esta abordagem fosse adaptada para o modelo TVX, a mesma tiraria proveito do fato da descrição em XML Schema estar incluída junto ao conteúdo do documento.

Experimentos executados pelos próprios autores, contudo, demonstram que esta técnica é na verdade mais lenta do que validar o documento modificado a partir do zero. Outra limitação da técnica proposta é que lida com uma única operação por vez, não sendo capaz de lidar um conjunto de operações que deva ser encarado como uma transação. Em função desta limitação, a primeira abordagem – onde a verificação de validade é postergada até o fim da execução das modificações – é mais apropriada para o uso com o modelo TVX, pois na seção 5.6.2 foi visto que, para o funcionamento correto das operações de modificação dentro do modelo TVX, é necessário que as cláusulas UPDATE sejam tratadas como transações. Em uma transação, é requerido que o documento esteja em um estado consistente apenas ao final da execução da mesma, e não necessariamente em seus passos intermediários; outra propriedade de transações é que todas as operações nelas contidas devem ocorrer com sucesso, ou todas devem ser abortadas.

5.7 Considerações Finais

Este capítulo apresentou uma linguagem de manipulação de dados para o modelo TVX, a qual pode ser vista principalmente como uma modificação da linguagem XQuery. Através da DML mostrada, a qual inclui operações para criação de documentos e para manipulação tanto do aspecto bitemporal quanto do aspecto de versionamento, é possível escrever expressões que alterem o conteúdo do documento sem se preocupar com a codificação interna empregada pelo modelo TVX.

Com este capítulo, conclui-se a apresentação do modelo TVX. O capítulo seguinte submete o modelo definido a um estudo de caso envolvendo uma aplicação real, analisando a adequação do modelo TVX aos requisitos da aplicação alvo e comparando os resultados obtidos com um método alternativo para o armazenamento do histórico de modificações em um documento XML.

6 ESTUDO DE CASO

Este capítulo apresenta a modelagem de uma aplicação real através do modelo TVX, apresentado nos capítulos anteriores. O estudo de um sistema real tem por objetivo analisar o modelo quanto a possíveis falhas e incompletudes, visando garantir a aplicabilidade e a adequação do mesmo.

O restante deste capítulo está organizado como segue. A seção 6.1 descreve a aplicação alvo, mostrando qual seu propósito e como organiza os dados. A seção 6.2 mostra como o modelo TVX satisfaz os requisitos de controle temporal exigidos pela aplicação, e a seção 6.3 compara a metodologia empregada atualmente pela aplicação alvo com a codificação segundo o modelo TVX, em termos da demanda por espaço de armazenamento.

6.1 Descrição da Aplicação

A aplicação alvo deste estudo de caso é um módulo de consulta ao texto da Constituição da República Federativa do Brasil, acessível através do *site* do Senado Federal (http://www.senado.gov.br). A necessidade de modelos temporais para armazenamento de textos legais em XML é um problema de reconhecida importância, abordado em trabalhos como (GRANDI et al, 2003). O sistema estudado permite consultas não somente ao texto atual da Constituição, como também a todos os estados passados do mesmo, resultantes das diversas emendas constitucionais promulgadas desde a criação da Constituição atual, em 05 de outubro de 1988.

O texto da constituição é armazenado em formato XML, sendo a partir deste convertido via um conjunto de regras XSLT (W3C 1999b) para o formato HTML, para visualização em um navegador *Web*. Atualmente, o método empregado para armazenar as informações temporais em formato XML é o *snapshot collection*, apresentado na seção 2.3. Em outras palavras, para cada estado da Constituição existe um arquivo XML contendo seu texto na íntegra; da forma como está implementado atualmente, o sistema permite consultas somente ao texto integral da Constituição.

Como já foi discutido no capítulo 2, embora este método seja o mais eficiente quando se trata da velocidade de recuperação de informações passadas, é também o mais ineficiente em termos do espaço requerido para o armazenamento da informação temporal. Isto acontece porque todo o texto da constituição é replicado a cada vez que uma modificação ocorre, mesmo que apenas uma pequena porção do texto seja afetada. Como exemplo, considere o texto da Emenda Constitucional de Revisão n.º 5, mostrado na Figura 6.1.

EMENDA CONSTITUCIONAL DE REVISÃO Nº. 5, DE 1994

A MESA DO CONGRESSO NACIONAL , nos termos do art. 60 da Constituição Federal, combinado com o art. 3º do Ato das Disposições Constitucionais Transitórias, promulga a seguinte emenda constitucional:

Art. 1º No art. 82, fica substituída a expressão "cinco anos" por "quatro anos".
Art. 2º Esta Emenda Constitucional entra em vigor no dia 1º de janeiro de 1995.

Brasília, 7 de junho de 1994

Mesa do Congresso Nacional
HUMBERTO LUCENA
Presidente
ADYLSON MOTTA
1º Vice-Presidente
LEVY DIAS
2º Vice-Presidente
WILSON CAMPOS
1º Secretário
NABOR JÚNIOR
2º Secretário
AÉCIO NEVES
3º Secretário
NILSON WEDEKIN
4º Secretário

Figura 6.1: Emenda Constitucional de Revisão n.º 5

Esta emenda altera apenas um artigo da Constituição, porém com sua promulgação um novo documento é gerado, o qual contém, além do texto corrigido do artigo 82, todos os demais artigos cujo texto não foi modificado pela emenda. Para se ter uma idéia do tamanho da sobrecarga que isto implica, em seu estado atual a Constituição apresenta duzentos e cinqüenta artigos (este estudo de caso leva em consideração todas as modificações ocorridas até a promulgação da Emenda Constitucional n.º 45, de 08 de dezembro de 2004).

O texto da Constituição é dividido em três partes: preliminar, normativa e final. A parte preliminar contém apenas a epígrafe, a ementa e o preâmbulo do documento; a parte final contém local e data da promulgação e a seção de assinaturas; o conjunto de leis propriamente dito fica, portanto, na parte normativa. O conteúdo desta parte segue uma estruturação hierárquica – sendo, portanto, ideal para uma representação em formato XML – sendo dividido em títulos, capítulos, seções, subseções, artigos, parágrafos, incisos e alíneas.

No topo da hierarquia em que se divide a parte normativa, estão os títulos. Cada título é identificado por um número e um nome, e possui um cabeçalho de texto. Os títulos podem conter artigos, ou podem se subdividir em capítulos. Cada capítulo possui a mesma estrutura de um título: é identificado por um número e um nome, possui um cabeçalho de texto e pode ou conter artigos ou subdivisões denominadas seções. A estrutura de uma seção é também a mesma dos títulos e capítulos, sendo que suas subdivisões recebem o nome de subseções; estas últimas não podem ser subdivididas, podendo conter somente artigos. Todas estas estruturas servem, portanto, apenas para agrupar os artigos que tratam de um mesmo assunto, pois é neles que reside o conteúdo propriamente dito. A Figura 6.2 mostra um fragmento da divisão da Constituição em títulos, capítulos, seções e subseções.

```
Título I - Dos Princípios Fundamentais
Título II - Dos Direitos e Garantias Fundamentais
  Capítulo I - Dos Direitos e Deveres Individuais e Coletivos
  Capítulo II - Dos Direitos Sociais
  Capítulo III - Da Nacionalidade
  Capítulo IV - Dos Direitos Políticos
  Capítulo V - Dos Partidos Políticos
Título III - Da Organização do Estado
  Capítulo I - Da Organização Político-Administrativa
  ...
  Capítulo V - Do Distrito Federal e dos Territórios
    Seção I - Do Distrito Federal
    Seção II - Dos Territórios
  Capítulo VI - Da Intervenção
  Capítulo VII - Da Administração Pública
    Seção I - Disposições Gerais
    ...
    Seção IV - Das Regiões
  ...
Título IV - Da Organização dos Poderes
  Capítulo I - Do Poder Legislativo
    Seção I - Do Congresso Nacional
    ...
    Seção VIII - Do Processo Legislativo
      Subseção I - Disposição geral
      Subseção II - Da Emenda à Constituição
      Subseção III - Das Leis
...
```

