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

Extensão de uma Linguagem de Consulta para Documentos XML com Características de Tempo e de Versão

por

CLÁUDIO HESSEL PEIXOTO GOMES

Dissertação submetida à avaliação,
como requisito parcial para a obtenção do grau de
Mestre em Ciência da Computação

Prof. Dr. Clesio Saraiva dos Santos
Orientador
Profa. Dra. Nina Edelweiss
Co-orientadora

Porto Alegre, abril de 2002

CIP – Catalogação na Publicação

Gomes, Cláudio Hessel Peixoto
Extensão de uma Linguagem de Consulta para Documentos XML com Características de Tempo e de Versão / por Cláudio Hessel Peixoto Gomes. – Porto Alegre: PPGC da UFRGS. 2002.
147 f. : il.

Dissertação (mestrado) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação, Porto Alegre, BR – RS, 2002. Orientador: Santos, Clesio Saraiva dos; Co-Orientadora: Edelweiss, Nina.

1. Linguagens de Consulta XML. 2. Modelo Temporal de Versões. I. Santos, Clesio Saraiva dos. II. Edelweiss, Nina. III. Título.

Agradecimentos

Inicialmente, gostaria de agradecer à Capes, pelo suporte financeiro prestado nesses dois anos de curso, que propiciaram realizá-lo em caráter exclusivo.

Partindo para os agradecimentos pessoais, não poderia começar, senão, pelas duas pessoas mais importantes que me guiaram pelos caminhos da pós nesse breve período que passei pela UFRGS. Em primeiro lugar, o professor Clesio. Ele teve a coragem de dar uma oportunidade para um gremista do interior realizar um curso de tão elevada distinção. Conseguiu me suportar sem dar uma reprimenda sequer! Professor, receba meus mais sinceros cumprimentos por me ajudar nessa etapa importante de minha vida, muitíssimo obrigado! Na segunda colocação, desempatando com o professor apenas pelo índice técnico (não me levem a mal, mas sou aficionado por futebol!), está a professora Nina. Tive a honra de ter sua companhia e de receber sua ajuda nesse último ano do mestrado. Ela foi fundamental na parte final do trabalho, além da sempre prestativa presença em todas as reuniões do grupo. Professora, obrigado por tudo!

Falando em grupo, não há como esquecer das minhas colegas de grupo. Sim! Tive a sorte de estar cercado por mulheres para realizar meu trabalho excetuando, obviamente, o "capitão" Clesio. Aglê, Anelise, Adriana, Lialda, Mirella, Paôla, Renata e Sílvia, agradeço a todas vocês pela equipe eficiente que formamos. Dentre todas, três merecem alguns agradecimentos a mais. Adriana e Renata, não esqueci a ajuda que vocês prestaram logo que ingressei no curso, exatamente no momento difícil no qual o professor ainda não possuía condições de reassumir suas atividades. Obrigado, minhas pré-orientadoras. Lialda, agradeço toda a ajuda que me deste. Como seguimos por uma área com enfoque um pouco diferenciado da linha original de nosso orientador, tivemos de trabalhar juntos para realizar nossas pesquisas de uma maneira mais produtiva. Colega, obrigado. Agradeço, também, a todo o pessoal do grupo do professor Heuser e do grupo da professora Mara. Neste último, em especial, à Fabiana.

Como anteriormente destaquei as colegas, preciso ressaltar os colegas/amigos que seguiram o mesmo caminho que eu. André, Goiano, Kenzo e Pablo, já nos conhecíamos antes, mas tivemos, também, de nos "suportar" dentro do curso. Mesmo assim, valeu! O pessoal do FutInf merece um destaque especial: Alécio, Carlos, César, Eidy, Everaldo, Filipe, João, Luís Otávio, Nilton, Rafael, Renato e Vinícius, agradeço as conversas "sempre relacionadas às nossas pesquisas" que tivemos. Destaco os áureos tempos de RU e futebol que não poderão ser esquecidos. Pessoal, valeu mesmo a parceria que formamos.

Professores e funcionários do Instituto de Informática, vocês também merecem minha total consideração. Obrigado a todos.

Para finalizar, preciso agradecer ao "grupo" mais importante de que faço parte: minha família. Não posso destacar alguém em especial, cada um de vocês sabe que é muito importante para mim. Precisei me afastar um pouco de vocês para realizar o mestrado. Todavia, toda a saudade foi transformada em dedicação para vencer essa etapa. Acredito que consegui! Obrigado a todos vocês!

Sumário

Lista de Abreviaturas

BD	Banco de Dados
BDOO	Banco de Dados Orientados a Objetos
DAD	Document Access Definition
DBMS	Database Management System
DTD	Document Type Definition
JDBC	Java Database Connector
Lore	Lightweight Object Repository
Lorel	Lore Language
ODMG	Object Data Management Group
OEM	Object Exchange Model
OID	Object Identificator
OQL	Object Query Language
SGBD	Sistema de Gerenciamento de Banco de Dados
SQL	Structured Query Language
TVM	Temporal Version Model
TVQL	Temporal Version Query Language
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
W3C	World Wide Web Consortium
XML	Extensible Markup Language
XML-QL	Query Language for XML
XPath	XML Path Language
XQL	XML Query Language

Lista de Figuras

Lista de Tabelas

Resumo

O uso da XML (*Extensible Markup Language*) em aplicações envolvendo bancos de dados vem se consolidando nos últimos dois anos. Os principais sistemas de gerenciamento de banco de dados já incorporam essa tecnologia em suas mais recentes versões. Dentre diversas aplicações destaca-se a publicação de dados relacionais em visões XML. Diferentemente da XML, o Modelo Temporal de Versões (TVM) não apresenta suporte entre os bancos de dados atuais. Esse modelo, que une características temporais com o conceito de versão para projetar aplicações orientadas a objetos, precisa ser mapeado para ser adequadamente controlado em um SGBD (Sistema de Gerenciamento de Banco de Dados). Cumprida essa etapa, aplicações do TVM também podem gerar visões XML. Nesse trabalho é inicialmente apresentada uma forma de representar instâncias de aplicações do TVM em um formato XML. Os documentos definidos a partir desse formato de representação são utilizados como base para consultas. Em seguida, é proposta uma extensão de uma linguagem de consulta XML visando proporcionar recursos para a recuperação de informações temporais e de versão representadas em documentos XML. São definidas funções temporais e versionadas que são incorporadas à linguagem base. O funcionamento das funções e a especificação de consultas temporais versionadas são descritos em detalhes no decorrer do trabalho. Uma ferramenta que implementa a linguagem base é utilizada na realização de testes visando validar as novas funções.

Palavras-chave: XML, TVM, TVQL, linguagens de consultas XML, XQuery, funções extensíveis.

...

TITLE: "EXTENDED XML DOCUMENTS QUERY LANGUAGE WITH TIME AND VERSION FEATURES"

Abstract

The use of the XML in applications involving databases has grown in the last two years. Recent versions of the main database management systems already incorporate this technology. Publishing relational data in XML can be identified as one of the different applications of XML. The Temporal Version Model (TVM) has no support in current databases. This model matches temporal features with the version concept to project object-oriented applications and needs to be mapped to be managed in a DBMS (Database Management System). Once this mapping is achieved, TVM applications can also generate XML views. This work presents initially a way to represent TVM instances in a XML format. Thus, documents created following this representation may be used in queries. Afterwards, an extension of a XML query language is proposed, aiming to provide features for the retrieval of temporal and version information represented in XML documents. Temporal and version functions are defined and incorporated in a base language. The way functions are executed and the specification of temporal version queries are described in details. A tool that implements the base language is used to test the new functions, with the aim of validation.

Keywords: XML, TVM, TVQL, XML query languages, XQuery, Extensible functions.

1 Introdução

1.1 Motivação

Existem muitas aplicações que consideram aspectos temporais, o conceito de versão, ou ainda, a integração destas duas características. Este último grupo é menos numeroso, pois não existem muitos modelos que permitem modelagens com esse grau de detalhamento. O Modelo Temporal de Versões (TVM) [MOR 2001] integra características temporais ao nível de objetos, versões, relacionamentos e propriedades. Uma de suas vantagens é a facilidade de associação com especificações existentes, visto que nem toda classe precisa ser modelada como temporal versionada.

Atualmente, não existem bancos de dados que suportem o modelo TVM. Assim, para que os dados de aplicações orientadas a objetos modeladas através do TVM sejam gerenciados em bancos de dados, mostra-se necessária a realização de mapeamentos para as estruturas de dados de bancos de dados orientados a objetos ou objeto-relacionais (IBM DB2, por exemplo). Para tornar mais perceptível o processo de recuperação de informações desses dados, deve existir uma linguagem de consulta condizente com o modelo. Nesse ponto se enquadra a TVQL (*Temporal Version Query Language*) [MOR 2001a], baseada em SQL (*Structured Query Language*) e totalmente orientada aos conceitos do TVM.

Com o uso da linguagem de consulta é possível estabelecer visões da base TVM, de acordo com a necessidade da aplicação e do usuário. Visões são utilizadas, por exemplo, no intercâmbio de dados entre aplicações distintas. Com a rápida disseminação da linguagem XML e sua crescente integração com bancos de dados, um de seus inúmeros usos possíveis é o de materializar visões relacionais em XML. Nesse caso, as visões teriam origem relacional, mas representariam dados temporais versionados e orientados a objetos em documentos XML.

Esses documentos, criados a partir de dados armazenados em SGBDs, podem ser consultados no intervalo existente entre as aplicações que os intercambiam. Para isso, existem várias especificações de linguagens para dados XML. No entanto, não se tem conhecimento de alguma que apresente um suporte preferencialmente voltado ao controle de tempo e ao controle de versão. Exatamente nesse ponto está concentrada a pesquisa abordada no presente trabalho, ou seja, consultar documentos XML criados a partir de dados de aplicações modeladas pelo TVM.

1.2 Objetivos

O objetivo do trabalho é definir uma extensão, com características de tempo e de versão, para uma linguagem de consultas a documentos XML. O modo escolhido para estender a linguagem base foi a definição de novas funções. Dessa forma, não é modificada a especificação original da linguagem de consultas.

O primeiro passo da pesquisa configura-se na definição da linguagem que receberá a extensão. Tem-se conhecimento de que as pesquisas relacionadas a linguagens de consultas a dados XML encontram-se num estádio ainda indefinido, com várias propostas, incluindo muitas especificações. Por esses motivos, a etapa de escolha

da linguagem é de caráter primordial para o restante da pesquisa. É fundamental que a linguagem se torne padrão na área de consultas a documentos XML e que possua características as quais permitam uma extensão sem grandes impactos.

No entanto, para justificar a extensão temporal e de versão da linguagem, mostra-se imprescindível que os documentos representem informações com o controle de tempo e de versão. Para atender a essa necessidade, um modelo de representação de informações em XML, com essas características, também é proposto. Essa representação segue os preceitos do Modelo Temporal de Versões; logo, as informações representadas nos documentos XML com características de tempo e de versão são instâncias de aplicações modeladas a partir do TVM. Esses documentos podem ser utilizados para apresentar visões da base TVM em XML, além de serem úteis no intercâmbio de dados entre aplicações de banco de dados que utilizam o TVM como modelo.

Para validar a extensão, é utilizada uma ferramenta que implementa a especificação da linguagem de consultas XML definida na fase inicial do trabalho. Considerando um documento com características de tempo e de versão, são estabelecidas consultas que utilizam os recursos da extensão. Assim, constata-se, de forma prática, como as novas funções operam e como elas fornecem seus resultados.

1.3 Organização do Texto

Além desta seção introdutória, o texto é composto de mais sete capítulos. O segundo expõe as funcionalidades de algumas linguagens de consultas para dados XML juntamente com um estudo comparativo que destaca a linguagem escolhida como base de extensão. O capítulo três apresenta, de forma bastante resumida, o Modelo Temporal de Versões.

O quarto capítulo mostra a linguagem de consulta definida para o Modelo Temporal de Versões. Seguindo as principais características dessa linguagem, é definida a extensão da linguagem de consulta XML. O capítulo cinco descreve uma proposta de representação de instâncias de aplicações modeladas a partir do TVM em XML. Dessa representação são criados documentos com características de tempo e de versão.

O sexto capítulo apresenta a sintaxe de todas as funções definidas para estender a linguagem base. O capítulo posterior apresenta alguns exemplos de consultas que utilizam as funções propostas. Essas consultas são baseadas em uma pequena aplicação-exemplo e testadas com uso de uma ferramenta que implementa a linguagem base da extensão. Ainda nesse capítulo é brevemente apresentada uma proposta de ambiente de utilização da linguagem estendida. O último capítulo traz uma revisão do trabalho ressaltando algumas conclusões e propondo novos trabalhos.

Ao final do texto são apresentados os seguintes anexos:

- Anexo 1: um arquivo exemplo de XML com sua respectiva DTD (*Document Type Definition*);
- Anexo 2: DTD da representação de instâncias do TVM em XML;
- Anexo 3: XML Schema da representação de instâncias do TVM em XML;
- Anexo 4: instâncias de uma aplicação modelada através do TVM em XML.

2 Linguagens de Consulta XML

Dados disponíveis eletronicamente podem ser representados de diferentes formas, partindo dos dados sem estrutura (sistema de arquivos) até alcançar sistemas altamente estruturados, como os bancos de dados relacionais [ABI97a]. Essas informações podem ser atingidas através de aplicações de acesso e intercâmbio de dados e, inclusive, de linguagens de consulta.

Alguns dados possuem uma representação implícita, ou seja, a estrutura existe, porém deve ser extraída. Dados com essas características são denominados **dados semi-estruturados** [BUN 97, SUC98a, SUC98b]. Eles até podem possuir um esquema explícito e uniforme, porém não são fortemente tipados nem são representados através de tabelas, como no modelo relacional.

Para realizar consultas sobre informações com essas particularidades, uma linguagem deve ser mais flexível do que as encontradas nos SGBDs. No entanto, as técnicas já consagradas nos bancos de dados devem ser ao máximo integradas a esse novo conceito de linguagem, assim como técnicas específicas de recuperação de informação para documentos eletrônicos. Uma das várias propostas para trabalhar com consultas sobre dados semi-estruturados é a Lorel (*Lore Language*), que utiliza as técnicas de BDOO (Banco de Dados Orientados a Objetos) e documentos XML [BRA 2000, LIG 99].

Exclusivamente para documentos XML algumas pesquisas [BUN 98, DEU 99, OLK98, W3C2000] definiram operações que devem estar presentes em qualquer proposta de linguagem que deseje trabalhar com dados XML. A princípio, as linguagens devem operar sobre um único documento ou sobre coleções; devem selecionar um documento completo ou partes, de acordo com condições definidas em suas cláusulas. Além disso, as linguagens devem construir documentos baseados nos resultados obtidos a partir de uma consulta.

Paralelamente às proposições de novas linguagens, alguns trabalhos [ABI 98, ATZ 98] discutem a real necessidade de criação de novas linguagens de consulta para dados XML. Defendem a utilização de recursos existentes e consolidados, sobretudo OQL (*Object Query Language*), que faz parte do padrão ODMG (*Object Data Management Group*) [CAT 97]. Basicamente, é proposta a definição de repositórios de documentos XML como bases orientadas a objetos, o que permitiria que linguagens de consulta orientadas a objetos fossem utilizadas para consultar e transformar fontes XML.

Neste capítulo, não será aprofundada a adaptação de tecnologias, e, sim, apresentadas as novas propostas de linguagens. Foram previamente selecionadas quatro linguagens, que representam as principais tendências dentro de consultas para dados XML. São descritas suas características básicas sempre com utilização de exemplos. Após a explanação, é realizada uma análise comparativa entre as quatro propostas de acordo com preceitos fundamentais na área de linguagens de consulta XML. Vários pontos poderiam ser destacados para a realização desse comparativo, contudo foram escolhidos aqueles que estão relacionados com a iteração usuário-linguagem e com a capacidade de extensão, objetivo deste trabalho. Ao final do capítulo, além de destacar a linguagem que servirá como base para a extensão, são apresentadas considerações

gerais sobre as linguagens juntamente com um quadro que resume os pontos utilizados no estudo comparativo.

1.4 Lorel

Para atender às particularidades encontradas em dados semi-estruturados [BUN 97], desenvolveu-se um projeto na Universidade de Stanford chamado *Projeto Lore* (*Lightweight Object Repository*) [ABI 97b], cujo objetivo era prover um ambiente adequado para armazenar, consultar e atualizar dados semi-estruturados. Uma das metas do projeto era desenvolver um modelo que fosse uma extensão do modelo ODMG, juntamente com uma linguagem que fosse baseada na OQL. A linguagem estendida da OQL desenvolvida no projeto foi denominada Lorel (*Lore Language*).

A linguagem Lorel permite que o usuário, mesmo com um conhecimento parcial do modelo que está consultando, consiga expressar consultas significativas que possam recuperar os resultados esperados. O modelo de dados ao qual são aplicadas as consultas Lorel é o *Object Exchange Model* (OEM). A representação de um banco de dados baseado no modelo OEM pode ser considerada um grafo com valores complexos representados nos nodos internos e com valores atômicos representados nos nodos folha [ABI 97b].

Analisando o modelo OEM como uma extensão do modelo ODMG, justifica-se a escolha da linguagem OQL como base da linguagem Lorel. Além da integração com o modelo ODMG, a linguagem OQL apresenta similaridades com SQL, o que facilita seu entendimento e sua utilização. Uma das características da OQL encontrada na Lorel é o emprego de expressões de caminho, as quais permitem que consultas possam ser feitas sem que haja um preciso conhecimento da estrutura a ser consultada. Expressões de caminho são fundamentadas em rótulos e em *wildcards*, conceitos que serão apresentados mais adiante nesta seção.

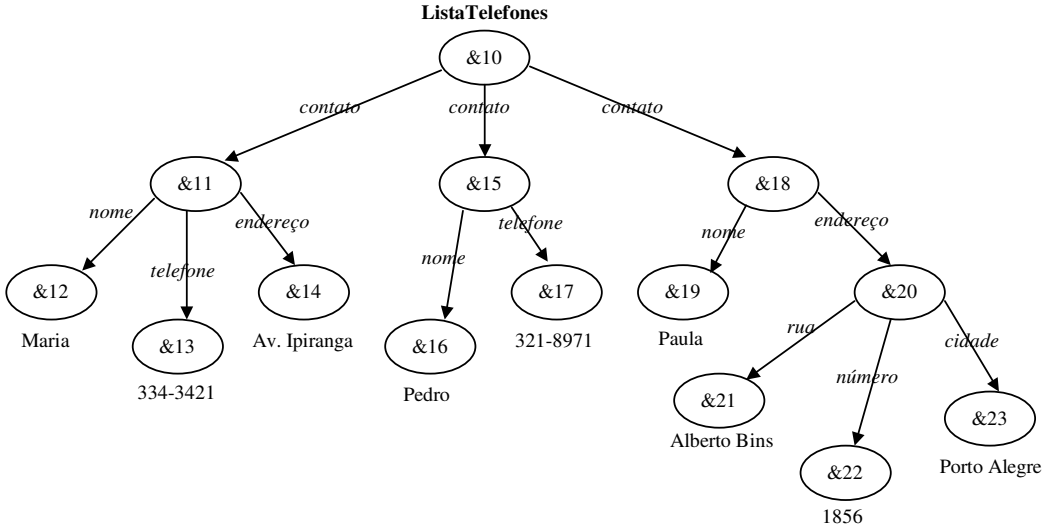

FIGURA 2.1 – Grafo OEM

1.4.1 Expressões de caminho simples

O emprego de expressões de caminho pode ser considerado como uma forma navegacional de consulta. Observando o grafo representado na **FIGURA 2.1**, nota-se que as linhas que unem os nodos são rotuladas. Esses rótulos permitem que o usuário atinja um dado conjunto de objetos apenas seguindo uma seqüência no grafo OEM.

Uma expressão de caminho simples pode ser definida como: $\mathbf{O.r_1….r_2}$, onde $\mathbf{r_1….r_2}$ são os rótulos definidos no grafo OEM e **O** indica o nome do objeto ou uma variável que o represente. Baseando-se no objeto `ListaTelefones`, um exemplo de expressão de caminho pode ser: `ListaTelefones.contato.nome`. Esse caminho indica que, partindo de um objeto `ListaTelefones`, segue-se por uma aresta `contato`, até atingir a aresta `nome`.

Na Lorel, as expressões de caminho apresentam particularidades de acordo com a cláusula em que são aplicadas. As variações são resumidas a seguir:

- **Cláusula `FROM`:** a ocorrência de um caminho de expressão na cláusula `FROM` pode ser ilustrada no seguinte exemplo:

  ```
  SELECT n
  FROM ListaTelefones.contato.nome n
  ```

  Uma alternativa que pode ser utilizada na cláusula `FROM` é a redução do caminho de expressão, comumente utilizado em expressões OQL. Esse preceito caracteriza-se pela definição de uma variável após cada rótulo presente na cláusula.

  ```
  FROM ListaTelefones.contato.nome,
       ListaTelefones.contato.endereço.rua
  ```

  Considerando a cláusula acima, é possível visualizar as mesmas expressões como se fossem conjuntos. Nesse contexto, um conjunto é uma expressão de caminho que pode ser utilizada como base de outras expressões de caminho, por exemplo: `ListaTelefones.contato`. Esse conjunto pode ser identificado com uma variável, assim como são definidas, em SQL, as variáveis que representam relações. Assim, se houver uma referência ao conjunto ou à variável que o representa, também haverá uma referência, nesse exemplo, a todos os contatos da lista de telefones. Adaptando a cláusula anterior, ela pode ser expressa dessa forma:

  ```
  FROM ListaTelefones.contato c, c.nome, c.endereço e, e.rua
  ```

  Nota-se que os rótulos podem ser conectados com variáveis previamente definidas, pois as variáveis simplesmente representam um conjunto definido sobre uma expressão de caminho.

- **Cláusula `SELECT`:** normalmente, a existência de uma expressão de caminho na cláusula `SELECT` reflete de uma expressão definida na cláusula `FROM`. Logo, se a expressão definida na cláusula `FROM` possuir uma variável vinculada, esta também poderá estar espelhada na cláusula

SELECT. Essa situação pode ser observada na tradução da seguinte consulta:

```
SELECT ListaTelefones.contato
FROM ListaTelefones.contato.telefone t
WHERE t = “334-3421”
```

Como há redundância de expressão de caminho nas duas cláusulas, pode ser definida para a expressão ListaTelefones.contato uma variável que represente o conjunto:

```
SELECT c
FROM ListaTelefones.contato c, c.telefone t
WHERE t = “334-3421”
```

Nesse exemplo foi observado que a expressão presente na cláusula FROM é mais complexa (extensa) que a presente na cláusula SELECT. A situação inversa também pode ser tratada utilizando recursão e aninhamento:

```
SELECT ListaTelefones.contato.endereco.cidade
FROM ListaTelefones.contato
```

A consulta apresentada acima objetiva encontrar os nomes das cidades dos contatos presentes na lista de telefones. Usando variáveis, a mesma consulta pode ser assim reescrita:

```
SELECT (SELECT x FROM c.endereco e, e.cidade x)
FROM ListaTelefones.contato c
```

A variável definida para a cláusula FROM da consulta também foi utilizada na cláusula FROM da consulta aninhada. Com base nessa variável, foi possível estabelecer expressões que alcançassem o resultado requerido, ou seja, o nome da cidade. Contudo, algumas ocorrências poderão resultar em conjuntos vazios. Existem contatos que possuem endereço não segmentado em elementos rua/número/cidade e contatos que não possuem endereço. Essas situações não resultarão em erro, em conjuntos vazios.

- **Cláusula WHERE:** assim como na cláusula SELECT, quando uma expressão de caminho na cláusula WHERE é um prefixo ou uma parte de uma expressão de caminho da cláusula FROM, mostra-se possível trocar a expressão por uma variável, como foi mostrado nos exemplos anteriores. Contudo, existem casos em que a expressão de caminho na cláusula WHERE não é um prefixo, como no exemplo a seguir:

```
SELECT ListaTelefones.contato
FROM ListaTelefones.contato
WHERE ListaTelefones.contato.endereço.numero = 1856
```

Nessa situação, uma forma de representar a consulta seria:

```
SELECT c
FROM ListaTelefones.contato c
```

```
WHERE exists e in c.endereço : exists n in e.numero : n = 1856
```

Essa consulta garante que retornarão os contatos que possuem ao menos um endereço, sendo que os endereços devem ter ao menos um número igual a 1856. Ainda existe a possibilidade de uma mesma expressão de caminho ocorrer mais de uma vez na cláusula `WHERE` sem aparecer na cláusula `FROM`:

```
SELECT ListaTelefones.contato.nome
FROM ListaTelefones.contato
WHERE ListaTelefones.contato.endereco.rua = ”Alberto Bins” or
     (ListaTelefones.contato.endereco.numero = 1856 and
      ListaTelefones.contato.endereco.cidade = “Porto Alegre”)
```

Lorel permite que essa mesma consulta seja representada da seguinte maneira:

```
SELECT o
FROM ListaTelefones.contato o
WHERE exists x in o.endereço : exists r in x.rua :
      exists n in x.numero : exists c in x.cidade :
      (r = “Alberto Bins” or (n = 1856 and c = “Porto Alegre”))
```

1.4.2 Expressões gerais de caminho

Expressões gerais de caminho são consideradas mais qualificadas que as expressões de caminho simples, pois permitem que *wildcards* sejam aplicados tornando mais avançado o processo de localização de objetos na base de dados OEM.

No primeiro exemplo a expressão busca atingir os objetos do tipo `telefone`. O grafo OEM, mostrado anteriormente, indica que logo após a aresta `contato` atinge-se a aresta e o objeto `telefone`. Caso ocorresse, entre essas duas arestas, a presença de uma outra, por exemplo `nrotelefone`, e o objetivo fosse chegar ao valor de todos os objetos `telefone`, a expressão a ser utilizada seria a seguinte:

```
ListaTelefones.contato(.nrotelefone)?.telefone
```

O termo `(.nrotelefone)?` indica opcionalidade. Então, atingem-se os objetos `telefone` com a existência ou não da aresta `nrotelefone`. O exemplo que segue refere-se à seguinte expressão:

```
ListaTelefones.contato.#@x.ender%.rua
```

O símbolo `#` indica existência de um número não conhecido de arestas com rótulos não especificados. A variável `@x` tem como função controlar a ocorrência desses rótulos indeterminados. A expressão `ender%` define que, necessariamente, existirá uma aresta com um rótulo iniciado com a string “`ender`”.

Existem outras expressões gerais de caminho que não estão aqui descritas. Destaca-se que essas expressões podem ser utilizadas normalmente em consultas Lorel. Contudo, as mesmas consultas que usam expressões gerais de caminho não podem ser traduzidas para OQL, pois a linguagem de consulta do modelo ODMG não possui suporte para esses tipos de expressão.

Visando potencializar a abrangência das expressões gerais de caminho, são disponibilizadas construções denominadas *wildcards*. Elas são utilizadas em situações nas quais o usuário não possui um completo conhecimento sobre os rótulos do modelo, muito menos sobre a ordem em que estes aparecem.

Quando o nome de um rótulo não é conhecido com exatidão ou se há rótulos com nomes semelhantes que resultam num mesmo valor de objeto, utiliza-se o *wildcard* "`%`". Supondo que o usuário tenha noção da existência de rótulos `tel` e `telefone`, que indicam o número de telefone dos contatos, ele pode realizar a seguinte consulta para obter o nome dos contatos:

```
SELECT ListaTelefones.contato.nome
WHERE ListaTelefones.contato.tel% = "332-2145"
```

Se o usuário nem mesmo soubesse os nomes dos rótulos, poderia utilizar o *wildcard* "`%`" para realizar a consulta. O *wildcard* "`#`" é utilizado para comparar caminho de dados de tamanho zero ou maior, sendo largamente combinado com variáveis de caminho.

Essas variáveis representam caminhos de dados no grafo OEM, ou seja, rótulos e objetos. Sobre variáveis de caminho normalmente se aplicam funções. Um exemplo é a função `path-of`, que retorna o caminho de dados presente numa string. A função permite obter a estrutura de dados representada no grafo OEM, por exemplo:

```
SELECT distinct path-of(x)
FROM ListaTelefones.#@x.cidade
```

Essa consulta resultará num conjunto de todos os caminhos presentes no banco de dados OEM que alcançam o rótulo `cidade`. No grafo que está sendo utilizado para as consulta, só haveria um caminho possível. Se o modelo apresentasse diferentes opções de atingir um elemento `cidade`, essas seriam recuperadas pela função.

1.4.3 Construção de resultados

Uma consulta Lorel, assim como em SQL e em OQL, resulta num *bag* (conjunto que pode apresentar repetições de elementos) ou num *set* (conjunto de elemento únicos, sem repetição) com a aplicação da palavra-chave `distinct`. Em consultas Lorel, onde não são utilizados aninhamentos, geram-se objetos OEM simples. A estrutura desses objetos é identificada pelo nome `answer`. Já o resultado é mostrado através das arestas de ligação que compõem esse novo objeto. Por exemplo:

```
SELECT c
FROM ListaTelefones.contato c
WHERE c.endereço.cidade = "Porto Alegre"
```

A consulta acima tem como objetivo retornar os contatos da lista de telefones que possuem ao menos um endereço cuja cidade seja Porto Alegre. O objeto OEM criado a partir dessa consulta seria:

```
answer &35
   contato &18
```

O objeto `&35` pode ser utilizado como entrada para outras consultas, porém é necessário que seja renomeado. Um exemplo de consulta que demonstra a utilização de múltiplas expressões na cláusula `SELECT` e seu correspondente resultado é o seguinte:

```
SELECT c.telefone, c.endereço
FROM ListaTelefones.contato c

answer &36
   contato &11
      telefone &13 "334-3421"
      endereço &14 "Av. Ipiranga"
   contato &15
      telefone &17 "321-8971"
   contato &18
      endereço &20
```

Além de realizar consultas, a linguagem Lorel possibilita atualizações. É possível criar e excluir nomes da base de dados e criar novos objetos atômicos ou complexos [ABI 97b]. Ainda, Goldman [GOD 99] define a forma de migração do modelo de dados semi-estruturado para o modelo XML no ambiente Lore, bem como as adaptações necessárias para que a linguagem tenha seu uso potencializado.

1.5 XML-QL

Extração, transformação e integração de dados são alguns dos problemas encontrados na área de banco de dados. Normalmente, utiliza-se uma linguagem de consulta relacional (SQL) ou orientada a objetos (OQL) como solução. Contudo, essas linguagens de consulta não podem ser aplicadas diretamente sobre documentos XML. Isso porque existem diferenças, por exemplo, entre a representação dos dados em XML e no modelo de dados relacional.

Alguns projetos de linguagens de consulta para dados semi-estruturados foram desenvolvidos, entre eles o que deu origem à linguagem Lorel. Uma proposta de linguagem de consulta que visa atender especificamente às necessidades do modelo de dados XML é a XML-QL (*Query Language for XML*) [DEU 98]. Trata-se de uma linguagem declarativa que combina elementos da sintaxe XML com uma sintaxe semelhante às tradicionais linguagens de consulta para bancos de dados.

1.5.1 Exemplos de consultas XML-QL

Nessa seção são apresentadas características da linguagem XML-QL mediante exemplos de consultas. O documento base para as consultas é o `listel.xml`, que representa os dados referentes aos contatos telefônicos de uma empresa de acordo com a seguinte DTD:

```
<!ELEMENT listatelefones (contato+)>
<!ELEMENT contato (nome, telefone, endereco?)>
<!ATTLIST contato tipo CDATA #REQUIRED>
<!ELEMENT nome (#PCDATA)>
<!ELEMENT telefone (#PCDATA)>
<!ELEMENT endereco (rua, numero?, cidade)>
<!ELEMENT rua (#PCDATA)>
<!ELEMENT numero (#PCDATA)>
<!ELEMENT cidade (#PCDATA)>
```

A DTD define que um elemento `listatelefones` possui um ou mais elementos `contato`. Já um elemento `contato` possui os elementos `nome` e `telefone`, que são requeridos e o elemento `endereco`, que é opcional. O elemento `contato` também possui um atributo `tipo` que informa se o contato é uma pessoa física ou uma pessoa jurídica. O elemento `endereco` possui os elementos `rua`, `cidade` e `numero`, sendo que este último é opcional. Os elementos `nome`, `telefone`, `rua`, `numero` e `cidade` recebem cadeias de caracteres.

```
<listatelefones>
   <contato tipo = ”Pessoa Física”>
      <nome>Maria</nome>
      <telefone>334-3421</telefone>
      <endereco>
         <rua>Av. Ipiranga</rua>
         <cidade>Porto Alegre</cidade>
      </endereco>
   </contato>
   <contato tipo = ”Pessoa Física”>
      <nome>Pedro</nome>
      <telefone>321-8971</telefone>
   </contato>
   <contato tipo = ”Pessoa Física”>
      <nome>Paula</nome>
      <telefone>271-1999</telefone>
      <endereco>
         <rua>Alberto Bins</rua>
         <numero>1856</numero>
         <cidade>Caxias do Sul</cidade>
      </endereco>
   </contato>
   <contato tipo = ”Pessoa Jurídica”>
      <nome>Enterprise</nome>
      <telefone>111-1212</telefone>
      <endereco>
         <rua>Bento Gonçalves</rua>
         <numero>14</numero>
         <cidade>Porto Alegre</cidade>
      </endereco>
   </contato>
</listatelefones>
```

O primeiro exemplo de consulta mostra o emprego de *padrões de elemento* [DEU 98] para verificar a ocorrência de valores em um documento XML. Essa consulta resulta no contato da lista de telefones que possui o número de telefone 334-3421 no documento `listel.xml`.

```
WHERE <listatelefones>
         <contato>
            <nome>$n</nome>
            <telefone>334-3421</telefone>
         </contato>
      </listatelefones> IN “listel.xml”
CONSTRUCT $n
```

Implicitamente, essa consulta busca em todos os elementos `<contato>` ao menos um elemento `<nome>` e um elemento `<telefone>` que possua o valor 334-3421. Para cada situação em que a consulta é satisfeita, associa-se à variável `$n` o nome corresponde ao contato. Logo, o resultado dessa consulta será uma lista de nomes limitados pela variável `$n`. Nota-se que a variável é precedida pelo símbolo `$`. Isso visa distinguir *strings* que compõem o documento XML e variáveis usadas pela XML-QL. Ainda, a linguagem permite substituir *tags* finais `</elemento>` por `</>`. Essa abreviação torna-se importante quando são utilizadas variáveis tag, tópico que será abordado no decorrer desta seção.