Figura 6.2: Divisão da Constituição em títulos, capítulos, seções e subseções

Cada artigo é identificado por um número e é constituído por um bloco de texto. Este bloco pode ser único ou pode dividir-se em incisos e parágrafos; parágrafos também podem conter incisos como subdivisões, que por sua vez podem subdividir-se em alíneas. A Figura 6.3 mostra um trecho do texto da Constituição, onde se pode ver a estrutura de um artigo. Além do número que o identifica, cada artigo começa por um cabeçalho de texto, que na figura vai desde "Todos são iguais perante a lei..." até "...nos termos seguintes:". Com exceção do cabeçalho, todos os demais trechos de texto são opcionais. As porções de texto identificadas por números romanos (começando por "I – homens e mulheres são iguais..." e indo até "LXXVII – são gratuitas as ações... exercício da cidadania.") são os incisos. As subdivisões dos incisos XXVIII e XXXVIII, identificadas por letras, são denominadas alíneas. Por fim, os trechos precedidos pelo símbolo § e identificados por um número ordinal são os parágrafos. Apesar deste exemplo em particular não o mostrar, cada parágrafo pode conter também diversos incisos.

A Figura 6.4 mostra um trecho da codificação XML empregada atualmente para armazenar o texto da Constituição Brasileira. A organização do texto da Constituição em formato XML toma proveito da hierarquização natural presente em seu conteúdo. Uma descrição em XML Schema da estrutura destes documentos pode ser encontrada no Anexo.

Título II - Dos Direitos e Garantias Fundamentais

Capítulo I - Dos Direitos e Deveres Individuais e Coletivos

Art. 5º Todos são iguais perante a lei, sem distinção de qualquer natureza, garantindo-se aos brasileiros e aos estrangeiros residentes no País a inviolabilidade do direito à vida, à liberdade, à igualdade, à segurança e à propriedade, nos termos seguintes:
I - homens e mulheres são iguais em direitos e obrigações, nos termos desta Constituição;
II - ninguém será obrigado a fazer ou deixar de fazer alguma coisa senão em virtude de lei;
III - ninguém será submetido a tortura nem a tratamento desumano ou degradante;
...
XXVIII - são assegurados, nos termos da lei:
a) a proteção às participações individuais em obras coletivas e à reprodução da imagem e voz humanas, inclusive nas atividades desportivas;
b) o direito de fiscalização do aproveitamento econômico das obras que criarem ou de que participarem aos criadores, aos intérpretes e às respectivas representações sindicais e associativas;
XXIX - a lei assegurará aos autores de inventos industriais privilégio temporário para sua utilização, bem como proteção às criações industriais, à propriedade das marcas, aos nomes de empresas e a outros signos distintivos, tendo em vista o interesse social e o desenvolvimento tecnológico e econômico do País;
...
XXXVIII - é reconhecida a instituição do júri, com a organização que lhe der a lei, assegurados:
a) a plenitude de defesa;
b) o sigilo das votações;
c) a soberania dos veredictos;
d) a competência para o julgamento dos crimes dolosos contra a vida;
...
LXXVII - são gratuitas as ações de habeas corpus e habeas data, e, na forma da lei, os atos necessários ao exercício da cidadania.
§ 1º As normas definidoras dos direitos e garantias fundamentais têm aplicação imediata.
§ 2º Os direitos e garantias expressos nesta Constituição não excluem outros decorrentes do regime e dos princípios por ela adotados, ou dos tratados internacionais em que a República Federativa do Brasil seja parte.

Figura 6.3: Trecho da Constituição

6.2 Adequação do Modelo TVX aos Requisitos da Aplicação

Esta seção discute a aplicabilidade do modelo TVX como método de representação de informações temporais no contexto da aplicação alvo. A discussão inicia pelas limitações do modelo TVX, mostrando como estas não impedem sua plena utilização para modelagem da aplicação alvo, e, em seguida, mostra como o modelo TVX atende às exigências feitas pela aplicação.

O modelo TVX possui limitações, as quais não devem conflitar com os requisitos da aplicação alvo, caso se deseje utilizá-lo. A primeira limitação diz respeito ao esquema do documento: uma vez definido um esquema, o mesmo valerá por toda a história do documento, sem possibilidade de ser alterado posteriormente. Nesse estudo de caso, esta restrição não é um problema, pois o esquema não está sujeito a mudanças. Dito de outra forma, embora o conteúdo do texto da Constituição varie com o tempo, sua estrutura permanece sempre a mesma. Dessa forma, no contexto desta aplicação não é necessário qualquer mecanismo para o gerenciamento da evolução de esquemas.

A segunda limitação do modelo TVX diz respeito à granularidade do versionamento. No modelo TVX, é possível definir versões apenas do documento como um todo; ou seja, não é possível definir versões de partes do documento. Mais uma vez, isto não representa um problema para esta aplicação; neste contexto, definir versões apenas de certos trechos da Constituição corresponderia a se ter diferentes alternativas para as leis fundamentais da União, sendo que em qualquer momento o conjunto de leis deve ser um só. A possibilidade de versionamento do documento inteiro, contudo, encontra serventia nesta aplicação; através dela é possível, por exemplo, controlar os históricos

de duas versões distintas da Constituição, uma referente à Constituição durante o período do Regime Militar e outra referente à Constituição da Nova República.