Como foi visto no primeiro exemplo, a consulta retorna uma lista com o nome dos contatos que possuem um determinado número de telefone. Contudo, em várias situações poderá ser importante construir um novo dado XML a partir do resultado obtido. Deve-se salientar que não é necessária a ocorrência da mesma estrutura do documento de entrada na construção da saída. No exemplo a seguir, busca-se agrupar sob o elemento `<resultadoconsulta>` os nomes dos contatos concentrados na variável `$n`.

```
WHERE <listatelefones>
         <contato>
            <nome>$n</>
            <telefone>334-3421</>
         </>
      </> IN "listel.xml"
CONSTRUCT
      <resultadoconsulta>
         <nomecontato>$n</>
      </>
```

A consulta acima tem o mesmo objeto da primeira, porém utiliza o recurso de abreviação de *tags* e apresenta uma nova estrutura para o resultado:

```
<resultadoconsulta>
   <nomecontato>Maria</nomecontato>
</resultadoconsulta>
```

No entanto, a construção de resultados não agrupa, no mesmo elemento, o nome de todos os contatos que possuem aquele número de telefone. Caso existissem outros contatos com número de telefone igual a 334-3421, os nomes apareceriam em diferentes elementos `<resultadoconsulta>`, e não sob um único elemento que agrupasse todos os nomes de contato.

No próximo exemplo é demonstrada uma forma de tratar a situação destacada acima: utilização de consulta aninhada. O objetivo é atingir os nomes dos contatos que possuem o número de telefone 334-3421 agrupados de acordo com a cidade. Ou seja, o intuito é recuperar os nomes e colocá-los sob um elemento `<resultadoconsulta>`, agrupados de acordo com o valor do elemento `<nomecidade>`.

```
WHERE <listatelefones>
         <contato>$x</>
      </> IN "listel.xml",

      <endereco>
         <cidade>$c</>
```

```
      </>
      <telefone>334-3421</> IN $x
CONSTRUCT
      <resultadoconsulta>
         <nomecidade>$c</>
         WHERE
            <nome>$n</> IN $x
         CONSTRUCT
            <nomecontato>$n</>
      </>
```

O primeiro passo refere-se à atribuição do conteúdo resultante do desmembramento do elemento `<contato>` à variável `$x`. Essa operação permite que a variável possa aparecer no lado direito da expressão `IN` ou em qualquer expressão `CONSTRUCT` aninhada.

No primeiro `WHERE` são selecionadas as cidades cujo contato possui o telefone 334-3421. No primeiro `CONSTRUCT` define-se que cada elemento `<resultadoconsulta>` possuirá um elemento `<nomecidade>` com um valor existente na variável `$c`. Se existirem diferentes cidades, ocorrerá, em mesmo número, o aparecimento de elementos `<resultadoconsulta>`. O segundo `WHERE` tem a função de retornar os nomes dos contatos que possuem o mesmo nome de cidade; o segundo `CONSTRUCT` apenas define qual é a forma de apresentação desses nomes. O resultado da consulta seria o seguinte:

```
<resultadoconsulta>
   <nomecidade>Porto Alegre</nomecidade>
   <nomecontato>Maria</nomecontato>
   <nomecontato>Enterprise</nomecontato>
</resultadoconsulta>
<resultadoconsulta>
   <nomecidade>Caxias do Sul</nomecidade>
   <nomecontato>Paula</nomecontato>
</resultadoconsulta>
```

Se uma estrutura necessita ser repetida ao longo da consulta, torna-se possível utilizar a cláusula `CONTENT_AS` cuja aplicação preserva o aninhamento original do documento. A seguir é apresentada a mesma consulta com o adendo da cláusula `CONTENT_AS`. Deve ser ressaltada a reorganização na cláusula `WHERE`:

```
WHERE <listatelefones>
         <contato>
            <endereco>
               <cidade>$c</>
            </>
            <telefone>334-3421</>
         </>
      </> CONTENT_AS $x IN "listel.xml"
CONSTRUCT
      <resultadoconsulta>
         <nomecidade>$c</>
         WHERE
            <nome>$n</> IN $x
         CONSTRUCT
            <nomecontato>$n</>
      </>
```

A próxima consulta exemplifica a ocorrência de *joins* em XML-QL. Essa operação é expressa através da comparação de elementos que possuem um mesmo valor. A consulta busca os nomes das cidades desde que essas apareçam tanto nos contatos de tipo pessoa física quanto nos contatos de tipo pessoa jurídica. Segue o exemplo:

```
WHERE <listatelefones>
         <contato tipo = "Pessoa Jurídica">
            <endereco>$x</>
         </>
      </> IN "listel.xml",
      <cidade>$c</> IN $x,

      <listatelefones>
         <contato tipo = "Pessoa Física">
            <endereco><cidade>$c</></>
         </>
      </> IN "listel.xml"
CONSTRUCT
      <resultadoconsulta>
         <nomecidade>$c</>
      </>
```

A junção ocorre através da utilização da variável `$c` nas duas partes que compõem a cláusula `WHERE`. Ainda na mesma consulta, existe diferença entre o elemento de entrada e a estrutura de saída produzida pelo `CONSTRUCT`. Caso as duas estruturas sejam iguais, pode ser usada a cláusula `ELEMENT_AS`, que armazena numa variável o valor de entrada na forma de elemento. Assim, a mesma estrutura pode ser utilizada para a construção do resultado:

```
WHERE <listatelefones>
         <contato>
            <nome>$n</>
            <telefone>111-1212</>
         </>
     </> ELEMENT_AS $x IN "listel.xml"
CONSTRUCT
      $x
```

O resultado possui a mesma estrutura da entrada proposta na cláusula `WHERE`:

```
<listatelefones>
   <contato>
      <nome>Enterprise</>
      <telefone>111-1212</>
   </contato>
</listatelefones>
```

Até agora, os exemplos apresentados operaram e recuperaram os conteúdos de elementos. Essas ações são realizadas com o emprego de variáveis de iteração. Para trabalhar com os nomes dos elementos, a XML-QL disponibiliza uma outra classe de variáveis: as *variáveis tag*. Para exemplificar, a seguinte consulta busca o número de telefone dos contatos cuja cidade ou rua tenham o nome Caxias do Sul:

```
WHERE <listatelefones>
         <contato>
            <telefone>$t</>
            <endereco>
               <$x>Caxias do Sul</>
            </>
         </>
      </> IN “listel.xml”,
      $x  IN {rua, cidade}
CONSTRUCT
      <resultadoconsulta>
         <numerotelefone>$t</>
         <$x>Caxias do Sul</>
      </>
```

A variável `$x` irá conter o elemento `<rua>` ou o elemento `<cidade>`. Na cláusula `CONSTRUCT` também pode ser aplicada a variável tag sem restrições. Essa consulta apenas retornará para variável tag o elemento `<cidade>`. Se o documento `listel.xml` possuísse alguma rua cujo nome fosse Caxias do Sul, a consulta também retornaria os elementos `<rua>`.

Outra característica avançada disponibilizada pela XML-QL é a possibilidade de concepção de uma visão integrada a partir da combinação de fontes de dados diferentes. Exemplo: considera-se que a consulta será realizada sobre dois documentos, `listel.xml` e `lisban.xml`. Este último possui dados bancários de alguns dos contatos presentes no primeiro documento. O elemento `<nome>` ocorre tanto no primeiro quanto no segundo documento. O objetivo dessa consulta é estabelecer um resultado que una o número de telefone (primeiro documento) e o número da conta bancária (segundo documento). Para isso, deve ser realizada a união das duas fontes através do nome do contato:

```
WHERE <listatelefones>
         <contato>
            <nome>$n</>
            <telefone></> ELEMENT_AS $t
         </>
      </> IN “listel.xml”,

      <listaclientes>
         <cliente>
            <nome>$n</>
            <numeroconta></> ELEMENT_AS $c
         </>
      </> IN “lisban.xml”
CONSTRUCT
      <resultadoconsulta>
         <nome>$n</>
         $t $c
      </>
```

Caso os dados bancários do contato Pedro estivessem presentes no documento `lisban.xml`, o resultado seria assim expresso:

```
<resultadoconsulta>
```

```
    <nome>Pedro</nome>
    <telefone>321-8971</telefone>
    <numeroconta>57.396-739</numeroconta>
</resultadoconsulta>
```

A XML-QL apresenta outras funcionalidades além destas. Destaca-se a possibilidade de definição de funções. Novas funções permitem adequar a linguagem de acordo com cada aplicação proposta pelo usuário. Todavia, algumas características não foram incluídas nesta versão, tais como o suporte a entidades definidas na DTD, predicados definidos pelo usuário que permitem o emprego de linguagens externas como Java e Pearl, suporte a agregados e *semijoins*, entre outras.

1.6 XQL

A XQL (*XML Query Language*) é uma linguagem projetada especialmente para XML [ROB 98], cujo núcleo é fundamentado na XPath (*XML Path Language*) [CLA 99, MAL 2001]. Ela fornece uma descrição bastante compreensível dos elementos, bem como uma maneira direta de consultá-los. Para navegar na hierarquia de elementos dos documentos, ela utiliza uma notação de caminho semelhante à das URL (*Uniform Resource Locator*). A XQL foi projetada para ser usada em diferentes ambientes XML, assim como adaptada em linguagens de programação. Logo, ela é extremamente útil para a recuperação de informações de um documento. O resultado de uma consulta XQL pode ser um elemento, uma lista deles ou um documento XML completo.

Em agosto de 1999, foi lançada uma nova especificação da linguagem [ROB 99]. Nela foram adicionadas funções de extensão, suporte a *links* e, principalmente, *joins*. Tais modificações surgiram a partir de discussões ocorridas no W3C QL '98 Workshop, que também definiu a necessidade de combinar informações oriundas de fontes heterogêneas. Uma das preocupações dos autores era manter a simplicidade da linguagem mesmo com a adição de funções complexas.

A existência de um modelo de dados XML próprio foi uma das motivações para o desenvolvimento do projeto da linguagem XQL. Esse modelo não é semelhante ao modelo relacional, nem mesmo ao modelo orientado a objetos. Em XQL, um documento é uma árvore rotulada onde os nodos representam entidades, elementos, atributos, etc. É importante ressaltar que os relacionamentos entre dados contêm uma grande quantidade de informações. Essa é uma das razões por que documentos estruturados no formato XML são tão explorados atualmente [ROB 99].

Pode-se notar que os nodos possuem um papel central em consultas XQL. Esses nodos que formam a entrada de uma consulta podem ser originários de diferentes fontes; podem ser o resultado de uma consulta anterior; podem ser o conteúdo de um repositório de documentos ou de qualquer outra fonte, desde que sua origem seja identificada. Uma inovação apresentada na nova especificação XQL é o emprego de junções entre conjuntos de nodos que permite que subárvores de uma fonte de dados sejam inseridas numa nova subárvore definida na condição do *join* (exemplificado adiante). Outras funcionalidades da linguagem são apresentadas através de exemplos ao longo desta seção.

1.6.1 Exemplos de consultas XQL

Antes de detalhar a linguagem XQL, serão apresentados exemplos de consultas Em cada exemplo é dada uma idéia rápida do objetivo de cada consulta e a sua

correspondente sintaxe. O documento que serve de base para as consultas é o mesmo discutido na seção sobre a XML-QL.

Para retornar todos os elementos `<nome>`, basta especificar uma *string* simples com o nome do elemento requerido:

```
nome
```

O uso do operador "`/`" indica existência de hierarquia entre elementos. Se for necessário retornar todos os elementos `<telefone>` que são filhos de elementos `<contato>`, define-se a seguinte consulta:

```
contato/telefone
```

Para representar consultas que partem do elemento raiz do documento, fez-se necessário indicar o operador à frente do caminho de localização dos elementos:

```
/listatelefones/contato/telefone
```

Para especificar o conteúdo de um elemento ou o valor de um atributo, utiliza-se o operador "`=`". Nessa consulta são retornados todos elementos `<nome>` cujo nome do contato seja igual a Paula:

```
contato/nome = "Paula"
```

Para comparar o valor de um atributo, o nome deste deve ser precedido pelo símbolo "`@`". Os atributos são tratados como se fossem elementos-filho:

```
contato/@tipo = "Pessoa Física"
```

O operador "`//`" indica um número qualquer de níveis de elemento para atingir o nodo especificado. Por exemplo, buscam-se todos os elementos `<cidade>`, independentemente da sua localização no documento, partindo do nodo raiz:

```
/listatelefones//cidade
```

Quando o operador "`//`" é encontrado no início de um caminho, significa que todos os elementos definidos após o operador serão recuperados independentemente da sua posição no documento. A seguinte consulta retorna qualquer elemento cidade que esteja presente no documento:

```
//cidade
```

O operador "`[]`" filtra o conjunto de nodos baseado na condição definida em seu interior. A consulta a seguir retorna os contatos, sendo que cada um desses contatos deve possuir um atributo chamado `tipo`, com o valor Pessoa Jurídica:

```
/listatelefones/contato[@tipo = "Pessoa Jurídica"]
```

A consulta anterior retorna os subelementos de `<contato>`. Já a consulta `contato/@tipo = "Pessoa Jurídica"` retorna o conteúdo do atributo `tipo`, e não os

contatos e seus subelementos. Apesar da similaridade, elas retornam resultados completamente diferentes.

Os dois próximos exemplos mostram a consulta e a respectiva representação dos resultados. Novamente será utilizado o documento XML apresentado na seção sobre XML-QL.

1. Retornar todos os números de telefones existentes no documento.

```
//telefone

<xql:result>
   <telefone>334-3421</telefone>
   <telefone>321-8971</telefone>
   <telefone>271-1999</telefone>
   <telefone>111-1212</telefone>
</xql:result>
```

2. Retornar o nome do contato cujo número de telefone seja 111-1212.

```
//contato[telefone = "111-1212"]//nome

<xql:result>
   <nome>Enterprise</nome>
</xql:result>
```

Uma consulta XQL sempre é aplicada sobre um **contexto**, que é uma lista de nodos. As expressões mostradas na **TABELA 2.1** são chamadas de **termos**. Eles têm a função de selecionar nodos com características particulares dentro de um contexto, baseado-se no tipo ou no nome do nodo.

TABELA 2.1 - Termos

Termo	Descrição	Função
N	nome do elemento	Todos os nodos do contexto onde o tipo do nodo é elemento e seu nome é "n"
*	nome do elemento com wildcards	Todos os nodos do contexto onde o tipo do nodo é elemento
@n	nome do atributo	Todos os nodos do contexto onde o tipo do nodo é atributo e seu nome é "n"
@*	nome do atributo com wildcards	Todos os nodos do contexto onde o tipo do nodo é atributo
Text()	nodo de texto	Todos os nodos do contexto onde o tipo do nodo é texto

Em expressões XML, nomes podem ser associados com prefixos de *namespaces* [BRA 2000, MAR 2000]. O uso de *namespaces* é fundamental quando da integração de duas ou mais fontes. O seguinte exemplo mostra a declaração de uma variável que será equivalente ao URI (*Uniform Resource Identifier*) `listel.xml` e sua utilização numa consulta que busca todos elementos `<contato>` daquele *namespace*:

```
x := "www.ti/exemplo.xml";
//x:contato
```

Independentemente do uso ou não de *namespaces*, o conceito de comparação está presente na XQL e refere-se à adição de *constraints* baseadas no valor de um nodo. Nos exemplos abaixo, o tipo do nodo está no lado esquerdo da expressão. Conseqüentemente, o valor a ser comparado está à direita:

```
contato = "Enterprise"
@tipo = "Pessoa Física"
endereco[numero < 20]
```

Os operadores de comparação da XQL são apresentados na **TABELA 2.2**.

TABELA 2.2 – Operadores de comparação

Comparação	Sintaxe
Igualdade	`x = "valor"`
	`x eq "valor"`
Comparação *case insensitive*	`x ieq "valor"`
Desigualdade	`x != "valor"`
	`x ne "valor"`
Verificação de conteúdo	`x contains "valor"`
Verificação do conteúdo *case insensitive*	`x icontains "valor"`

Termos e outras expressões XQL podem ser combinados através da aplicação de operadores lógicos e de operadores sobre conjuntos. A **TABELA 2.3** apresenta alguns operadores disponíveis em XQL.