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!-- <!DOCTYPE norma SYSTEM "norma.dtd">  -->
<norma tipo="CON" numero="001988" complemento="0" ... seq-consolidacao="0">
 <parte-preliminar>
  <epigrafe>Constituição Federal de 1988</epigrafe>
  <ementa>Constituição da República Federativa do Brasil.</ementa>
  <preambulo>
    Nós, representantes do povo brasileiro, ... promulgamos, sob a proteção de Deus, a
    seguinte CONSTITUIÇÃO DA REPÚBLICA FEDERATIVA DO BRASIL.
  </preambulo>
 </parte-preliminar>
 <parte-normativa>
  <titulos>
   <titulo ID="tit_I_">
    <cabecalho ID="tit_I_cabec_">
     <cabecalhotexto>Título  I - Dos Princípios Fundamentais</cabecalhotexto>
    </cabecalho>
    <artigos>
     <artigo ID="art_1_" numero="1">
      <artigotexto ID="art_1_caput_">
       Art. 1º A República Federativa do Brasil, ..., constitui-se em Estado democrático
       de direito e tem como fundamentos:
      </artigotexto>
      <incisos>
       <inciso ID="art_1_inc_I_" numero="I">
        <incisotexto>I - a soberania;</incisotexto>
       </inciso>
       ...
      </incisos>
      <paragrafos>
       <paragrafo ID="art_1_par_1_" numero="1">
        <paragrafotexto>
         Parágrafo único. Todo o poder emana do povo, que o exerce por meio de
         representantes eleitos ou diretamente, nos termos desta Constituição.
        </paragrafotexto>
       </paragrafo>
      </paragrafos>
     </artigo>
     ...
    </artigos>
   </titulo>
   <titulo ID="tit_II_">
    <cabecalho ID="tit_II_cabec_">
     <cabecalhotexto>Título  II - Dos Direitos e Garantias Fundamentais</cabecalhotexto>
    </cabecalho>
    <capitulos>
     <capitulo ID="tit_II_cap_I_">
      <cabecalho ID="tit_II_cap_I_cabec_">
       <cabecalhotexto>Capítulo I - Dos Direitos e Deveres Individuais e Coletivos
       </cabecalhotexto>
      </cabecalho>
      <artigos>
      ...
  </titulos>
 </parte-normativa>
 <parte-final>
  <localdata>Brasília, 5 de outubro de 1988.</localdata>
  <assinatura>
   <entidades>
    <entidade>
     <pessoas>
      <pessoa><nome>Ulysses Guimarães</nome><cargo>Presidente</cargo></pessoa>
      ...
      <pessoa><nome>Virgílio Távora</nome></pessoa>
     </pessoas>
    </entidade>
   </entidades>
  </assinatura>
 </parte-final>
</norma>
```

Figura 6.4: Codificação em XML do texto da Constituição Brasileira

Uma vez estabelecido que as limitações do modelo TVX não violam as necessidades da aplicação, deve-se determinar se os demais requisitos da mesma são atendidos pelo modelo. A princípio, todo o conteúdo da parte normativa está sujeito a modificações – isto não é problema, visto que o modelo TVX é capaz de registrar a evolução tanto de elementos quanto de atributos e nodos de texto. Essas modificações são promovidas através de Emendas Constitucionais; deve ser possível retornar o texto ao estado em que se encontrava anteriormente à aplicação de uma emenda qualquer. Para isto, é necessário utilizar o tempo de transação. Recorrendo novamente à Figura 6.1, vê-se que o texto da Emenda Constitucional de Revisão n.º 5 contém a data em que esta foi assinada; esta data corresponde, pois, ao tempo de transação inicial do novo estado gerado por esta emenda.

Se todas as emendas entrassem em vigor na data de sua publicação, o tempo de transação bastaria para se ter acesso aos diversos estados do documento. Algumas emendas constitucionais, contudo, entram em vigor em uma data posterior à data de sua publicação; a Emenda Constitucional de Revisão n.º 5, por exemplo, inclui em seu texto um artigo explicitando que a mesma "entra em vigor no dia 1º de janeiro de 1995". Esta data corresponde, em realidade, ao tempo de validade inicial das modificações promovidas pela referida emenda ao texto da Constituição. Fica caracterizada, portanto, a necessidade tanto do tempo de transação quanto do tempo de validade. Como o modelo TVX inclui esses dois recursos, conforme demonstrado nos capítulos anteriores, pode-se concluir que o modelo TVX é capaz de atender aos requisitos temporais da aplicação alvo.

6.3 Comparação entre o Modelo TVX e o método *Snapshot Collection*

Esta seção apresenta um experimento comparativo envolvendo o modelo TVX e o método *snapshot collection*. O foco do experimento é a quantidade de espaço de armazenamento requerida por cada uma das técnicas ao longo da evolução do texto da Constituição.

Os dados acerca do método *snapshot collection* foram obtidos em (LIMA, 2005). Neste método, a base de dados é composta por uma série de arquivos XML cuja estrutura é a mesma mostrada na figura 6.4. A cada nova Emenda Constitucional, uma nova cópia do estado mais recente é gerada e anexada à base, após terem sido feitas as devidas correções ao seu conteúdo.

Os dados referentes ao modelo TVX foram obtidos através de um processo semi-automatizado: o estado original do texto da Constituição, contido no primeiro arquivo XML pertencente à base de dados do método *snapshot collection*, foi convertido automaticamente através de uma rotina de *software*, gerando um documento XML como o visto na figura 6.5. A seguir, com base no texto das Emendas Constitucionais, disponíveis também através do site do Ministério da Fazenda, este documento foi alterado manualmente, de forma a refletir apropriadamente as modificações ocorridas em seu conteúdo. Ao contrário do método convencional, no modelo TVX a base de dados resultante é composta por um único arquivo XML; a cada modificação realizada, as alterações são registradas no próprio arquivo, sem perda do estado anterior. Isto é feito através da manipulação adequada dos rótulos temporais disponíveis para cada objeto dentro do arquivo XML, conforme explicado no capítulo 3.

A Tabela 6.1 mostra uma comparação entre o espaço total ocupado pelo modelo TVX e pelo método *snapshot collection* para se armazenar o documento XML contendo o texto da Constituição Brasileira conforme o mesmo sofre alterações.

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<Document Current="1" NextID="8130">
 <Schema> ... </Schema>
 <Version VersionID="1" Name="Principal">
  <Element Name="norma" GlobalID="1">
   <Validity>
    <TS ITT="05/10/88" FTT="now" IVT="05/10/88" FVT="now"/>
   </Validity>
   <Attributes>
    <Attribute Name="tipo">
     <Validity>
      <TS ITT="05/10/88" FTT="now" IVT="05/10/88" FVT="now"/>
     </Validity>
     <String>
      <Value>CON</Value>
      <TS ITT="05/10/88" FTT="now" IVT="05/10/88" FVT="now"/>
     </String>
    </Attribute>
    ...
    <Attribute Name="seq-consolidacao">
     <Validity>
      <TS ITT="05/10/88" FTT="now" IVT="05/10/88" FVT="now"/>
     </Validity>
     <String>
      <Value>0</Value>
      <TS ITT="05/10/88" FTT="now" IVT="05/10/88" FVT="now"/>
     </String>
    </Attribute>
   </Attributes>
   <Content>
    <Element Name="parte-preliminar" GlobalID="2">
     <Validity>
      <TS ITT="05/10/88" FTT="now" IVT="05/10/88" FVT="now"/>
     </Validity>
     <Attributes/>
     <Content>
      <Element Name="epigrafe" GlobalID="3">
       <Validity>
        <TS ITT="05/10/88" FTT="now" IVT="05/10/88" FVT="now"/>
       </Validity>
       <Attributes/>
       <Content>
        <Text GlobalID="4">
         <Validity>
          <TS ITT="05/10/88" FTT="now" IVT="05/10/88" FVT="now"/>
         </Validity>
         <String>
          <Value>Constituição Federal de 1988</Value>
          <TS ITT="05/10/88" FTT="now" IVT="05/10/88" FVT="now"/>
         </String>
         <Previous>
          <Pointer Value="0">
           <TS ITT="05/10/88" FTT="now" IVT="05/10/88" FVT="now"/>
          </Pointer>
         </Previous>
         <Next>
          <Pointer Value="0">
           <TS ITT="05/10/88" FTT="now" IVT="05/10/88" FVT="now"/>
          </Pointer>
         </Next>
        </Text>
       </Content>
       ...
    </Element>
    <Element Name="parte-normativa" GlobalID="9">
     ...
    </Element>
    ...
   </Content>
   <Previous> ... </Previous>
   <Next> ... </Next>
  </Element>
 </Version>
</Document>
```