TABELA 2.3 – Operadores lógicos e de conjuntos

Sintaxe	Função	Resultado
`not (e)`	Negação	Todos os nodos no contexto para que a expressão avaliada seja nula
`e1 union e2`	União	União de e1 e e2
`e1 intersect e2`	Intersecção	Interseção de e1 e e2
`e1 | e2`	União	União de e1 e e2
`e1 ~ e2`	Ambos	Se ambos não são vazios, retornar e1 unido com e2 se vazio, retornar uma lista vazia
`e1 or e2`	Ou	Se a união entre e1 e e2 não for vazia, retorna verdadeiro, senão, retorna falso
`E1 and e2`	E	Se a interseção entre e1 e e2 não for vazia, retorna verdadeiro, senão, retorna falso

O operador "`~`" foi proposto em XQL porque algumas consultas usam filtros para expressar *constraints* de uma mesma informação que é retornada fora do filtro, resultando em expressões redundantes. A consulta abaixo utiliza filtros para expressar

que apenas os contatos com endereço de Caxias do Sul e que possuem um número de telefone são requeridos, além do que, tanto o endereço quanto o número do telefone devem ser retornados:

```
//contato[endereco[cidade = "Caxias do Sul"] and .//telefone]
/(endereco | .// telefone)
```

O operador ".//" indica qualquer telefone que esteja dentro do contexto (o ponto indica o mesmo contexto). Usando o operador "~", a mesma consulta pode ser representada de uma forma mais concisa:

```
//contato/(endereco[cidade = "Caxias do Sul"] ~ .//telefone)
```

Muitas vezes pode ser importante agrupar os resultados de uma consulta usando a estrutura original do documento. Supondo que no documento existissem contatos com mais de um telefone, deseja-se recuperar a listagem dos telefones de cada contato colocando cada grupo de telefones sob a tag `<contato>` correspondente. Na sintaxe, o elemento à esquerda das chaves é chamado de **elemento de agrupamento,** que é usado para agrupar os resultados da consulta definida dentro das chaves:

```
//contato { .//telefone }
```

Para cada elemento de agrupamento encontrado pela consulta, o operador de agrupamento cria um elemento vazio com o mesmo nome. O resultado da consulta contido dentro das chaves é incorporado a esse elemento vazio na forma de subelementos. Na prática, o resultado seria este:

```
<xql:result>
   <contato>
      <telefone>334-3421</telefone>
      <telefone>999-9999</telefone>
   </contato>
   <contato>
      <telefone>321-8971</telefone>
   </contato>
   <contato>
      <telefone>271-1999</telefone>
   </contato>
   <contato>
      <telefone>111-1212</telefone>
      <telefone>555-5555</telefone>
   </contato>
</xql:result>
```

Além dos operadores, a XQL também propõe funções (**TABELA 2.4**), juntamente com o conceito de **funções extensíveis**. Estas são escritas de acordo com a necessidade do usuário e podem ser adaptadas às consultas. A implementação das funções deve ser realizada na mesma linguagem em que a implementação da XQL foi feita. Algumas linguagens que podem ser utilizadas para esse fim são C++, Java, Perl, entre outras.

TABELA 2.4 – Funções XQL

Nome da função	Descrição
`attribute(), attribute('nome')`	Retorna os atributos no contexto. Se o parâmetro "nome" é fornecido, retorna os atributos com aquele nome
`element(), element('nome')`	Retorna os elementos no contexto. Se o parâmetro "nome" é fornecido, retorna os elementos com aquele nome
`node()`	Retorna todos os nodos do contexto
`count()`	Retorna o número de referências aos nodos que aparecem num conjunto de elementos

Como salientado no início desta seção, o emprego de junções é uma das novas características da XQL. O objetivo é combinar informações de múltiplas fontes e criar uma visão única das mesmas. Por exemplo, se houver o interesse de combinar as informações da lista de telefones com o cadastro bancário de clientes:

```
<dadosbancarios>
   <cliente tipo = "Pessoa Física">
      <nome>Maria</nome>
      <numeroconta>2854839593</numeroconta>
   </cliente>
   <cliente tipo = "Pessoa Física">
      <nome>Maria</nome>
      <numeroconta>9573064938</numeroconta>
   </cliente>
   <cliente tipo = "Pessoa Física">
      <nome>Paula</nome>
      <numeroconta>1953759702</numeroconta>
   </cliente>
   <cliente tipo = "Pessoa Jurídica">
      <nome>Enterprise</nome>
      <numeroconta>7940285943</numeroconta>
   </cliente>
</dadosbancarios>
```

Os dois documentos possuem o elemento `<nome>` fundamental para a realização da junção. A idéia é agregar a estrutura da lista de telefones com o número da conta bancária do contato. Se um contato possui apenas uma conta bancária, o resultado desejado pode ser conseguido através da consulta:

```
/listatelefones/contato { nome | telefone |
/dadosbancarios/cliente { numeroconta } }
```

Já, no documento de dados bancários, existe mais de uma ocorrência de conta bancária para o mesmo cliente. Para realizar a consulta, é necessário restringir os dados bancários àqueles contatos que possuem o mesmo nome dos clientes. Aqui é introduzido o conceito de correlação de variáveis. No exemplo, `$n:=nome` indica que a variável "`$n`" recebe o valor do nome para cada contato do contexto. A expressão

`/dadosbancarios/cliente[nome = $n]` restringe os dados bancários para aqueles que correspondem ao valor presente em "$n".

```
/listatelefones/contato[$n := nome] { nome | telefone
/dadosbancarios/cliente[nome = $n] { numeroconta }}
```

O resultado seria algo semelhante a essa estrutura:

```
<listatelefones>
   <contato tipo = "Pessoa Física">
      <nome>Maria</nome>
      <telefone>334-3421</telefone>
      <numeroconta>2854839593</numeroconta>
      <numeroconta>9573064938</numeroconta>
   </contato>
   <contato tipo = "Pessoa Física">
      <nome>Paula</nome>
      <telefone>271-1999</telefone>
      <numeroconta>1953759702</numeroconta>
   </contato>
   <contato tipo = "Pessoa Jurídica">
      <nome>Enterprise</nome>
      <telefone>111-1212</telefone>
      <numeroconta>7940285943</numeroconta>
   </contato>
</listatelefones>
```

Algumas potencialidades da linguagem XQL foram mostradas nessa seção. Consultas simples, baseadas em filtragens pouco complexas e agrupamentos, são facilmente expressas e compreendidas. Contudo, a XQL dificulta o processo de expressar consultas mais complexas (com aninhamento, por exemplo). O implemento de *joins* pode ser saudado como uma interessante inovação, porém é obscurecido pela dificuldade de expressão. Salienta-se a vantagem que o usuário tem de estender a linguagem base com o adendo de funções extensíveis.

1.7 XQuery

Uma das grandes vantagens de utilização da XML concentra-se na flexibilidade de representar variados tipos de informação. Essas informações podem, por exemplo, estar armazenadas em bancos de dados ou representadas através de documentos. Logo, uma boa linguagem de consulta deve ser capaz de recuperar dados presentes nessas diferentes fontes.

Dentro deste contexto, a XQuery [CHA 2001] pode ser considerada uma linguagem eficiente para a recuperação de dados, principalmente XML. Foi projetada tendo como base os requisitos propostos pelo **W3C XML Query Workgroup** [CHA 2001a] para consultar diferentes fontes XML, incluindo a combinação existente entre bancos de dados e documentos XML.

A XQuery é derivada diretamente de uma linguagem XML proposta por Chamberlin, Robie e Florescu, denominada Quilt [CHA 2000, ROB 2000], que tem como destaque a conjunção de aspectos relevantes de outras linguagens. São eles: da XPath e da XQL, herda a sintaxe baseada em expressões de caminho suportadas pela

hierarquia do documento; da XML-QL, herda o uso de variáveis de ligação para criação de novas estruturas; da SQL, utiliza a seqüência de cláusulas fundamentadas em palavras-chave (estilo semelhante às cláusulas `SELECT-FROM-WHERE`) e, da OQL, herda a noção de linguagem funcional, na qual expressões podem ser especializadas através de aninhamentos.

Toda a entrada e saída de uma consulta XQuery seguem o padrão definido pelo modelo de dados XML [FER 2001], no qual um documento é modelado como uma árvore de nodos. Para consultar esses nodos a XQuery utiliza-se de um conjunto bem definido de expressões, as quais serão ilustradas a seguir.

1.7.1 Exemplos de consultas XQuery

As expressões disponibilizadas pela XQuery serão aqui apresentadas através de exemplos. O documento que representa uma lista de contatos de uma empresa continua sendo a base dos exemplos de consulta.

Uma consulta XQuery, necessariamente, parte da definição de expressões de caminho. Elas são representadas através de expressões XPath abreviadas (uso concentrado, especialmente, na localização de nodos). O resultado da aplicação de uma expressão de caminho é uma lista de nodos que respeita a hierarquia original do documento.

Como apresentado preliminarmente na seção sobre XQL, as expressões de caminho são compostas por passos que correspondem ao caminhamento realizado sobre o documento. Cada passo pode sofrer alguma forma de condicionamento (uso de predicados) e cada passo resulta numa lista de nodos que é utilizada como entrada para o passo seguinte. Convenciona-se utilizar a função `document` antes da expressão de caminho. Através dela, é definido o documento sobre o qual a expressão será aplicada.

Para explanar o funcionamento de uma expressão de caminho completa (passos, predicados, funções, ...), considera-se que a partir do documento `listel.xml` é necessário atingir o nodo `telefone`, cujo valor é 271-1999. Esse nodo é filho de um elemento `contato` cujo atributo `tipo` tem valor igual a Pessoa Física e descende indiretamente do nodo raiz `listatelefones`:

```
document("listel.xml")/listatelefones/
   contato[@tipo = "Pessoa Física"]/telefone[. = "271-1999"]
```

Após a declaração da função `document`, estão claramente destacadas as partes constituintes da expressão. Excetuando o primeiro "`/`", os demais símbolos delimitam os passos da expressão. O primeiro passo apenas indica que o elemento `listatelefones` é raiz do documento, por conseqüência, da expressão. O segundo e o terceiro, além de considerarem a existência dos respectivos nodos, também aplicam filtros específicos para a respectiva lista de nodos recuperada. O símbolo "`.`", presente no segundo predicado, denota comparação do nodo corrente (`telefone`) com um determinado valor (271-1999).

As expressões de caminho podem ser qualificadas através do uso de *namespaces*, da mesma forma que, em XQL, o prefixo do *namespace* é associado ao URI do documento. Isso permite integrar fontes distintas dentro de uma simples expressão de caminho. No exemplo está sendo considerado que o documento `listel.xml` é identificado pelo URI `www.exemplo.com/listel`. Ressalta-se a referência ao prefixo dentro da expressão:

```
NAMESPACE lista = “www.exemplo.com/listel”
document(“listel.xml”)//lista:endereco/rua
```

Sabe-se que a definição de uma consulta parte de uma expressão de caminho que determina a lista de nodos a ser avaliada. Todavia, a definição de expressões de caminho não se restringe apenas a esse uso, podendo ser utilizadas em diferentes pontos e cláusulas de uma consulta.

`FOR`, `LET`, `WHERE` e `RETURN` são as palavras-chave que definem as cláusulas numa consulta XQuery. Sem exceção, as expressões de caminho podem ser encontradas em qualquer uma das cláusulas, especialmente na cláusula `FOR`. Nela, variáveis assumem os nodos recuperados pelas expressões de caminho, sendo que cada variável está somente ligada a uma expressão. O mesmo ocorre na cláusula `LET`, com uma diferença: os valores de uma expressão estão simplesmente ligados a uma variável. Já, na cláusula `FOR`, ocorrem iterações sobre as nodos recuperados pela expressão. Considere o exemplo:

```
FOR $var IN document(“listel.xml”)/listatelefones/contato
LET $var := document(“listel.xml”)/listatelefones/contato
```

A variação sintática das duas cláusulas é mínima, no entanto o resultado obtido é substancialmente diferente. Enquanto na primeira cláusula os contatos são tratados individualmente, na segunda a variável `$var` assume todos os contatos como um objeto único de retorno. Se for proposto um adendo à expressão solicitando a recuperação dos elementos `contato` que possuem o subelemento `nome`, o uso da cláusula `FOR` refletirá na recuperação apenas daqueles nodos `contato` que possuem um filho `nome`, como estruturas únicas e independentes. Se a mesma expressão é integrante de uma cláusula `LET`, a ocorrência de um `contato` com filho `nome` resultará numa lista completa de todos os contatos que satisfazem a condição, porém de uma forma agrupada e não individualizada.

Ambas as palavras-chave, assim como a `WHERE` e a `RETURN`, não são sensíveis ao caso, diferentemente das variáveis e dos nomes que compõem as expressões de caminho. Uma consulta pode possuir várias cláusulas `FOR` e `LET`, sendo que o resultado de cada cláusula é uma lista ordenada de tuplas ligadas às variáveis. Se a palavra `DISTINCT` for aplicada sobre uma expressão/cláusula e existirem duplicatas, a lista resultado deixará de ser ordenada.

Sendo ou não ordenadas, as listas associadas às variáveis definidas nas cláusulas `FOR` e `LET` podem sofrer uma “filtragem” com o emprego da cláusula `WHERE`, que é opcional e pode conter vários predicados conectados por operadores lógicos. Os predicados a serem aplicados dependem muito dos valores assumidos pelas variáveis. Se a variável é resultado de uma cláusula `FOR`, os predicados são aplicados sobre nodos simples. Todavia, se a variável decorre de uma cláusula `LET`, os predicados devem ser adaptados para o uso sobre listas de nodos.

Todos os nodos/tuplas que satisfazem às condições impostas na cláusula `WHERE` podem ser utilizados na cláusula `RETURN`. Nela são geradas as saídas da consulta, que vão desde simples valores atômicos até conjuntos ordenados de nodos. Existe a possibilidade da construção de elementos e atributos com ou sem referência às variáveis.

Simplificando, essa é a estrutura de cláusulas da XQuery, também conhecida por **expressões FLWR** (FOR, LET, WHERE e RETURN). Para caracterizar o uso dessas expressões, são apresentados três exemplos:

```
FOR $x IN document("listel.xml"),
    $y IN $x//contato
WHERE $y//cidade = "Caxias do Sul"
RETURN $y
```

Nessa consulta o destaque está na declaração da cláusula FOR. Em uma primeira análise, fica claro que uma expressão de caminho não precisa, obrigatoriamente, estar vinculada à função document. Na consulta, todo o conteúdo do documento é associado à variável $x sem uso de qualquer expressão de caminho. Em uma situação como esta, o ideal seria utilizar a cláusula LET para simplesmente atribuir o conteúdo do documento à variável. Mesmo assim, a atual representação não é errônea, no entanto é pouco otimizada. Na segunda parte da cláusula FOR é aplicada uma expressão sobre a lista de nodos delimitada pela variável $x, resultando na variável $y. Já, na cláusula WHERE, um predicado simples de comparação é proposto, destacando que uma expressão de caminho é agora aplicada sobre a variável $y. Finalmente, todo o conteúdo da variável $y que satisfaz à condição especificada na cláusula WHERE é apresentado como resultado da consulta:

```
<contato tipo="Pessoa Fisica">
  <nome>Paula</nome>
  <telefone>271-1999</telefone>
  <endereco>
    <rua>Alberto Bins</rua>
    <numero>1856</numero>
    <cidade>Caxias do Sul</cidade>
  </endereco>
</contato>
```

O segundo exemplo vale-se de uma expressão de caminho na cláusula LET. Comparando com a primeira consulta, essa configuração dispensa o emprego das cláusulas FOR e WHERE. Analisando a expressão, vê-se que ela tem como objetivo filtrar a lista de nodos que serve de entrada para consulta. Assim, constata-se que condições não precisam ser unicamente definidas na cláusula WHERE, cabendo ao usuário, de acordo com a situação, escolher a melhor alternativa. Na cláusula RETURN é apresentada a possibilidade de construção de elementos e atributos. Se houvesse uma simples referência à variável $x, como no exemplo anterior, o resultado estaria encapsulado entre elementos <telefone>. Com essa construção, o que era conteúdo de elemento no documento torna-se valor de atributo na saída. Além desta, inúmeras variações de construções de resultados podem ser criadas, com ou sem o uso de variáveis. A consulta que recupera o número do telefone do contato Maria e o seu respectivo resultado construído a partir da cláusula RETURN são assim representados:

```
LET $x := document("listel.xml")//contato[nome = "Maria"]/telefone
RETURN <Maria telefone={$x}/>
```