Figura 6.5: Codificação do texto da Constituição Brasileira segundo o modelo TVX

Tabela 6.1: Demanda de espaço – modelo TVX x *Snapshot Collection*

Data	Espaço Ocupado (em kB)		Data	Espaço Ocupado (em kB)	
	Snapshot Collection	TVX		*Snapshot Collection*	TVX
05/10/88	652	8847	09/12/99	11809	9136
31/03/92	1092	8862	14/02/00	12746	9168
17/03/93	1534	8911	25/05/00	13214	9184
14/09/93	1976	8911	29/05/00	13682	9200
07/06/94	4188	8929	13/09/00	14623	9232
15/08/95	5952	8956	11/09/01	15099	9249
09/11/95	6392	8973	11/12/01	15581	9265
30/04/96	6833	8982	13/12/01	16063	9281
21/08/96	7274	8982	20/12/01	16545	9298
12/09/96	8157	9012	28/05/02	17028	9314
04/06/97	8599	9027	12/06/02	17512	9331
05/02/98	9044	9042	19/12/02	17996	9347
16/02/98	9489	9057	29/05/03	18478	9364
04/06/98	9946	9073	31/12/03	19455	9397
15/12/98	10412	9089	30/06/04	19948	9414
18/03/99	10877	9105	08/12/04	20467	9431
02/09/99	11344	9121	-	-	-

Os resultados da Tabela 6.1 são mostrados também na Figura 6.6. Como se pode ver, com o método *snapshot collection* a base de dados cresce consideravelmente a cada novo estado inserido. A transformação da codificação original do documento para a codificação do modelo TVX, por sua vez, aumenta consideravelmente o espaço necessário para o armazenamento do estado inicial. Isto já era esperado, por duas razões: em primeiro lugar, o crescimento acelerado da base de dados é uma característica comum a todo e qualquer modelo de dados temporal, pois informações nunca são perdidas, e a cada objeto é necessário associar uma série de propriedades para controle da informação temporal. Em segundo lugar, não se pode esquecer que o modelo TVX foi codificado em XML, o qual é puramente textual, e que codificações textuais de estruturas complexas são extensas por natureza.

Com um número suficiente de atualizações, contudo, vê-se que o modelo TVX ultrapassa em muito a eficácia de armazenamento em comparação com o método convencional, pois a partir do primeiro estado toda a informação necessária para o gerenciamento da informação temporal já está presente no modelo TVX, sendo necessário apenas corrigir os fragmentos do documento que sofrem modificações, ao passo que com o método *snapshot collection* cada novo estado introduz uma grande carga redundante na base de dados. Neste exemplo, o método TVX ocupou no final apenas 46% do espaço necessário para o método alternativo; genericamente, contudo, esta diferença entre as demandas de espaço varia com o número de modificações realizadas, bem como com o percentual do documento que sofre alterações entre um estado e outro.

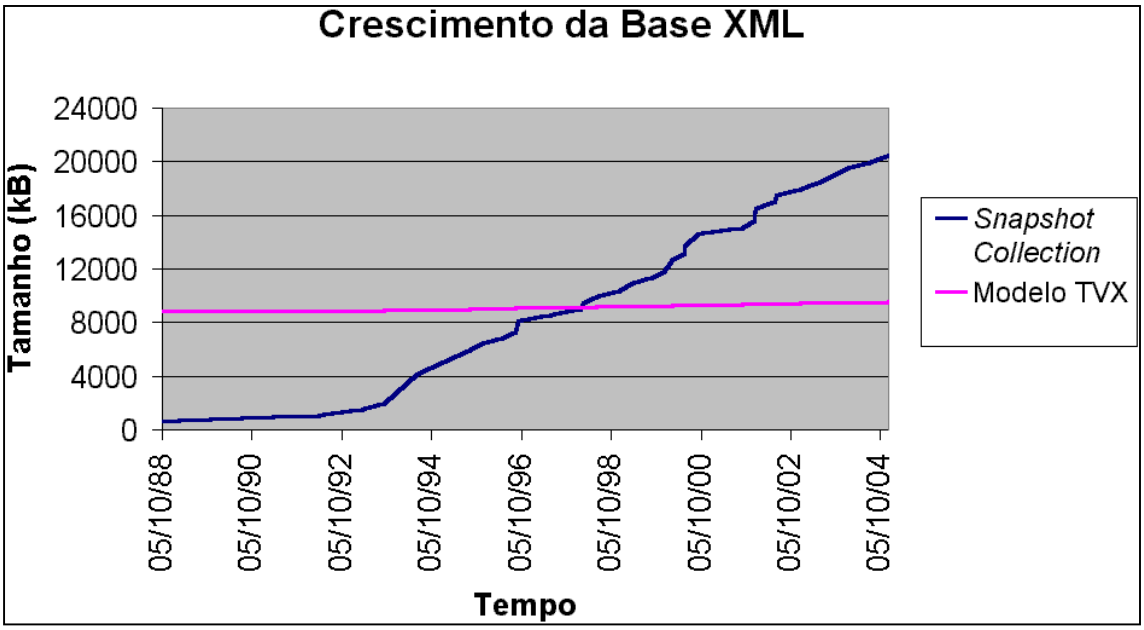


Figura 6.6: Demanda de espaço – modelo TVX x *Snapshot Collection*

6.4 Considerações Finais

Este capítulo apresentou um Estudo de Caso referente à viabilidade do emprego do modelo TVX em uma aplicação real. A aplicação utilizada requeria o armazenamento com registro histórico do texto da Constituição Brasileira, em formato XML. Foi verificado que as características do modelo TVX eram adequadas para suprir as necessidades da aplicação estudada, as quais não conflitavam com as limitações do modelo, mencionadas no capítulo 3.

A implementação com o modelo TVX foi comparada com um método convencional, o *snapshot collection*, o qual vem sendo usado atualmente na aplicação estudada. Como resultado, observou-se que a conversão do formato original dos documentos XML para o formato utilizado no modelo TVX ocasionou um grande aumento no espaço ocupado pela base de dados, quando comparado com a técnica convencional. Contudo, conforme o conteúdo do documento sofria alterações, essa diferença foi sendo gradualmente atenuada e eventualmente o modelo TVX passou a ser mais vantajoso do que a outra abordagem em termos de espaço ocupado. Observou-se também uma tendência ao aumento dessa diferença, pois no experimento realizado a taxa média de crescimento da base de dados para cada novo estado consolidado ficou em torno de 0,2% com o modelo TVX, muito inferior à taxa de 13% medida para o método *snapshot collection*. Conclui-se, portanto, que o modelo TVX encontra aplicabilidade em situações reais, podendo ser inclusive uma solução melhor do que outras empregadas na atualidade.

7 CONCLUSÕES

Este trabalho apresentou o modelo TVX, capaz de gerenciar a evolução do conteúdo de documentos XML através da união dos conceitos de bitemporalidade e versionamento. O diferencial desta proposta em relação a outras existentes está justamente em combinar estas duas técnicas em uma única abordagem. Através dos metadados adicionados aos documentos XML, é possível recuperar visões presentes e passadas sobre o documento em qualquer ponto no tempo, sem a necessidade de recorrer a operações complexas de *backup* e *recovery*, bem como representar diferentes alternativas para a evolução do mesmo.

Tabela 7.1: Comparação entre o modelo TVX e os demais modelos estudados

Modelo	Rótulos			Versões	Foco	Protótipo	Orientado a XML
	TT	TV	Tipo				
TVX	**Sim**	**Sim**	**Intervalo Temporal**	**Sim**	**Dados**	**Não**	**Sim**
TXPath	Não	Sim	Intervalo Temporal	Não	Dados	Não	Sim
Edit-Based UBCC	Não	Não	-	Não	Operações	Sim	Não
Copy-Based UBCC	Não	Não	-	Não	Dados	Sim	Não
RBVM	Não	Não	-	Não	Dados	Sim	Sim
SPaR	Não	Sim	Intervalo Temporal	Não	Dados	Sim	Sim
TXML	Sim	Sim	Intervalo Temporal	Não	Dados	Não	Sim
Xyleme	Sim	Não	Ponto Temporal	Não	Operações	Sim	Sim
Método de Wong & Lam	Sim	Não	Ponto Temporal	Não	Operações	Sim	Sim

A Tabela 7.1 mostra, resumidamente, uma comparação entre e modelo TVX e os demais modelos apresentados na seção 2.3. Para cada modelo, é apresentado que tipo de rótulos temporais utiliza – se utiliza tempo de transação (coluna TT) e/ou tempo de validade (coluna TV), e de que tipo é o rótulo utilizado. Apenas dois modelos utilizam tanto tempo de transação quanto tempo de validade, sendo um deles o modelo TVX. Ademais, além do modelo TVX, nenhuma outra abordagem voltada a documentos semi-estruturados utiliza o conceito de versionamento apresentado na seção 2.2 (embora utilizem o termo "versões" para se referir aos diversos estados de um documento). A

coluna "Foco" separa os modelos entre focados em dados e focados em operações – aqueles focados em dados registram a história armazenando os diversos estados assumidos pelo documento, enquanto que aqueles focados em operações optam por armazenar a história através do registro das operações que transformam um estado em outro. Isto influencia no momento da realização de consultas: nos modelos focados em operações, é necessário realizar uma série de cálculos para recuperar estados passados, enquanto que nos modelos focados em dados basta filtrar os dados indesejáveis. Como ilustrado na tabela através da coluna "Protótipo", a grande desvantagem do modelo TVX em relação a outras abordagens existentes é a inexistência de estudos relativos a questões de implementação. Por fim, dos modelos contemplados na Tabela 7.1, apenas os modelos *Edit-Based UBCC* e *Copy-Based UBCC* são estudos sobre dados semi-estruturados em geral, e não específicos para documentos XML, portanto não tiram proveito de características próprias da linguagem.

A codificação empregada pelo modelo TVX toma também a forma de um documento XML, apresentando todas as vantagens inerentes a essa linguagem, como por exemplo, independência de plataforma. Adicionalmente, o formato XML empregado pelo modelo TVX não apresenta conteúdo misto, caindo na categoria dos documentos XML orientados a dados (vide seção 1.1). Estes são documentos fracamente semi-estruturados ou, dito de outra forma, fortemente estruturados, sendo facilmente implementáveis tanto em bases XML nativas quanto em bases relacionais.

Além de um modelo para a organização lógica da história de um documento em formato XML, foram também definidas linguagens de consulta e de manipulação de dados que atuam sobre a estrutura gerada pelo modelo TVX. Ambas estendem as construções da linguagem XQuery e as expressões de caminho da linguagem XPath para possibilitar o acesso ao conteúdo do documento XML, com relação aos critérios de bitemporalidade e de versionamento. Através da linguagem de consulta, é possível consultar tanto a história presente quanto a história passada de todas as versões da base de dados. Através da linguagem de manipulação de dados, é possível escrever expressões que expandam a hierarquia de versões e que alterem o conteúdo de cada uma delas. Ambas as linguagens buscam tornar os detalhes de implementação do modelo TVX transparentes para o usuário final.

O estudo de caso de uma aplicação real também é ilustrado, demonstrando a aplicabilidade do modelo TVX em uma situação real e destacando as vantagens de sua utilização em relação ao método empregado atualmente. A desvantagem observada é a grande sobrecarga de espaço imposta pela conversão do formato original dos documentos XML para o formato utilizado no modelo TVX, a qual, contudo, tende a desaparecer e até mesmo virar a favor do modelo TVX conforme a base de dados sofre modificações.

Conforme discutido na seção 3.7, apesar de todas as vantagens propiciadas pelo uso conjunto de bitemporalidade e versionamento, o modelo TVX apresenta limitações – a saber, a falta de suporte ao gerenciamento da evolução dos esquemas dos documentos XML e a granularidade alta do versionamento, o qual atua somente sobre o documento como um todo, e não sobre seus fragmentos. Com a eliminação dessas limitações, o modelo TVX seria útil a uma classe mais ampla de aplicações, portanto há espaço para o surgimento de extensões do trabalho realizado.

Dentro do grupo de pesquisa no qual foi desenvolvido, o modelo TVX consiste em um primeiro trabalho na busca de se adaptar a pesquisa sobre modelos de dados tradicionais já existentes para o contexto de documentos XML. Assim sendo, este trabalho não esgota o assunto, podendo ser estendido em trabalhos futuros de diversas maneiras:

- **implementação de um protótipo –** a existência de um componente de *software* capaz de suportar o modelo TVX possibilitaria realizar comparações não só em relação à demanda de espaço, como foi feito no Estudo de Caso, mas também em relação à eficiência na execução de consultas e operações de atualização da base de dados. Há uma série de aspectos relativos à implementação física do modelo que não foram abordados neste trabalho, como organização física dos dados, implementação de índices, otimização de consultas, etc. Com um protótipo em mãos, torna-se possível experimentar diversas alternativas para estes tópicos e comprovar, na prática, qual apresenta melhores resultados;

- **adaptação para um modelo multitemporal –** o modelo TVX, conforme definido neste trabalho, pode ser considerado um modelo de dados bitemporal – isto é, a cada objeto são associados tanto tempo de transação quanto tempo de validade. A modificação do mesmo para um modelo multitemporal consiste na possibilidade de se especificar quais objetos devem ser temporalizados e de que forma; isto é, quais devem ser controlados apenas com tempo de transação, quais devem ser controlados apenas com tempo de validade, quais devem ser bitemporais e quais devem ser estáticos – estes últimos não recebem qualquer tipo de controle temporal, sendo que um novo estado sempre sobrescreve completamente o estado anterior, que é irremediavelmente perdido no processo. A possibilidade de definição de dados estáticos é importante porque evita a sobrecarga de informações temporais em situações em que não se deseja controlar a evolução de certos objetos; tomando como exemplo o Estudo de Caso, a rigor não é necessário incluir informações temporais para a seção de assinaturas da Constituição, pois esta parte do documento é imutável. Para mesclar dados estáticos e dinâmicos, é necessário alterar o modelo de diversas formas, aumentando a complexidade do mesmo; para a declaração do esquema dos documentos, por exemplo, não seria mais possível utilizar XML Schema puro, mas sim definir alguma extensão onde fosse possível especificar, para cada objeto XML, como este deve ser controlado em relação a seu histórico;