Resultado:

```
<Maria telefone="334-3421" />
```

A terceira consulta pretende agrupar os telefones de acordo com a cidade, sendo que o resultado deve estar contido dentro de elementos <cidades_telefones> e o conjunto de telefones deve estar agrupado em elementos cujos nomes sejam das respectivas cidades. Para alcançar esse objetivo, inicialmente é construído o elemento base do resultado. Construções dessa espécie são comuns na cláusula RETURN, porém podem ser instituídas em outras partes da consulta desde que os símbolos "{ }" indiquem a existência de expressões XQuery avaliáveis. Caso os símbolos fossem omitidos, nenhuma consulta seria analisada e o resultado seria a própria declaração da consulta delimitada, simplesmente, pelos elementos <cidades_telefones>. Nessa mesma consulta é ilustrado o uso das duas cláusulas (LET e FOR) de forma cooperativa, visto que a iteração a ser executada sobre a variável $y depende do conteúdo previamente destacado pela cláusula LET na variável $x. Na cláusula RETURN do primeiro nível da consulta, aparecem duas novidades: a construção explícita de um elemento com base em uma variável e a definição de uma subconsulta com variáveis definidas em um nível superior. A construção do elemento serve para definir os elementos com os nomes das cidades e a subconsulta serve para popular esses novos elementos com os subelementos que representam os números de telefones. A consulta discutida e o seu concernente resultado são os seguintes:

```
<cidades_telefones>
{
LET $x := document("listel.xml")/listatelefones/contato
FOR $y IN distinct($x//cidade)
RETURN <{$y}>
       {
       FOR $z IN $x
       WHERE $z/endereco/cidade = $y
       RETURN $z/telefone
       }
       </{$y}>
}
</cidades_telefones>
```

Resultado:

```
<cidades_contatos>
  <Porto Alegre>
    <telefone>334-3421</telefone>
    <telefone>111-1212</telefone>
  </Porto Alegre>
  <Caxias do Sul>
    <telefone>271-1999</telefone>
  </Caxias do Sul>
</cidades_contatos>
```

Em qualquer dos exemplos, se houvesse necessidade de controlar a ordem dos elementos poderia ser utilizada a cláusula SORTBY. A seqüência a ser ordenada precisa representar toda a consulta ou apenas uma expressão dentro da mesma. A cláusula SORTBY pode conter uma ou mais expressões de ordenação, que podem, ou não, ser seguidas pelas palavras ASCENDING e DESCENDING. A declaração da cláusula SORTBY pode ser feita em qualquer ponto da consulta, desde que após as cláusulas LET e FOR.

```
<resultado>
{
FOR $x IN document("listel.xml")//telefone
SORTBY(.)
WHERE $x != "111-1212"
RETURN <{$x/../nome} telefone={$x}/>
}
</resultado>
```

A consulta acima demonstra o uso da cláusula `SORTBY`. Na variável `$x` estão listados os telefones que correspondem à saída desejada para a consulta. Apenas o número 111-1212 deve ser desconsiderado. A ordem em que os elementos construídos serão retornados é definida pela expressão de ordenação que se refere ao próprio elemento `telefone`, justificando o porquê da expressão simplificada `"."`. Por padrão, assume-se que a ordem a considerar é ascendente, decorrente da não-definição das palavras `ASCENDING` e `DESCENDING`. Abaixo estão descritos os resultados com o uso de `SORTBY (1)` e aquele que mantém a ordem original do documento `(2)`:

```
(1)                                (2)
<resultado>                        <resultado>
  <Paula telefone="271-1999" />      <Maria telefone="334-3421" />
  <Pedro telefone="321-8971" />      <Pedro telefone="321-8971" />
  <Maria telefone="334-3421" />      <Paula telefone="271-1999" />
</resultado>                       </resultado>
```

Desconsiderando a ordenação, em nenhuma das consultas houve necessidade de tratamento especial para os resultados recuperados. Todavia, se fosse preciso aplicar uma condição sobre um determinado tipo de valor retornado, o mesmo poderia ser feito através de **expressões condicionais**. Para elucidar o uso dessas expressões tem-se o exemplo:

```
<resultado>
{
FOR $x IN document("listel.xml")//contato[endereco]
RETURN IF ($x[@* = "Pessoa Fisica"]) THEN
          <PF nome={$x/nome} telefone={$x/telefone}/>
       ELSE
          <PJ nome={$x/nome} telefone={$x/telefone}/>
}
</resultado>
```

Na variável `$x` estão aqueles contatos que possuem um elemento filho denominado `endereco`. O que se deseja é recuperar o nome e o telefone desses contatos e apresentá-los como atributos de um elemento. Este possui o nome alusivo ao tipo de pessoa ao qual o contato corresponde: se pessoa física, `<PF>`, ou se pessoa jurídica `<PJ>`. Nesse ponto se enquadra a expressão condicional, a qual define que qualquer atributo (nesse caso só existe um) que possua o valor `"Pessoa Física"` terá o resultado representado pelo elemento `<PF>`; do contrário, a representação da saída se dará pelo elemento `<PJ>`:

```
<resultado>
  <PF nome="Maria" telefone="334-3421" />
  <PF nome="Paula" telefone="271-1999" />
  <PJ nome="Enterprise" telefone="111-1212" />
```

```
</resultado>
```

Expressões condicionais podem utilizar funções em suas declarações. Além das já conhecidas funções `document` e `distinct`, a XQuery contém todas as funções presentes na XPath, bem como todas as funções de agregação existentes na SQL (`avg`, `count`, `sum`, ...). Além dessas, a linguagem possibilita que o usuário defina suas próprias funções. Para cada função criada devem ser definidos os seus parâmetros e o tipo de retorno. Obrigatoriamente, deve possuir uma expressão ou uma consulta completa como "corpo" da função. A declaração e o emprego de uma função podem ser assim vislumbrados:

```
DEFINE FUNCTION retornarRua(LIST $l) RETURNS rua
{
   FOR $a in $l
   RETURN IF (empty($a//rua)) THEN
             <nao_ha_rua_definida/>
          ELSE
             $a//rua
}

<resultado>
{
FOR $x IN document("listel.xml")//contato
RETURN retornarRua($x)
}
</resultado>
```

Na definição da função `retornarRua` está sendo previsto o retorno dos elementos `rua` com base numa lista de nodos que serve de parâmetro de entrada. A especificação da XQuery informa que o tipo do parâmetro deve estar sempre associado à variável que o representa. No entanto, as atuais implementações dispensam essa convenção, admitindo parâmetros de qualquer espécie. No corpo da função é declarada uma consulta que resulta num conjunto de elementos que podem ser utilizados em novas funções ou consultas. Nesse exemplo, o uso da função é o mais trivial, sendo utilizada apenas no retorno da consulta que a invoca. Como resultado da consulta que aplica a função `retornaRua`, tem-se:

```
<resultado>
  <rua>Av. Ipiranga</rua>
  <nao_ha_rua_definida />
  <rua>Alberto Bins</rua>
  <rua>Bento Goncalves</rua>
</resultado>
```

A possibilidade de criação de funções é uma dentre as inovações proporcionadas pela XQuery. Nessa breve abordagem não foram discutidas novidades como os quantificadores, que testam a existência de algum ou de todos elementos em uma condição, nem a capacidade de definição de tipos de dados derivados da XML Schema (aplicação ainda incipiente). Outra funcionalidade não destacada é o suporte a consultas sobre dados relacionais, característica que não é relevante no presente trabalho.

O que pode e deve ser salientado da linguagem é o grande poder de expressão de consultas, por unir os pontos destacados de outras linguagens numa especificação clara

e funcional. A flexibilidade proporcionada para a manipulação dos resultados é algo pouco visto nas demais linguagens, bem como a boa integração com a linguagem XPath. A XQuery ainda recebe complementações e ajustes, entretanto tende a se tornar a principal linguagem de consulta para dados XML.

1.8 Comparativo entre Linguagens

A evolução das pesquisas sobre processamento de consultas parte dos bancos de dados relacionais, passa pelos bancos de dados orientados a objetos e chega aos bancos de dados semi-estruturados e XML. Mesmo com esse longo caminho percorrido, os princípios fundamentais continuaram sempre os mesmos [FER 99]. Destacando a área de dados semi-estruturados e XML, existe uma distinção interna entre as abordagens de estudo. Num lado, está a comunidade de banco de dados que busca tratar grandes repositórios de dados, integração de fontes heterogêneas e padronização de formatos de troca de documentos, no qual se destacam as linguagens Lorel e XML-QL. No outro, está o grupo concentrado no âmbito da programação funcional, no qual as consultas são basicamente definidas a partir de combinações de funções, aonde se encaixa a XQL. E flutuando entre esses grupos estão as linguagens híbridas (XQuery), que aproveitam os conceitos das duas correntes.

Numa primeira análise, nota-se que grande parte dos autores de publicações sobre linguagens de consulta para dados semi-estruturados e XML é originária do campo de banco de dados. Obviamente, essa informação não invalida as pesquisas realizadas pela comunidade da programação funcional. Contudo, constata-se que os conceitos das linguagens de consulta para banco de dados tendem, também, a ser incorporados ao "mundo semi-estruturado".

No decorrer desta seção, são efetivamente comparadas as quatro linguagens, de acordo com alguns aspectos [BON 2000, FER 99, QUA 98] considerados importantes pelos principais pesquisadores dessa área. Também são definidos alguns pontos de comparação pelo autor, os quais são fundamentais para justificar a escolha de uma linguagem como base da extensão proposta na atual pesquisa.

1.8.1 Modelo de dados

Lorel, XML-QL e XQuery introduziram modelos de dados próprios em suas pesquisas, ao contrário da XQL, que segue um "modelo de dados XML implícito". Na Lorel, um elemento XML é um par `<id, valor>`, onde `id` é um identificador único para o elemento e o `valor` pode ser tanto uma string quanto um valor complexo. A representação desse modelo é definida através de um grafo onde os nodos correspondem aos valores dos elementos e as arestas identificam o elemento XML. Na XML-QL; um documento é modelado através de um "grafo XML". Cada vértice é representado por uma string única denominada OID (*Object Identificator*); arestas são rotuladas com identificadores de elementos; nodos são rotulados com pares atributo-valor; folhas representam valores e todo o grafo XML possui um nodo *root* que possui distinção sobre os demais.

Na XQL a representação está baseada simplesmente em documentos ordenados, nos quais diferentes nodos representam entidades, elementos, atributos, etc. O relacionamento entre os dados concentra a maior representação de informação no documento de acordo com a abordagem XQL. Na XQuery, o modelo é um refinamento daquele proposto para a linguagem XPath, onde o documento é modelado como uma

árvore de nodos ordenados. A mais recente versão do modelo integra as necessidades da XPath e da XQuery e disponibiliza suporte a XML Schema e a coleções de documentos.

1.8.2 Seleção

Essa funcionalidade, obviamente, é suportada por todas as linguagens. Seleção de um documento é o resultado da aplicação de uma consulta sobre um documento, resultando na recuperação de elementos de acordo com certas condições. Em Lorel, a estrutura é a seguinte:

```
'SELECT' {expressão} [ 'FROM' {expressão} ] [ 'WHERE' {expressão} ]
```

Já em XML-QL, a estrutura simplificada pode ser assim representada:

```
'WHERE' {predicado} 'CONSTRUCT' { '{´ consulta '}' }
```

Na XQL, a seleção resume-se à navegação através de elementos. No entanto, a consulta é especificada sobre um caminho hierárquico fixo, ou seja, para definir predicados para consultas em XQL, é necessário que estes apareçam inseridos no caminho de localização (expressão de caminho):

```
['./'|'/'|'//'|'.//'] elemento [ '[´ predicado ']' ] [ caminho ]
```

Na XQuery, a seleção é realizada através de expressões FLWR, nas quais expressões de caminho estão inclusas. A BNF bastante reduzida é a seguinte:

```
'FOR' clausulaFor | 'LET' clausulaLet
[ WHERE predicado ]
'RETURN' construção do resultado
```

1.8.3 Joins

Uma junção compara dois ou mais elementos/atributos XML de um ou mais documentos. Em Lorel, junções são totalmente suportadas dentro de um ou entre documentos. As variáveis que participam da junção são explicitamente definidas, assim como ocorre na OQL. A XML-QL trata junções de forma implícita, pois os nomes das variáveis devem ser iguais, assim como os valores que elas representam. Todavia, nada impede que junções sejam definidas sobre vários documentos desde que as variáveis possuam um mesmo nome.

Em XQL são permitidos semi-joins, ou seja, junção de dados que estão em um caminho com dados que estão presentes no mesmo documento. Em sua última especificação é prevista a realização de junções entre diferentes documentos através de variáveis de correlação. A XQuery prevê tanto uso de *inner join* quanto de *outer join*. A partir das variáveis de ligação, a XQuery realiza junções independentemente do número de fontes consideradas.

1.8.4 Resultados

Espera-se que as linguagens possam adaptar os resultados sobre diferentes visões, mas o ideal é o retorno de um documento XML ou seções dele como resposta. As linguagens XML-QL, XQL e XQuery atendem a esse requisito. O mesmo não ocorre com a Lorel. Nela, o resultado é um conjunto de OIDs que identificam os elementos

recuperados. Essa limitação na representação de resultados em um formato XML pode ser destacada como um ponto negativo da linguagem Lorel.

1.8.5 Expressões de caminho

Como foi visto na seção sobre a linguagem Lorel, quando dados semi-estruturados são consultados e não se tem um conhecimento profundo sobre a estrutura de representação, mostra-se ser possível utilizar uma forma navegacional de consulta baseada em expressões de caminho. A forma mais interessante de aplicação de expressões de caminho refere-se à utilização de *wildcards*. Tal característica permite que se atinja um determinado nodo sem que se saiba toda a lista de elementos que compõem o caminho. Essa importante funcionalidade é suportada por todas as linguagens apresentadas neste trabalho.

Em Lorel, as expressões de caminho são flexíveis (descendem da OQL) e permitem grandes variações com o uso de *wildcards*. Porém, qualquer expressão de caminho definida em Lorel deve se basear no elemento raiz do documento. Na linguagem XML-QL, as expressões de caminho são admitidas em expressões regulares. Esse assunto não foi apresentando na seção sobre a XML-QL, mas pode ser conferido em [DEU 98].

Tanto em XQL quanto em XQuery, as expressões são largamente utilizadas. Em XQL, a expressão é a própria consulta, cujo refinamento é feito através de operadores e de funções. Na XQuery, expressões de caminho estão presentes em todas as cláusulas, desde a seleção dos nodos até a definição do formato de apresentação do retorno.

1.8.6 Expressões quantificadas

Um predicado existencial aplicado sobre um conjunto de instâncias é satisfeito se ao menos uma instância atende ao predicado. Em todas linguagens analisadas, os predicados são considerados quantificados existencialmente.

Um predicado universal é somente satisfeito quando todas as instâncias submetidas ao predicado atendem a ele. Essa característica só não está presente na linguagem XML-QL. Na Lorel, uma variável pode ser quantificada universalmente através da aplicação da expressão `FOR ALL`. Em XQL, o mesmo pode ser obtido através da definição do prefixo `$all$` para o predicado. Na XQuery, através da combinação das cláusulas `EVERY`, `IN` e `SATISFIES`, também pode ser verificado se uma variável é universalmente quantificada. Sugere-se consultar a bibliografia das respectivas linguagens para obtenção de maiores detalhes.

1.8.7 Construção e agrupamento de elementos

A construção de elementos consiste no emprego de algum mecanismo de criação aplicado sobre a consulta. Apenas na XQL não é possível a construção de elementos. Na Lorel, um novo elemento é criado a partir do emprego de funções [ABI 97b]. Na XML-QL; a construção de elementos é prevista na cláusula `CONSTRUCT`. Na XQuery, um elemento construtor consiste em uma *tag* inicial e em outra final, que delimitam uma lista de expressões com variáveis cuja função é fornecer o conteúdo do novo elemento.

A capacidade de agregar elementos de um resultado com o emprego de funções especiais, tal como `GROUP BY`, só está presente na linguagem Lorel. Nas demais linguagens, através de combinações de consultas, o mesmo efeito pode ser alcançado (com restrições na XQL).

1.8.8 Agregação e aninhamento de consultas

As funções de agregação permitem computar valores numéricos agrupados. Na Lorel e na XQuery, as funções de agregação são totalmente implementadas. Na primeira proposta da XML-QL não estavam previstas tais funções. Em XQL há suporte apenas à função `count`, que possibilita totalizar o número de referências a um elemento associado a um conjunto.

Com relação ao aninhamento, Lorel, XML-QL e XQuery permitem que consultas sejam aninhadas independentemente do nível desejado; na XQL, não existe suporte.

1.8.9 Ordenação

Consiste em ordenar os elementos do resultado na forma ascendente ou descendente. Em Lorel e XML-QL, emprega-se a cláusula `ORDER BY` para atingir esse fim; já a XQL não apresenta método de ordenação; na XQuery, a cláusula `SORTBY` permite que elementos sejam ordenados. Vale ressaltar que todas as linguagens retornam os resultados seguindo a ordem definida pelo documento original, caso nenhum método de ordenação seja utilizado.

1.8.10 Declaração de funções

A capacidade de permitir a definição de funções é um aspecto de relevante importância para o presente trabalho. A declaração de funções é uma das mais simples maneiras de estender uma linguagem.