- **expansão do histórico para incluir as operações de atualização –** o modelo TVX é centrado em dados e não em operações – isto é, registra a evolução dos documentos XML armazenando os diferentes estados pelos quais a informação passa, e não através das operações que provocam as modificações. Esta abordagem é a mais adequada quando não se deseja manter um *log* das transações executadas; por exemplo, em um sistema onde milhares de transações são executadas diariamente, a sobrecarga de se arquivar cada uma das operações de cada transação concretizada possivelmente ultrapassaria o custo de armazenamento da própria base de dados. Ademais, nos modelos estudados que são centrados em operações, é necessário realizar uma série de cálculos para recuperar um estado passado do documento, processando novamente transações executadas no passado. Em certos cenários, contudo, pode ser desejável manter também um histórico das operações de modificação executadas. Tomando novamente o exemplo do Estudo de Caso, registrar as operações executadas significaria guardar um histórico do texto das Emendas Constitucionais promulgadas; certamente, alguém que fosse consultar o texto da Constituição gostaria de ter acesso também ao histórico das Emendas Constitucionais. Neste exemplo, guardar o histórico das Emendas não provoca uma grande sobrecarga

na base de dados, pois são modificações que ocorrem ocasionalmente – nos dezesseis anos desde a promulgação da Constituição até a escrita deste trabalho, foram criadas apenas cerca de cinqüenta emendas. Em casos como este, seria benéfico que houvesse uma extensão do modelo atual capaz de registrar não somente o histórico dos dados como também o histórico das operações executadas, quando for conveniente;

- **redução da granularidade do versionamento –** neste trabalho, a granularidade do versionamento é o documento como um todo. Um conceito de versionamento mais flexível e poderoso é visto em (GOLENDZINER, 1995), onde cada um dos objetos elementares pode ser versionado. Em outras palavras, no contexto deste trabalho isto significaria possibilitar que cada elemento, atributo e fragmento de texto em um documento XML apresentasse diferentes versões, e não apenas o elemento raiz, como é feito atualmente. Com isto surgiria o conceito de *configurações* – objetos versionados compostos de outros objetos versionados. Em seu estado atual, o modelo TVX dispensa uma grande atenção ao aspecto bitemporal da evolução do conteúdo dos documentos XML; uma seqüência natural deste trabalho seria desenvolver melhor a parte do versionamento, permitindo que cada sub-árvore em um documento XML pudesse apresentar versões distintas;

- **inclusão de mecanismos para evolução de esquemas –** o modelo TVX se preocupa com o controle das modificações executadas sobre o conteúdo de um documento XML que pode ou não ter um esquema definido. Quando há um esquema definido, este é imutável e rege todos os estados assumidos pelo documento em qualquer instante de tempo. Uma poderosa extensão ao modelo consistiria em permitir que o próprio esquema do documento estivesse sujeito a modificações ao longo do tempo; dessa forma haveria diversas versões do esquema, válidas durante períodos distintos. Não só o modelo de dados, como também as linguagens de consulta e de manipulação de dados deveriam ser estendidas para refletir esta extensão ao modelo atual. Em (SU et al, 2001) é apresentado um conjunto de operações destinadas a executar modificações sobre a DTD de um documento XML, porém sem registrar seu histórico, apenas armazenando o estado mais recente. É interessante destacar que, como no modelo TVX a descrição do esquema é feita utilizando um dialeto XML, é possível utilizar a própria codificação definida neste trabalho para registrar a evolução do esquema de um documento.

REFERÊNCIAS

ALLEN, J. F. Maintaining Knowledge about Temporal Intervals. **Communications of the ACM**, New York, v. 26, n. 11, p. 832-843, Nov. 1983.

AMAGASA, T.; YOSHIKAWA, M.; UEMURA, S. A Data Model for Temporal XML Documents. In: INTERNATIONAL CONFERENCE ON DATABASE AND EXPERT SYSTEMS APPLICATIONS, DEXA, 11., 2000, London. **Database and Expert Systems Applications**: proceedings. Berlin: Springer-Verlag, 2000.

AMAGASA, T.; YOSHIKAWA, M.; UEMURA, S. **Realizing Temporal XML Repositories using Temporal Relational Databases.** In: INTERNATIONAL SYMPOSIUM ON COOPERATIVE DATABASE SYSTEMS FOR ADVANCED APPLICATIONS, 2001. **Proceedings…** [S.l.:s.n.], 2001.

ANTUNES, D. C. **Modelagem temporal de sistemas**: uma abordagem fundamentada em redes de Petri. 1997. Dissertação (Mestrado em Ciência da Computação) – Instituto de Informática, UFRGS, Porto Alegre.

BERTINO, E.; FERRARI, E.; GUERRINI, G. A Formal Temporal Object-Oriented Data Model. In: INTERNATIONAL CONFERENCE ON EXTENDING DATABASE TECHNOLOGY, EDBT,5., 1996, Avignon. **Advances Database Technology**: proceedings. Berlin: Springer-Verlag, 1996. (Lecture Notes in Computer Science, 1057).

BOURRET, R. **XML and Databases.** 2004. Disponível em: <http://www.rpbourret.com/xml/XMLAndDatabases.htm>. Acesso em: jan. . 2005.

CAPROTTI, O.; CARLISE, D. P.; COHEN, A. M. **The OPENMath Standard**. 1999. Disponível em: < http://www.nag.co.uk/projects/OPENMath/omstd >. Acesso em: abr. 2004.

CHAWATHE, S.; ABITEBOUL, S.; WIDOW, J. Representing and Querying Changes in Semistructured Data. In: INTERNATIONAL CONFERENCE ON DATA ENGINEERING, 1998. **Proceedings…** [S.l.:s.n.], 1998.

CHIEN, S.-Y. et al. Storing and Querying Multiversion XML Documents using Durable Node Numbers. In: INTERNATIONAL CONFERENCE ON WEB INFORMATION SYSTEMS ENGINEERING, 2001, Kyoto, Japan. **Proceedings…** [S.l.:s.n.], 2001.

CHIEN, S.-Y. et al. Efficient Complex Query Support for Multiversion XML Documents. In: INTERNATIONAL CONFERENCE ON EXTENDING DATABASE TECHNOLOGY, EDBT, 2002. **Proceedings…** [S.l.:s.n.], 2002.

CHIEN, S.-Y.; TSOTRAS, V. J.; ZANIOLO, C. Efficient Management of Multiversion Documents by Object Referencing. In: INTERNATIONAL CONFERENCE ON VERY LARGE DATA BASES, 2001. **Proceedings…** [S.l.:s.n.], 2001.

CHIEN, S.-Y.; TSOTRAS, V. J.; ZANIOLO, C. **XML Document Versioning.** Santa Barbara, EUA: SIGMOD, 2001.

CHIEN, S.-Y.; TSOTRAS, V. J.; ZANIOLO, C. Efficient schemes for managing multiversion XML documents. **VLDB Journal**, Heidelberg, v. 11, n.4, p.332-353, 2002.

CLIFFORD, J.; CROKER; A. The Historical Relational Data Model (HRDM) and Algebra Based on Lifespans. In: INTERNATIONAL CONFERENCE ON DATA ENGINEERING, 1987. **Proceedings…** [S.l.:s.n.], 1987.

COBÉNA, G.; ABITEBOUL, S.; MARIAN, A. Detecting Changes in XML Documents. In: INTERNATIONAL CONFERENCE ON DATA ENGINEERING, 2002. **Proceedings…** [S.l.:s.n.], 2002.

COHEN, E.; KAPLAN, H.; MILO, T. Labeling Dynamic XML Trees**.** In: ACM PRINCIPLES OF DATABASE SYSTEMS, 2002. **Proceedings…** [S.l.:s.n.], 2002.

DYRESON, C. E. Observing Transaction-time Semantics with TTXPath. In: INTERNATIONAL CONFERENCE ON WEB INFORMATION SYSTEMS ENGINEERING, 2001. **Proceedings…** [S.l.:s.n.], 2001.

EDELWEISS, N.; OLIVEIRA, J. P. M. de. **Modelagem de aspectos temporais de sistemas de informação**. Recife, Brasil: UFPE, 1994. Trabalho apresentado na 9. Escola de Computação.

EDELWEISS, N.; OLIVEIRA, J. P. M. de; PERNICI, B. An Object-Oriented Temporal Model. In: INTERNATIONAL CONFERENCE ON ADVANCED INFORMATION SYSTEMS ENGINEERING, 5., 1993, Paris. **Advanced Information Systems Engineering**. Berlin: Springer-Verlag, 1993.

ELMASRI, R.; NAVATHE, S. B. **Fundamentals of Database Systems.** 3rd ed. Reading: Addison-Wesley, 2000.

ELMASRI, R.; WUU, G. T. J.; KOURAMAJIAN, V. A temporal model and query language for EER Databases. In: TANSEL, A. et al. **Temporal databases**: theory, design and implementation. Redwood City: The Benjamin/Cummings, 1993.

FLORESCU, D.; KOSSMANN, D. Storing and Querying XML Data Using an RDMBS**. IEEE Data Engineering Bulletin**, [S.l.], v. 22, n. 3, p.27-34, 1999.

GRANDI, F. et al. A Temporal Data Model and Management System for Normative Texts in XML Format. In: INTERNATIONAL WORKSHOP ON WEB INFORMATION AND DATA MANAGEMENT, 2003. **Proceedings…** [S.l.:s.n.], 2003.

GOLENDZINER, L. G. **Um modelo de versões para bancos de dados orientados a objetos.** 1995. Tese (Doutorado em Ciência da Computação)- Instituto de Informática, UFRGS, Porto Alegre.

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

JENSEN, C. S. **Temporal Database Management**. 1999. Tese (Doutorado em Ciência da Computação)- Department of Computer Science, Aalborg University, Aalborg.

KANE, B.; SU, H.; RUNDENSTEINER, E. A. Consistently Updating XML Documents Using Incremental Constraint Check Queries**.** In: WORKSHOP ON WEB INFORMATION AND DATA MANAGEMENT, 2002. **Proceedings…** [S.l.:s.n.], 2002.

LIMA, J. A. de O. **Constituição Federal em XML.** [mensagem pessoal]. Mensagem recebida por rgsantos@inf.ufrgs.br em 14 de janeiro de 2005.

MANUKYAN, M. G; KALINICHENKO, L. A. **Temporal XML.** Vilnius, Lituânia: Advances in Databases and Information Systems, 2001.

MARIAN, A. et al. Change-Centric Management of Version in a XML Warehouse. In: INTERNATIONAL CONFERENCE ON VERY LARGE DATA BASES, 2001. **Proceedings…** [S.l.:s.n.], 2001.

MELLO, R. dos S. **Gerenciamento de Dados XML.** Mini-Curso do 5. Palmas, Encontro dos Estudantes de Informática do Estado do Tocantins, 2003.

MORO, M. M. **Modelo Temporal de Versões**. 2001. Dissertação (Mestrado em Ciência da Computação)- Instituto de Informática, UFRGS, Porto Alegre.

NAVATHE, S. B.; AHMED, R. A. A Temporal Relational Model and a Query Language. **Information Systems**, [S.l.], v.47, n.2, p.147–175, Mar. 1989.

OBJECT MANAGEMENT GROUP (OMG). **Unified Modeling Language (UML) 2.0.** 2004. Disponível em: < http://www.uml.org/#UML2.0 >. Acesso em: fev. 2005.

ÖZSU, M. T. et al. TIGUKAT Object Management System: Initial Design and Current Directions. In: IBM/CAS CONFERENCE, 1993. **Proceedings…** [S.l.:s.n.], 2003.

SARDA, N. L. Extensions to SQL for Historical Databases. **IEEE Transactions on Knowledge and Data Engineering**, Los Alamitos, v.2, n.2, p. 220-230, June 1990.

SNODGRASS, R. T. **Developing Time-Oriented Database Applications in SQL.** San Francisco: Morgan Kaufmann, 2000.

SU, H. et al. XEM: Managing the evolution of XML Documents. In: INTERNATIONAL WORKSHOP ON RESEARCH ISSUES IN DATA ENGINEERING, 2001. **Proceedings…** [S.l.:s.n.], 2001.

SU, H. et al. A Lightweight XML Constraint Check and Update Framework. In: INTERNATIONAL WORKSHOP ON EVOLUTION AND CHANGE IN DATA MANAGEMENT, 2., 2002, Tampere. **Proceedings…** [S.l.:s.n.], 2002.

TANSEL, A. et al. **Temporal Databases**: theory, design, and implementation. RedwoodCity: Benjamin/Cummings, 1993.

TATARINOV, I. et al. **Updating XML.** Santa Barbara, EUA: SIGMOD, 2001.

TAUZOVICH, B. Towards temporal extensions to the entity-relationship model.In: INTERNATIONAL CONFERENCE ON CONCEPTUAL MODELING, 1991. **Proceedings…** [S.l.:s.n.], 1991.

THEODOULIDIS, C. I.; LOUCOPOULOS, P.; WANGLER, B. The Entity-Relationship Time Model and the Conceptual Rule Language. In: INTERNATIONAL CONFERENCE ON CONCEPTUAL MODELING, 1991. **Proceedings**… [S.l.:s.n.], 1991.

WANG, F.; ZANIOLO, C. Preserving and Querying Histories of XML-Published Relational Databases. In: INTERNATIONAL WORKSHOP ON EVOLUTION AND CHANGE IN DATA MANAGEMENT, 2002. **Proceedings…** [S.l.:s.n.], 2002.

WANG, F.; ZANIOLO, C. Representing and Querying the Evolution of Databases and their Schemas in XML. In: INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING AND KNOWLEDGE ENGINEERING, 2003. **Proceedings…**[S.l.:s.n.], 2003.

WANG, F.; ZANIOLO, C. Temporal Queries in XML Document Archives and Web Warehouses. In: INTERNATIONAL SYMPOSIUM ON TEMPORAL REPRESENTATION AND REASONING AND INTERNATIONAL CONFERENCE ON TEMPORAL LOGIC, 2003. **Proceedings…** [S.l.:s.n.], 2003.

WANG, F.; ZANIOLO, C. Publishing and Querying the Histories of Archived Relational Databases in XML. In: INTERNATIONAL CONFERENCE ON WEB INFORMATION SYSTEMS ENGINEERING, 2003. **Proceedings…** [S.l.:s.n.], 2003.

WANG, F.; ZANIOLO, C. XBiT: An XML-based Bitemporal Data Model**.** In: INTERNATIONAL CONFERENCE ON CONCEPTUAL MODELING, 2004, Xangai, China. **Conceptual Modeling**: proceedings. [S.l.:s.n.], 2004.

WANG, Y.; DeWITT, D. J.; CAI, J.-Y. X-Diff: An Effective Change Detection Algorithm for XML Documents. In: INTERNATIONAL CONFERENCE ON DATA ENGINEERING, 2003. **Proceedings…** [S.l.:s.n.], 2003.