Em uma primeira análise, apenas a Lorel não disponibiliza essa capacidade ao usuário. Ela trabalha com uma grande quantidade de funções para o tratamento de expressões de caminho, no entanto suprime essa possibilidade de extensão. A XML-QL permite que o usuário defina suas próprias funções com um porém: necessariamente os parâmetros de entrada devem ser documentos completos; pode ser um ou vários, mas é preciso que sejam documentos inteiros e não parte deles.

Como a XQL pode ser integrada em ambientes de programação, também permite definir funções de acordo com o ambiente em que está inserida. Se a linguagem é implementada em Java, funções definidas em Java também podem ser utilizadas. Além de documentos completos, as funções podem operar sobre conjuntos de nodos ou até mesmo sobre valores atômicos.

Assim como em XQL, a XQuery implanta o conceito de funções extensíveis que atuam sobre documentos, seções ou valores atômicos. A grande vantagem na definição de funções na XQuery concentra-se na linguagem em que elas são definidas. Não é necessário especificar funções numa linguagem de programação à parte; a própria XQuery, com suas construções, permite que funções sejam escritas. Todas as funções podem ser definidas recursivamente, ou seja, podem fazer referência a elas próprias em suas declarações. Uma limitação está na impossibilidade de serem sobrecarregadas. Logo, múltiplas funções com mesmo nome não são suportadas.

1.9 Considerações Finais

Além de apresentar o quadro comparativo (**TABELA 2.5**) que resume as características das linguagens de consulta dentro do âmbito desta pesquisa, alguns aspectos gerais devem ser ressaltados.

TABELA 2.5 – Linguagens de consulta – quadro comparativo

	Lorel	**XML-QL**	**XQL**	**XQuery**
Modelo de dados	Sim modelo OEM	Sim grafo XML	Parcial baseia-se, em parte, ao modelo original XML	Sim modelo XQuery/XPath
Seleção de documentos	Sim `SELECT`, `WHERE`, `FROM`	Sim `WHERE`, `CONSTRUCT`	Sim caminho hierárquico	Sim `FOR`, `LET`, `WHERE`, `RETURN`
Joins	Sim totalmente suportada	Sim implícita, variáveis de mesmo nome	Sim semi-joins e joins entre documentos	Sim inner joins e outer joins
Resultado das consultas	Parcial conjunto de OIDs	Sim documento XML	Sim documento XML	Sim documento XML
Expressões de caminho	Sim definidas a partir do elemento raiz	Sim definida dentro da especificação da tag	Sim suporte completo e avançado	Sim completo, expressões XPath
Quantificadores	Sim existencial e universal	Parcial apenas existencial	Sim existencial e universal	Sim existencial e universal
Construção e agrupamento de resultados	Sim construção através de funções; `GROUP BY`	Parcial não suporta a definição de grupos	Não agrupamento com combinação de operadores e expressões	Parcial não possui função específica para agrupamento
Agregação e aninhamento	Sim `MIN`, `MAX`, `SUM`, `COUNT`, `AVG`; aninhamento SQL-like	Parcial não possui agregação mas disponibiliza aninhamento	Parcial apenas o método `COUNT`; sem aninhamento	Sim todas as funções da SQL; prevê aninhamento em vários níveis
Ordenação	Sim `ORDER BY`	Sim `ORDER BY`	Não	Sim `SORTBY`
Definição de funções	Não	Sim, documentos completos como parâmetros	Sim, funções definidas em linguagens de programação	Sim, funções definidas na própria sintaxe da XQuery

Primeiro aspecto: elas são realmente declarativas? Uma linguagem é declarativa quando o usuário especifica a consulta e não se preocupa em saber a estratégia necessária para computá-la. Desse ponto de vista, todas as linguagens são declarativas. Lorel praticamente reescreve OQL. XML-QL faz uma unificação de alguns conceitos de linguagens de banco de dados com o estilo XML. A XQL faz uso de padrões fundamentados em caminhos de localização, cujas declarações são bastante simplificadas. E XQuery une os conceitos acima destacados em uma especificação de fácil compreensão e elevado poder de expressão.

Dentre as funcionalidades previstas, pode-se considerar a XQuery, seguida pela XML-QL e pela Lorel, como a mais completa. A Lorel destaca-se pelo poder das expressões de caminho, pelo controle de quantificadores, pelo agrupamento de elementos e pela gerência da ordenação. Considerando a sintaxe, inegavelmente, a XML-QL possui uma integração maior com XML, pois a própria escrita da consulta concentra-se em uma estrutura XML. A construção de resultados pode ser considerada um ponto importante na XML-QL. Na XQuery, além do qualificado suporte à construção de elementos, as expressões FLWR fundamentadas em expressões XPath permitem que consultas com graus de complexidade diferentes sejam definidas com clareza e objetividade. Expressões condicionais e funções extensíveis são, também, diferenciais da linguagem. Menos expressiva que as anteriores, a XQL tem sua aplicação mais voltada a consultas simplificadas sobre uma única fonte. Prevê junção para integração de documentos, todavia restrita por uma sintaxe confusa. Assim como a Lorel, a limitação na construção de resultados pesa negativamente na escolha da XQL como embasamento de extensão.

Apesar dessa limitação, a Lorel e a XQL são aquelas que possuem o uso mais simplificado. O usuário familiarizado com a OQL e com a SQL não encontra problemas na operação com a Lorel. A XQL é de fácil utilização excluindo a problemática encontrada na especificação de junções e acrescentando um bom conhecimento da XPath, o qual deve estar presente para um bom domínio da XQuery. O principal problema na XQuery é coordenar o correto uso das cláusulas `FOR` e `LET`; com exceção disso, ela pode ser vista, respeitando o caráter voltado para XML, como uma linguagem SQL-like. Da mesma forma pode ser considerada a XML-QL, que também possui sua estrutura definida em cláusulas. Todavia, a mistura de condições com seleções na cláusula `WHERE` torna uma pouco indistinta a definição de consultas mais complexas.

Após todo o levantamento feito acerca de cada linguagem e o posterior comparativo de suas mais relevantes funcionalidades, foi escolhida a XQuery como base para a extensão proposta. Isso se deve a três destacados motivos:

1. a real possibilidade de que ela se torne padrão pelo W3C para consultas a dados XML. As demais são propostas que foram submetidas ao W3C mas não apresentaram grande desenvolvimento nesse último ano;
2. compreensão do funcionamento das expressões FLWR, destacando a grande flexibilidade na construção de resultados, algo fundamental na composição de saídas qualificadas idealizadas na extensão;
3. funções extensíveis e expressões condicionais que dispensam a utilização de uma linguagem à parte para a definição de funções que são o núcleo da extensão proposta.

3 Modelo Temporal de Versões

O número de aplicações que possuem e/ou controlam aspectos temporais é bastante considerável. Todavia, nem todas são modeladas e desenvolvidas adequadamente para que representem com fidelidade informações temporais. Diversos modelos temporais foram propostos visando tornar mais eficiente o projeto de aplicações para os quais o armazenamento do histórico dos dados é relevante. Esses modelos vão desde extensões de modelos relacionais [GAD 88, NAV 89, TAN 86], passando por extensões de modelos entidade-relacionamento [ANT 97, LOU 91], chegando a extensões de modelos orientados a objetos [EDE 94, KAF 92]. Sinteticamente, destacam-se as seguintes características genéricas de modelos temporais: associação de informações referentes ao tempo de transação e/ou validade aos dados; uso de pontos no tempo, intervalos de tempo ou elemento temporal como primitiva de tempo, e uso de objetos ou entidades temporais, juntamente com não temporais no mesmo diagrama [MOR 2001].

Em algumas aplicações, controlar o processo de evolução de entidades em diferentes instantes ou de acordo com certo paradigma apresenta-se indispensável. Na modelagem desse tipo de aplicação, faz-se o uso do conceito de versão. A maioria dos estudos nessa área concentra-se na incorporação do conceito de versão a modelos orientados a objetos [AGR 91, GOL 95]. Nos casos em que alternativas de objetos são necessárias, prima-se pelo uso de modelos de versões. Entretanto, mesmo que versões de entidades sejam geridas, nem todo histórico de modificações dos dados dessas entidades é considerado. Logo, informações importantes podem ser alteradas sem que haja um controle de sua existência, sem considerar que qualquer modificação no objeto não terá atrelado a si o tempo em que ocorreu.

Na busca de solucionar esses e outros problemas de projeto, foi definido o Modelo Temporal de Versões orientado a objetos. Nele, os aspectos de tempo e de versão são conjugados permitindo que objetos e suas possíveis versões sejam armazenados, juntamente com o histórico dos seus atributos e dos seus relacionamentos. Assim, de acordo com a necessidade, o usuário pode definir aqueles elementos cujo controle de sua vida seja importante e, em caso de alteração, recuperá-los quando necessário for, mediante a especificação do tempo em que eles estavam ativos. Destaca-se que as classes, definidas com base nesse modelo, não precisam ser obrigatoriamente temporais versionadas. Caso o usuário deseje especificar classes em que o histórico de evolução de suas instâncias não seja importante, ele tem liberdade para definir classes convencionais e, por conseguinte, instanciar objetos sem características de tempo e de versão.

Neste capítulo são apresentadas algumas características que permitem um bom entendimento do Modelo Temporal de Versões. Alguns conceitos são expostos de forma superficial, no entanto todas as funcionalidades, com riqueza de detalhes, são encontradas no trabalho definido por Moro [MOR 2001].

1.10 Representação Temporal

No Modelo Temporal de Versões, objetos, versões, relacionamentos e atributos têm associação direta com o tempo. A representação do tempo no modelo é feita através de rótulos de tempo utilizando o conceito de bitemporalidade. Isso indica que tanto o tempo de transação quanto o tempo de validade da informação são considerados. O uso desse conceito prevê uma necessidade de gerência mais especializada. Contudo, os aspectos temporais são representados de maneira mais fiel, visto que o tempo em que a informação foi criada e o tempo em que ela vale são distintamente controlados.

Para realizar esse controle no nível dos objetos e de versões é predefinido, no modelo, o atributo *alive*. No momento da criação de um objeto ou versão, esse atributo lógico recebe o estado *true*. Como se trata, também, de um atributo temporal (possui rótulos representando o tempo de transação e de validade), seus tempos de transação e de validade inicial são armazenados. No instante em que um objeto ou versão for excluído logicamente[1], o atributo *alive* recebe o valor *false* e seu tempo de validade final adquire o tempo de transação em que a operação ocorreu. Assim, mantém-se a integridade dos intervalos de transação e de validade e, por decorrência, do histórico de vida dos objetos e/ou versões. Ressalta-se que cada versão possui uma linha de tempo distinta que deve estar contida na linha de tempo do objeto. Para a versão, o tempo é linear; já, para o objeto que possui versões, logo, linhas de tempo diferentes, o tempo é ramificado.

. Atributos e relacionamentos de objetos e/ou versões temporalizados não precisam ser obrigatoriamente temporais. Com base na necessidade do usuário podem ser definidos tanto atributos e relacionamentos estáticos quanto temporalizados. Por exemplo, no momento da especificação de uma classe, o usuário tem total liberdade de escolher quais atributos serão temporalizados e quais serão estáticos, e vice-versa. Sendo estáticos, a variação dos valores dos atributos e dos relacionamentos não é armazenada; sendo temporalizados, atributos e relacionamentos têm associados a si os rótulos predefinidos *tTimei*, *tTimef*, *vTimei* e *vTimef*, que representam, respectivamente, os tempos de transação inicial e final, e os tempos de validade inicial e final.

1.10.1 Integridade temporal

Como foi visto anteriormente, objetos, versões, atributos e relacionamentos têm associados rótulos de tempo. Considerando objetos e versões, os rótulos de tempo das versões devem estar contidos no rótulo temporal do objeto. Levando-se em conta relacionamentos e atributos, seus rótulos devem estar presentes na linha de tempo da versão a que pertencem. Essas considerações fazem parte do conjunto de regras de integridade para objetos [MOR 2001], mostrados a seguir:

- tempo de validade inicial de um atributo ou relacionamento temporal deve ser maior ou igual ao tempo de vida inicial da versão à qual pertence;
- tempo de transação inicial de um atributo ou relacionamento temporalizado deve ser maior ou igual ao tempo de vida inicial da versão à qual pertence;

[1] Por definição, dados com características temporais do TVM não são excluídos fisicamente. Fica a cargo do administrador a realização de tal operação.

- tempo de vida final de uma versão deve ser maior que o inicial e maior ou igual ao maior tempo de transação e validade final de seus atributos e relacionamentos temporalizados;
- tempo de vida inicial de uma versão deve ser menor que o final e maior ou igual ao tempo de vida inicial do objeto versionado;
- tempo de vida final de um objeto versionado deve ser maior que o inicial e maior ou igual ao maior tempo de vida final de suas versões.

Para que essas regras sejam viabilizadas, mostra-se indispensável a manutenção da integridade dos rótulos temporais. Visando a essa manutenção são consideradas as regras de inserção, atualização e exclusão para bancos de dados bitemporais definidas por Hübler [HUB 2000]. Resumindo-as:

- na inserção, qualquer informação inserida deverá receber os tempos de transação e os tempos de validade. Os primeiros são obtidos através do SGBD; já os últimos são fornecidos pelo usuário. Ainda, o usuário *deve* informar o tempo de validade inicial e *pode* definir o tempo de validade final. Caso não o informe, ele será igual a *null* (válido até que outra informação seja definida). O SGBD fornece o tempo de transação inicial e informa o tempo de transação final no momento em que ocorre uma atualização na base de dados;
- na atualização, para manter o histórico dos dados, exige-se que as informações não sejam fisicamente modificadas. Dessa forma, a alteração consistirá em novas inserções, que deverão manter a semântica da informação através dos intervalos de transação e de validade. Destaca-se que dependendo da atualização, pode ser necessário um rearranjo considerável dos rótulos temporais;
- na exclusão, os objetos, as versões, os atributos e os relacionamentos não são excluídos fisicamente; apenas seus tempos de validade final são encerrados. No que se refere às versões, o atributo que modela o estado da versão, *status*, recebe o valor *deactivated* e atributo *alive* recebe o valor *false*. Essas atualizações indicam que a versão não está mais ativa, sendo utilizada, a partir de então, apenas em consultas. Logo, se um objeto tiver sua vida encerrada, conseqüentemente suas versões, seus atributos e seus relacionamentos também deverão ser encerrados.

1.10.2 Tipos temporais

No modelo são definidos alguns tipos estruturados (classes) (**FIGURA 3.1**) que são empregados para modelar os rótulos temporais para atributos e relacionamentos. Dentre estes tipos, a classe base é a *TemporalLabel*. Nela são armazenados os tempos de transação inicial e final, e de validade inicial e final. Essa classe é usada como tipo complexo do atributo *tempLabel* nas classes *InstantAttribute* e *InstantRelationship*. A primeira classe é utilizada para armazenar o valor do atributo com seu rótulo temporal; a última armazena os identificadores dos objetos e/ou versões que fazem parte de um relacionamento, juntamente com seu rótulo temporal. Os atributos temporais (*InstantAttribute*) e os relacionamentos temporais (*InstantRelationship*) são componentes, respectivamente, das classes *TemporalAttribute* e *TemporalRelationship*. Além dos relacionamentos de agregação, esses dois pares de classes possuem os

relacionamentos de associação *CurrentAttr* e *CurrentRel,* que permitem a recuperação dos valores correntes para atributos e relacionamentos, concomitantemente.

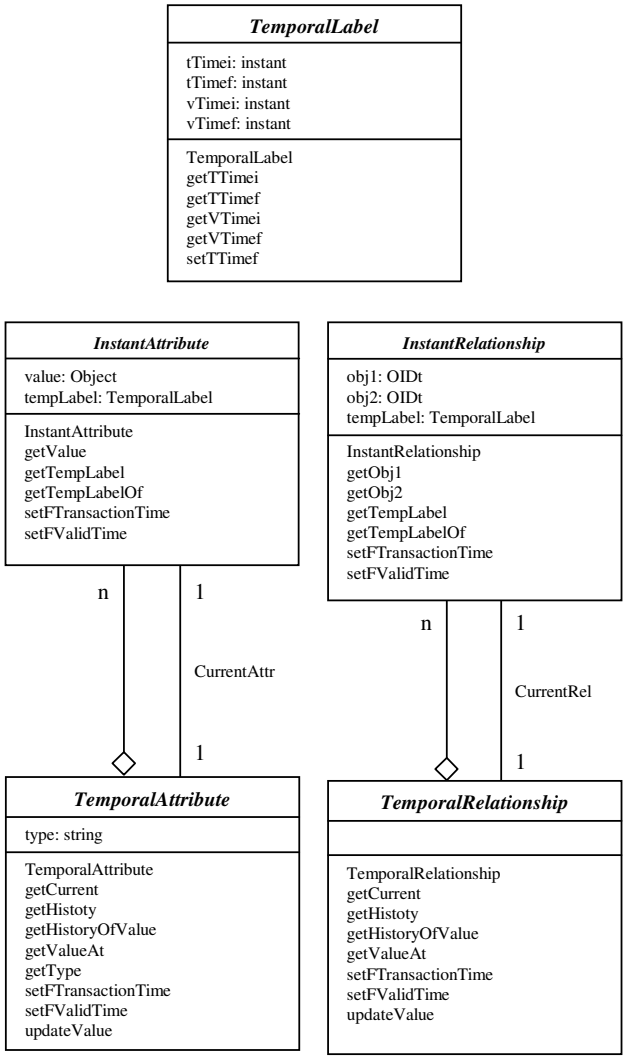

FIGURA 3.1 – Atributos e relacionamentos temporais

1.11 Representação de versões

Um objeto que apresenta versões é considerado um *objeto versionado*. Uma instância não versionada passa a ser versionada dinamicamente no momento em que houver uma derivação do objeto não versionado. Este passará a ser a primeira versão do objeto versionado recém-criado e a versão resultante da derivação tornar-se-á a *versão corrente*. No entanto, o usuário pode definir a versão corrente de acordo com sua necessidade.

As características acima destacadas configuram a criação de um objeto versionado de forma *implícita*. Todavia, um objeto versionado pode ser definido sem que haja versões associadas a ele. Nesse caso, uma operação especial deve ser empregada, configurando um caso de criação *explícita*.

As versões, independentemente da forma de criação do seu objeto versionado, podem assumir diferentes estados durante suas vidas. Os estados podem ser vistos abaixo e suas possíveis transições são descritas no diagrama de estados (**FIGURA 3.2**).

- *Working*: a versão automaticamente assume esse estado no momento em que é criada. Durante essa fase, a versão pode sofrer alterações até que alcance uma situação estável. Uma versão *Working* pode ser derivada, promovida para o estado *Stable*, consultada e excluída;
- *Stable*: neste estágio, a versão encontra-se num estado mais consistente, o que permite que seja compartilhada, porém não alterada. A versão atinge esse estado porque uma versão foi criada como sua derivada ou porque o usuário realizou uma promoção. Pode ser promovida para o estado *Consolidated*, derivada, consultada e excluída (desde que não possua versões sucessoras);
- *Consolidated*: uma versão atinge esse estado mediante ação específica do usuário (promoção do estado *Stable* para *Consolidated*). Não pode ser alterada nem excluída. Pode ser derivada, consultada e compartilhada por outros usuários;
- *Deactivated*: trata-se de uma versão que foi excluída. Neste estado, a versão apenas pode ser consultada.

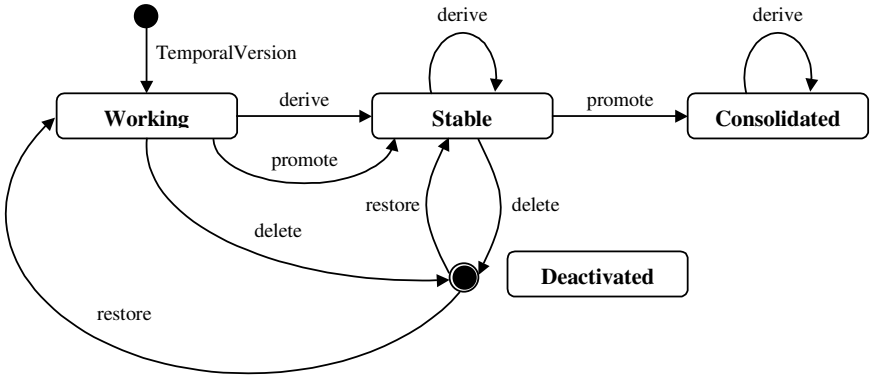

FIGURA 3.2 – Diagrama de estados

1.11.1 Hierarquia de classes

O conjunto de classes do Modelo Temporal de Versões possui como base a hierarquia proposta por Golendziner [GOL 95]. No entanto, para atender aos aspectos temporais adicionados ao modelo, foram propostas alterações nas classes existentes e adições de novas classes à hierarquia. A nova estrutura de classes é apresentada na **FIGURA 3.3**.

A partir dessa hierarquia, permite-se ao usuário especificar dois tipos de classes de aplicação: classes não temporais e não versionavéis e classes temporais versionadas. Ressalta-se que o usuário ou define classes convencionais ou especifica classes com características de tempo e de versão. Dessa forma, fica impossibilitada a criação de classes apenas temporais ou unicamente versionadas.

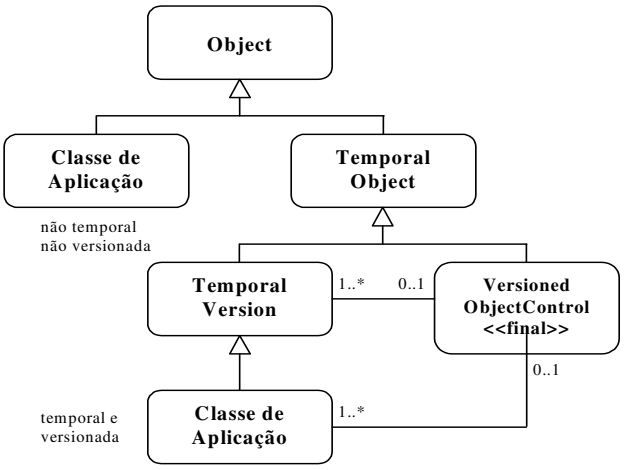

FIGURA 3.3 – Hierarquia de classes do Modelo Temporal de Versões

Considerando as novas classes, *TemporalObject* e *VersionedObjectControl* não são acessíveis para o usuário, sendo controladas apenas pelo sistema gerenciador. A última é uma *classe final*, logo não pode ser especializada. Já *TemporalObject* e *TemporalVersion* são *classes abstratas*, portanto não podem ser diretamente instanciadas. A seguir é apresentada a estrutura das classes, destacando, principalmente, os atributos que as compõem:

- *Object*: a classe *Object* é a raiz da hierarquia. Nela é especificado o atributo *tvOID*, que é utilizado pelo modelo para identificar cada instância de objeto. A estrutura do tvOID é composta pelo identificador da entidade, o identificador da classe e o número da versão. Por exemplo: tendo uma entidade com identificador *1*, uma classe identificada pelo algarismo *5* e o número da versão igual a *3*, o tvOID seria assim representado: *1,5,3*. A estrutura não difere para objetos não temporais e não versionados. A única alteração ocorre na representação do número de versões, que será sempre *null*.

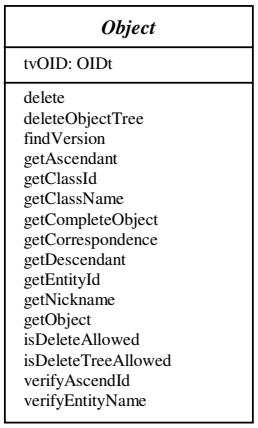

FIGURA 3.4 – Classe *Object*

- *TemporalObject*: essa classe representa os aspectos temporais do modelo. Ela possui a propriedade temporal *alive*, que indica se o objeto está ativo (válido) ou não. Como um objeto pode ser restaurado depois

de ter sido desativado, modelou-se o atributo como sendo temporal. Assim, o histórico das "vidas" do objeto pode ser controlado. Caso haja necessidade de recuperar o tempo de vida inicial e final do objeto, basta acessar os tempos de validade inicial e final do atributo *alive*.

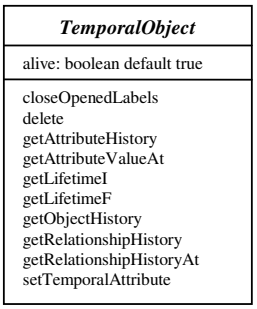

FIGURA 3.5 – Classe *TemporalObject*

- *VersionedObjectControl*: Os atributos *currentVersion*, *firstVersion*, *lastVersion*, *configurationCount*, *userCurrentFlag*, *versionCount* e *nextVersionNumber* permitem armazenar informações de controle das versões de objetos versionados. Excetuando o último atributo, todos são temporais. Com essas propriedades, é possível determinar qual versão é a corrente num determinado tempo, a primeira versão na hierarquia de derivação, a última versão, o número de configurações existentes, se a versão corrente foi assim definida pelo usuário, o número de versões ativas e o próximo número a ser utilizado para composição do tvOID de uma nova versão. Uma instância da classe *VersionedObjectControl* será criada a partir de uma derivação. Nesse momento, um objeto passa a ser versionado e o controle das suas versões passa também a ser pertinente.

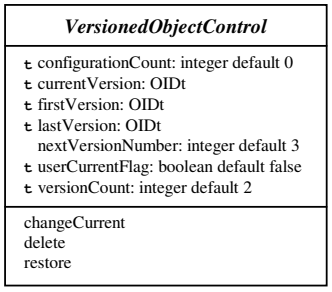

FIGURA 3.6 – Classe *VersionedObjectControl*

- *TemporalVersion*: as classes de aplicação herdam atributos e operações diretamente da classe *TemporalVersion*. Nela são definidos os atributos *ascendant, descendant, status, successor, predecessor* e *configuration*. Dentre os atributos, os dois últimos são não temporais e os demais são temporais. O atributo *status* armazena o estado da versão (*Working*, *Stable*, *Consolidated* ou *Deactivated*) e a propriedade *configuration* indica se a versão faz ou não parte de uma configuração [GOL 95]. *Predecessor* e *successor* controlam aspectos referentes à hierarquia de derivação. O conjunto de tvOID das versões que servira de base para a derivação de uma versão *V* é armazenado em *predecessor*. Já os tvOID das versões derivadas a partir da versão *V* estão contidas em *successor*.

Com uma semântica semelhante, porém para a hierarquia de extensão [GOL 95, MOR 2001], são definidos os atributos *ascendant* e *descendant*. Em *ascendant* estão presentes os tvOID das versões da classe superior na hierarquia que possuem um relacionamento de extensão com a versão. Em *descendant* o comportamento é o mesmo, no entanto consideram-se as versões da classe inferior na hierarquia.

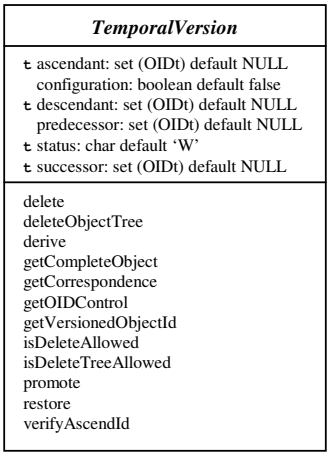

FIGURA 3.7 – Classe TemporalVersion

1.11.2 Funcionamento da hierarquia

Esta seção apresenta, através de um exemplo básico, o comportamento da hierarquia de classes do Modelo Temporal de Versões. No exemplo, deseja-se modelar uma classe de aplicação temporal versionada, instanciar um objeto da classe e derivar uma versão do objeto.

A classe a ser modelada é a *Computador*. Mostra-se importante armazenar informações referentes ao processador e ao valor do computador. Também se considera fundamental ter controle das diferentes versões de equipamentos que já foram criadas. Para cada uma dessas alternativas, deve-se manter o histórico do valor da máquina. Com base nessas informações, sabe-se que deverá ser criada uma classe temporal versionada para manter as diferentes versões de equipamentos e, também, que o atributo que representará o valor do computador deverá ser temporal:

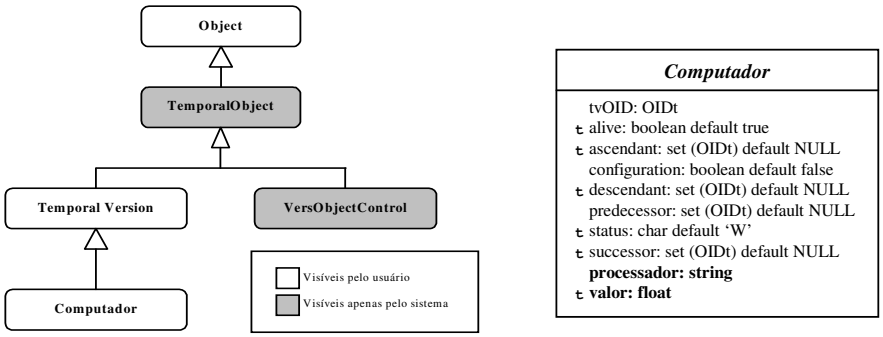

FIGURA 3.8 – Hierarquia de classes e especificação da classe *Computador*

Por se tratar de uma classe temporal versionada, a classe *Computador* deve ser definida como subclasse de *TemporalVersion*. A **FIGURA 3.8** apresenta a hierarquia

desde a classe *Object* até a classe de aplicação *Computador*. Na figura estão realçadas as classes que não são vistas pelo usuário, as quais são exclusivamente controladas pelo sistema, cabendo ao usuário interagir com as classes *Object*, *TemporalVersion* e classes de aplicação.

Ainda na **FIGURA 3.8** é apresentada a especificação da classe *Computador*. Para o usuário, basta adicionar os atributos necessários, sejam eles temporais ou não temporais, e definir a classe como herdeira de *TemporalVersion*. Na sua visão, a classe apenas será composta pelos atributos da aplicação (destacados em negrito na figura). No entanto, para o sistema, a classe é composta por esses e pelos demais atributos de controle que são herdados de *Object*, *TemporalObject* e *TemporalVersion*.

Com a especificação concluída, instâncias já podem ser criadas. Neste exemplo, está sendo considerada a criação de um objeto *ComputadorA* (**FIGURA 3.9**), sem versões. O objeto terá associado a si um tvOID 10,5,1: 10, para o identificador da entidade; 5, para o identificador da classe *Computador*, e 1, por se tratar da primeira versão (para fins de controle, o objeto sem versões é considerado como a versão um). Nesse momento, as propriedades *ascendant*, *descendant*, *predecessor* e *successor* possuem valor *null*, pois não existem correspondências com outras versões. Os atributos *alive* e *status* recebem *true* e *W* (*Working*), respectivamente. Como o objeto não faz parte de uma configuração, seu atributo *Configuration* detém o valor *false*.

ComputadorA	
tvOID	10,5,1
alive	true
ascendant:	null
configuration	false
descendant	null
predecessor	null
status	W
successor	null
processador	**850 Mhz**
valor	**5.600,00**

FIGURA 3.9 – Objeto sem versões – *ComputadorA*

No momento em que o objeto sofre uma derivação, ele evolui para o *status* de objeto versionado. Ele passa a ser a primeira versão (10,5,1) e a sua derivada torna-se a segunda versão (10,5,2). O objeto versionado *Computador* recebe o tvOID 10,5,0 e é associado a uma instância de *VersionedObjectControl* responsável pelo controle do objeto versionado recém-criado.

ComputadorA	
tvOID	10,5,1
alive	true
ascendant:	null
configuration	false
descendant	null
predecessor	null
status	S
successor	10,5,2
processador	**850 Mhz**
valor	**5.600,00**

ComputadorB	
tvOID	10,5,2
alive	true
ascendant:	null
configuration	false
descendant	null
predecessor	10,5,1
status	W
successor	null
processador	**1 Ghz**
valor	**7.500,00**

FIGURA 3.10 – Versões um e dois do objeto versionado *Computador*

A versão um passa para o status *Stable* e seu atributo *sucessor* recebe o valor do tvOID da nova versão, 10,5,2. A versão dois, logo após a derivação, possui os valores de atributos idênticos aos da versão um, com exceção da propriedade *predecessor*, que recebe o tvOID da versão da qual deriva, e do atributo *status*, que passa a ser *Working*.

Na **FIGURA 3.10** são apresentados os valores dos atributos para as duas versões. Vê-se que o atributo *processador* já possui um valor diferente daquele presente na versão um. Isso é reflexo da realização de uma operação sobre a nova versão, o que não implica qualquer modificação na versão base da derivação.

Quando o usuário faz uma referência ao objeto versionado, os atributos que ele está considerando são aqueles pertencentes à versão corrente, neste caso, a versão dois. O *status* do objeto versionado é sempre *Working* porque qualquer modificação nas versões reflete diretamente em seus atributos. O conceito de objeto versionado é apresentado na hierarquia pela união da instância de *TemporalVersion* do objeto versionado com a instância de *VersionedObjectControl* [MOR 2001]. A representação desse conceito é mostrada na **FIGURA 3.11**. Na instância de *VersionedObjectControl* (*VersObjControlComputador*), vale destacar a composição do tvOID. O identificador da entidade é o mesmo que do objeto versionado e suas versões. Como o objeto de controle não possui versões, o identificador do número de versões recebe *null*. A novidade aparece no identificador da classe. Como se trata de uma instância de uma classe diferente (não é instância de *Computador* e, sim, de *VersionedObjectControl*), o identificador da classe também possui um valor diferenciado.

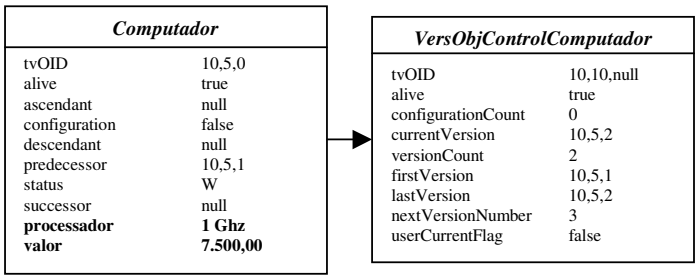

FIGURA 3.11 – Objeto versionado e instância de *VersionedObjectControl*

1.12 Considerações Finais

Neste capítulo, foi brevemente discutido o Modelo Temporal de Versões. Aspectos referentes à representação temporal e de versão foram enfocados, realçando regras de integridade temporal e tipos temporais, bem como a hierarquia de classes do modelo. No que tange às classes, houve um maior destaque à apresentação e explicação dos atributos do que das operações. Estas não são consideradas no desenvolvimento do trabalho, mas deverão fazer parte de futuras extensões do mesmo.