WONG, R. K.; LAM, N. Managing and Querying Multi-Version XML Data with Update Logging. In: ACM SYMPOSIUM ON DOCUMENT ENGINEERING, 2002. **Proceedings…** [S.l.:s.n.], 2002.

WONG, R. K.; LAM, N. Efficient Re-construction of Document Versions Based on Adaptative Forward and Backward Change Deltas. In: INTERNATIONAL CONFERENCE ON DATABASE AND EXPERT SYSTEMS APPLICATIONS, DEXA, 14., 2003. **Database and Expert Systems Applications**: proceedings. Berlin: Springer-Verlag, 2003. (Lecture Notes in Computer Science, v.2736).

WORLD WIDE WEB CONSORTIUM (W3C). **Extensible Markup Language (XML).** 1998. Disponível em: < http://www.w3.org/XML/ >. Acesso em: abr. 2004.

WORLD WIDE WEB CONSORTIUM (W3C). **XML Path Language (XPath).** 1999. Disponível em: < http://www.w3.org/TR/xpath/ >. Acesso em: abr. 2004.

WORLD WIDE WEB CONSORTIUM (W3C). **XSL Transformations (XSLT).** 1999. Disponível em: < http://www.w3.org/TR/xslt/ >. Acesso em: abr. 2004.

WORLD WIDE WEB CONSORTIUM (W3C). **W3C XML Schema.** 2000. Disponível em: <http://www.w3.org/XML/Schema/ >. Acesso em: abr. 2004.

WORLD WIDE WEB CONSORTIUM (W3C). **XQuery 1.0:** An XML Query Language. 2001. Disponível em: < http://www.w3.org/TR/xquery/ >. Acesso em: abr. 2004.

ZHANG, S.; DYRESON, C. E. Adding Valid Time to XPath. In: DATABASE AND NETWORK INFORMATION SYSTEMS, 2002. **Proceedings…** [S.l.: s.n.], 2002.

ANEXO DESCRIÇÃO EM XML SCHEMA DOS DOCUMENTOS XML DO ESTUDO DE CASO

```
<schema xmlns="http://www.w3.org/2001/XMLSchema">
 <element name="norma">
  <complexType>
   <sequence>
    <element name="parte-preliminar">
     <complexType>
      <sequence>
       <element name="epigrafe" type="string"/>
       <element name="ementa" type="string"/>
       <element name="preambulo" type="string"/>
      </sequence>
     </complexType>
    </element>
    <element name="parte-normativa">
     <complexType>
      <element name="titulos">
       <complexType>
        <element name="titulo" maxOccurs="unbounded">
         <complexType>
          <sequence>
           <element name="cabecalho" type="cabecalhoType"/>
           <choice>
            <element name="artigos" type="artigosType"/>
            <element name="capitulos">
             <complexType>
              <element name="capitulo" maxOccurs="unbounded">
               <complexType>
                <sequence>
                 <element name="cabecalho" type="cabecalhoType"/>
                 <choice>
                  <element name="artigos" type="artigosType"/>
                  <element name="secoes">
                   <complexType>
                    <element name="secao" maxOccurs="unbounded">
                     <complexType>
                      <sequence>
                       <element name="cabecalho" type="cabecalhoType"/>
                       <choice>
                        <element name="artigos" type="artigosType"/>
                        <element name="subsecoes">
                         <complexType>
                          <element name="subsecao" maxOccurs="unbounded">
                           <complexType>
                            <sequence>
                             <element name="cabecalho" type="cabecalhoType"/>
                             <element name="artigos" type="artigosType"/>
                            </sequence>
                            <attribute name="ID" type="ID" use="required"/>
                           </complexType>
                          </element>
                         </complexType>
                        </element>
                       </choice>
                      </sequence>
                      <attribute name="ID" type="ID" use="required"/>
                     </complexType>
```

```
                </element>
               </complexType>
              </element>
             </choice>
            </sequence>
            <attribute name="ID" type="ID" use="required"/>
           </complexType>
          </element>
         </complexType>
        </element>
       </choice>
      </sequence>
      <attribute name="ID" type="ID" use="required"/>
     </complexType>
    </element>
   </complexType>
  </element>
  <element name="parte-final">
   <complexType>
    <sequence>
     <element name="localdata" type="string"/>
     <element name="assinatura">
      <complexType>
       <element name="entidades">
        <complexType>
         <element name="entidade" maxOccurs="unbounded">
          <complexType>
           <sequence>
            <element name="nome" type="string" minOccurs="0"/>
            <element name="pessoas">
             <complexType>
              <element name="pessoa" maxOccurs="unbounded">
               <complexType>
                <sequence>
                 <element name="nome" type="string"/>
                 <element name="cargo" type="string" minOccurs="0"/>
                </sequence>
               </complexType>
              </element>
             </complexType>
            </element>
           </sequence>
          </complexType>
         </element>
        </complexType>
       </element>
      </sequence>
     </complexType>
    </element>
   </sequence>
   <attribute name="tipo" type="string" use="required"/>
   <attribute name="numero" type="integer" use="required"/>
   <attribute name="complemento" type="integer" use="required"/>
   <attribute name="sequencial" type="integer" use="required"/>
   <attribute name="data-assinatura" type="string" use="required"/>
   <attribute name="tipo-versao" type="string" use="required"/>
   <attribute name="seq-versao" type="integer" use="required"/>
   <attribute name="data-consolidacao" type="string" use="required"/>
   <attribute name="seq-consolidacao" type="integer" use="required"/>
  </complexType>
 </element>
 <complexType name="cabecalhoType">
  <element name="cabecalhotexto" type="string"/>
  <attribute name="ID" type="ID" use="required"/>
 </complexType>
 <complexType name="artigosType">
  <element name="artigo" maxOccurs="unbounded">
   <complexType>
    <sequence>
     <element name="artigotexto" type="string">
      <complexType>
       <attribute name="ID" type="ID" use="required"/>
      </complexType>
```

```
     </element>
     <element name="incisos" type="incisosType" minOccurs="0"/>
     <element name="paragrafos" minOccurs="0">
      <complexType>
       <element name="paragrafo" maxOccurs="unbounded">
        <complexType>
         <sequence>
          <element name="paragrafotexto" type="string"/>
          <element name="incisos" type="incisosType" minOccurs="0"/>
         </sequence>
         <attribute name="ID" type="ID" use="required"/>
         <attribute name="numero" type="string" use="required"/>
        </complexType>
       </element>
      </complexType>
     </element>
    </sequence>
    <attribute name="ID" type="ID" use="required"/>
    <attribute name="numero" type="integer" use="required"/>
   </complexType>
  </element>
 </complexType>
 <complexType name="incisosType">
  <element name="inciso" maxOccurs="unbounded">
   <complexType>
    <sequence>
     <element name="incisotexto" type="string"/>
     <element name="alineas" minOccurs="0">
      <complexType>
       <element name="alinea" maxOccurs="unbounded">
        <complexType>
         <element name="alineatexto" type="string"/>
         <attribute name="ID" type="ID" use="required"/>
         <attribute name="numero" type="string" use="required"/>
        </complexType>
       </element>
      </complexType>
     </element>
    </sequence>
    <attribute name="ID" type="ID" use="required"/>
    <attribute name="numero" type="string" use="required"/>
   </complexType>
  </element>
 </complexType>
</schema>
```