Ainda, foi apresentado o comportamento da hierarquia de classes através de um exemplo, porém não foi debatida a questão da existência de versões em vários níveis Essa situação configura o uso da herança por extensão, que permite um objeto ser desenvolvido em nível de abstração e detalhado nos diferentes níveis da hierarquia. Assim como a herança por extensão, o uso de configurações não foi detalhado neste capítulo. Sugere-se, para esses dois tópicos, uma conferência mais ampla do Modelo Temporal de Versões proposto por Moro.

4 Linguagem de Consulta do TVM

Com o objetivo de consultar dados de aplicações modeladas a partir do Modelo Temporal de Versões, foi definida a linguagem de consulta Temporal Version Query Language – TVQL [MOR01a]. Essa linguagem é baseada em SQL; logo, permite representar consultas através de cláusulas SELECT-FROM-WHERE, além de possibilitar o uso das funções básicas disponíveis em SQL. No entanto, para recuperar informações temporais e de versão, a TVQL define novas funcionalidades que atendem, em boa parte, a essa necessidade

A concepção orientada à SQL permite ao usuário expressar consultas TVQL com aspectos temporais e de versão de uma forma muito similar à utilizada em consultas convencionais. Isso é possível através do emprego de funções cujas declarações se assemelham às comumente vistas em SQL. Porém, os objetos sobre os quais essas novas funções são aplicadas são especificados com uma semântica diferenciada. Em SQL, os objetos são tabelas; já, em TVQL, não são tabelas, mas classes e versões de objetos instanciados dessas classes. Tal situação se justifica porque se trata de uma linguagem de consulta para um modelo orientado a objetos.

Neste capítulo, através de vários exemplos, são apresentadas e discutidas as características de tempo e de versão propostas pela TVQL. Ao longo das seções, alguns quadros mostram as possibilidades de combinação das propriedades e funções. Ao final, expõem-se algumas considerações acerca das novidades e das restrições.

1.13 Características Gerais

A linguagem de consulta do Modelo Temporal de Versões, TVQL, mantém os aspectos principais da SQL. As cláusulas básicas da TVQL seguem a mesma estrutura da linguagem para bancos de dados relacionais:

```
  SELECT <colunas e/ou funções>
  FROM   <tabelas>
[ WHERE  <condições> ]
```

A eliminação de duplicatas também é prevista nessa linguagem através do uso da palavra `DISTINCT`. Operadores lógicos, relacionais, de conjuntos (`UNION`, `INTERSECTION`, `DIFFERENCE`), de condições compostas (`IN`, `BETWEEN`) e de combinação de padrões (`LIKE`) possuem suporte. Além dos operadores, funções de agregação (`COUNT`, `SUM`, `AVG`, `MIN`, `MAX`) e tratamento de grupos de dados (`GROUP BY`, `HAVING`, `ORDER BY`) são igualmente disponibilizados.

1.14 Características Temporais

As linguagens convencionais de consulta não consideram o histórico temporal dos dados, ou seja, o que se está consultando é a visão atual e única dos dados sendo desprezado qualquer aspecto referente ao histórico das informações. Já a TVQL, como linguagem temporal, permite que todo histórico de dados seja considerado ou apenas os valores atuais.

Se uma consulta expressa em TVQL for definida no estilo apresentado abaixo, ela irá se basear única e exclusivamente nos valores atuais dos dados:

```
SELECT processador
FROM Computador
WHERE valor > 3000
```

Nota-se que a estrutura da consulta é idêntica à da SQL, o que foi idealizado na TVQL visando manter a maior integridade possível com a linguagem base. Nesse caso, a consulta pretende recuperar os valores **atuais** do atributo `processador` dos objetos instanciados a partir da classe `Computador` cujo `valor` **atual** seja superior a 3000. Para casos em que é necessário considerar o histórico, foi projetada a instrução `EVER`.

A palavra reservada `EVER` possibilita que todos os valores do histórico da vida de objetos e versões sejam analisados. O emprego dessa instrução pode se dar de duas maneiras distintas dentro da consulta:

- **Após a cláusula `SELECT`:** no momento em que for definida a palavra `EVER` logo após a cláusula `SELECT`, será retornado o histórico dos atributos temporais (`SELECT`) e considerado o histórico dos valores analisados na condição (`WHERE`):

  ```
  SELECT EVER processador, valor
  FROM Computador
  WHERE processador = “700”
  ```

  A consulta acima retornará o **histórico** dos atributos temporais `processador` e `valor` das instâncias de `Computador` em que a propriedade `processador`, **em algum momento**, teve o valor igual a 700 Mhz.

- **Após a cláusula `WHERE`:** utiliza-se essa faceta quando se deseja que apenas a cláusula `WHERE` seja temporal. Dessa forma, é possível selecionar os valores atuais dos atributos na cláusula `SELECT` e considerar o histórico na cláusula `WHERE`:

  ```
  SELECT processador, valor
  FROM Computador
  WHERE EVER processador = “700”
  ```

  Neste exemplo, serão recuperados os valores **atuais** das propriedades temporais `processador` e `valor` desde que o atributo `processador` já tenha assumido o valor 700 Mhz **em alguma situação**.

Em algumas circunstâncias, pode ser necessário selecionar os valores atuais de um atributo e o histórico de uma outra propriedade num mesmo momento. Para satisfazer a essa necessidade, além do uso da instrução `EVER`, é preciso aplicar uma função adicional. Essa função, denominada `PRESENT`, anula a condição que foi predefinida por `EVER`. Assim, passa-se a considerar apenas os valores atuais para o atributo ao qual é aplicada:

```
SELECT EVER processador, PRESENT(valor)
FROM Computador
WHERE processador = “700”
```

Levando em consideração computadores que **em algum momento** possuíram o processador 700 Mhz, a consulta selecionará o **histórico** dos processadores e apenas o **valor corrente** do atributo `valor`. Nessa mesma consulta, se a função `PRESENT` fosse aplicada ao predicado, apenas seriam válidos os objetos cujo atual valor do atributo `processador` fosse igual a 700 Mhz.

A mescla de histórico com valor atual pode ser aplicada, também, na cláusula `WHERE`. Supõe-se na próxima consulta que há necessidade de recuperar o valor atual para a propriedade `processador`. Mas, como condição, deseja-se que o valor do equipamento seja 4500 e que o processador, em alguma ocasião, tenha assumido o valor 500 Mhz:

```
SELECT processador
FROM Computador
WHERE EVER processador = "500" AND PRESENT(valor = 4500)
```

No entanto, se em uma situação for necessário recuperar o histórico de uma única propriedade sendo que todas as demais, inclusive as condições, valem-se apenas do estado atual, será obrigatório utilizar a função `PRESENT` nas demais propriedades e em todas as condições. Isso ocorre porque não há função análoga a `PRESENT` para considerar todo o histórico de um elemento de forma individualizada:

```
SELECT EVER atributo1, PRESENT(atributo2), PRESENT(atributo3), ...
FROM Classe
WHERE PRESENT(condição1) AND PRESENT(condição2) AND ...
```

TABELA 4.1 – Variantes de utilização das funções EVER e PRESENT

Possibilidades de utilização		**EVER**		**PRESENT**	
Seleção	**Condição**	**SELECT**	**FROM**	**SELECT**	**FROM**
atual	atual				
atual	com histórico		√		
atual	atual e com histórico		√		√
com histórico	atual	√			√
com histórico	com histórico	√			
com histórico	atual e com histórico	√			√
atual e com histórico	atual	√		√	√
atual e com histórico	com histórico	√		√	
atual e com histórico	atual e com histórico	√		√	√

Independentemente das possibilidades de utilização das funções `EVER` e `PRESENT` (**TABELA 4.1**), a linguagem propõe várias propriedades para tratamento de instantes e intervalos temporais.

Para recuperar instantes de transação e validade de atributos e relacionamentos temporais, são definidas quatro propriedades:

- **`tiInstant`**: recupera o tempo de transação inicial de um atributo ou relacionamento temporal;

- **`tfInstant`**: recupera o tempo de transação final de um atributo ou relacionamento temporal;
- **`viInstant`**: recupera o tempo de validade inicial de um atributo ou relacionamento temporal;
- **`vfInstant`**: recupera o tempo de validade final de um atributo ou relacionamento temporal.

Para exemplificar o uso dessas propriedades, consideram-se as seguintes consultas:

```
SELECT processador.tiInstant, processador.tfInstant
FROM Computador
WHERE EVER processador = "700"

SELECT valor
FROM Computador
WHERE processador.viInstant > "10-10-2001"
```

Como se pode ver, as propriedades aplicam-se tanto na cláusula `SELECT` quanto na cláusula `WHERE`. Na primeira consulta, objetiva-se selecionar o tempo de transação inicial e final do atributo `processador`, desde que esse possua em seu histórico o valor correspondente a 700 Mhz. O retorno dessa consulta seria os **instantes inicial e final atuais** do atributo `processador`, e não o seu **valor atual**.

Já na segunda consulta, é utilizada a propriedade predefinida `viInstant` na especificação de uma condição. Além desta, qualquer uma das propriedades pode ser utilizada para definição de condições. Em conjunto com operadores relacionais, elas podem compor comparações temporais com:

- **a palavra reservada `NOW`:** o atributo ou relacionamento temporal conjugado com uma propriedade predefinida para recuperação de instante pode ser comparado com o instante atual definido por `NOW`:

  ```
  valor.tfInstant <= NOW
  ```

- **um instante:** o atributo ou relacionamento temporal conjugado com uma propriedade predefinida para recuperação de instante pode ser comparado com um instante qualquer:

  ```
  processador.vfInstant <> "05-11-2001"
  ```

- **uma propriedade:** o atributo ou relacionamento temporal conjugado com uma propriedade predefinida para recuperação de instante pode ser comparado com uma propriedade definida pelo usuário, desde que ela seja do mesmo tipo que o instante recuperado pela propriedade prédefinida:

  ```
  valor.tiInstant > dataFabricacao
  ```

As alternativas de comparações temporais demonstradas acima podem ser integralmente utilizadas com as propriedades estabelecidas para a recuperação dos tempos de vida inicial e final dos objetos e versões:

- **`iLifetime`:** recupera o tempo de vida inicial de um objeto e/ou versão;
- **`fLifetime`:** recupera o tempo de vida final de um objeto e/ou versão.

Essas duas propriedades foram definidas para facilitar a obtenção dos tempos de vida de objetos e/ou versões. Na realidade, os tempos de vida são expressos pelo atributo *alive*. Como este atributo é temporal, bastaria utilizar as propriedades predefinidas de recuperação de instantes para obter o tempo de validade inicial e o tempo de validade final do objeto e/ou versão. Para ilustrar o uso das duas propriedades, considera-se a seguinte consulta:

```
SELECT iLifetime
FROM Computador
WHERE fLifetime > Now
```

A consulta simplesmente obtém o tempo de vida inicial dos objetos instanciados a partir da classe `Computador`, desde que o tempo de vida final dos mesmos objetos seja superior ao instante atual, definido por `NOW`.

Além das propriedades definidas para instantes e para tempos de vida de objetos e/ou versões, a TVQL também disponibiliza propriedades para tratamento de intervalos temporais. A representação temporal na forma de intervalos é mais significativa que a simples utilização de instantes, isso porque se tem a noção de início e fim, o que não é potencialmente expresso através de um único instante (a não ser que um mesmo instante represente o início e o fim da validade de uma informação – intervalo degenerado [DIV 92]).

No modelo, atributos e relacionamentos temporais apresentam dois tipos de intervalos: um intervalo de transação (delimitado pelo tempo de transação inicial e pelo tempo de transação final) e um intervalo de validade (delimitado pelo tempo de validade inicial e pelo tempo de validade final). De acordo com os instantes delimitadores, os intervalos podem ser diferentemente representados:

- **com conhecimento dos instantes inicial e final:** quando existe o conhecimento do instante inicial e também do instante final, o intervalo temporal é expresso da seguinte forma:

  ```
  [instanteInicial..instanteFinal]
  ```

- **com conhecimento do instante inicial:** se o instante inicial está definido e o instante final não (representando infinito futuro), a representação do intervalo se faz da forma:

  ```
  [instanteInicial..]
  ```

- **com conhecimento do instante final:** quando o instante final é conhecido e o instante inicial é nulo (infinito passado), o intervalo é expresso:

  ```
  [..instanteFinal]
  ```

Independentemente do intervalo, os delimitadores "[" e "]" e o símbolo ".." estão presentes em todas as formas de representação. A diferenciação ocorre no momento em que um dos instantes não está especificado. Se ambos são conhecidos, o

símbolo *".."* apenas denota que o intervalo possui como extremidades os dois instantes. Caso o instante inicial não esteja definido, o símbolo *".."* passa a representar os instantes que são anteriores ao instante final do intervalo. A mesma simbologia é utilizada quando não se tem o conhecimento do instante final. Porém, nesta situação, o que estão sendo representados são os instantes posteriores ao tempo inicial do intervalo.

Com o formato de representação de intervalos consolidado, a TVQL define propriedades específicas para recuperar intervalos temporais. São elas:

- **`tInterval`**: recupera o intervalo de tempo de transação de um atributo ou relacionamento temporal;
- **`vInterval`**: recupera o intervalo de tempo de validade de um atributo ou relacionamento temporal.

Da mesma maneira que as propriedades predefinidas para instantes, as propriedades para recuperação de intervalos podem ser utilizadas tanto na cláusula `SELECT` quanto na cláusula `WHERE`. Nesta última, são aplicados novos operadores para compor condições temporais.

A consulta a seguir demonstra a possibilidade de emprego das propriedades predefinidas para intervalos na cláusula `SELECT`. Além das funções `EVER` e `PRESENT`, o que se destaca é a recuperação do **histórico dos intervalos de validade do atributo `valor`**:

```
SELECT EVER valor.vInterval
FROM Computador
WHERE PRESENT(valor > 1500)
```

Como apresentado anteriormente, as propriedades para recuperação de intervalos também podem ser utilizadas na composição de condições temporais na cláusula `WHERE`. No entanto, os operadores relacionais são pouco adequados para realizar comparações, principalmente pelo formato característico de se representar intervalos. Diante dessa situação, foram definidos seis operadores, cujas funcionalidades são assim descritas:

- **`INTERSECT`**: operador utilizado para verificar a existência de intersecção entre intervalos.

  Configura-se existência de intersecção quando há, ao menos, um instante coincidente que integre ambos intervalos analisados. Para isso, se o instante inicial do intervalo um for menor ou igual ao instante final do intervalo dois, e o instante final do intervalo um for maior ou igual ao instante inicial do intervalo dois, estará configurada uma situação de interseção entre intervalos

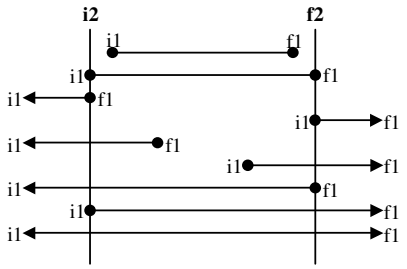

FIGURA 4.1 – Exemplos de intersecção entre intervalos

Exemplo:

```
SELECT processador
FROM Computador
WHERE EVER valor.vInterval INTERSECT [..01-10-2001]
```

A consulta recuperará o valor atual do atributo `processador` desde que algum intervalo de validade do atributo `valor` tenha intersectado o intervalo que compreende todo o infinito passado até o instante 01-10-2001. Destaca-se nessa consulta o uso da função `EVER`, da propriedade de recuperação de intervalos (`vInterval`) e do operador de comparação de intervalos (`INTERSECT`).

- **`OVERLAP`**: operador utilizado para verificar a existência de sobreposição de intervalos.

  A sobreposição de intervalos ocorre quando um intervalo está contido em outro (intervalo um sobrepõe/contém o intervalo dois), ou seja, o instante inicial do intervalo um deve ser menor ou igual ao instante inicial do intervalo dois, e o instante final do intervalo um necessita ser maior ou igual ao instante final do intervalo final do intervalo dois.

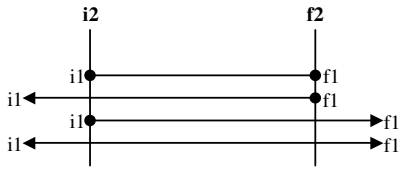

FIGURA 4.2 – Sobreposição de intervalos

Exemplo:

```
SELECT iLifetime
FROM Computador
WHERE processador.vInterval OVERLAP valor.vInterval
```

Nesse exemplo são retornados os tempos de vida inicial dos objetos cujo intervalo de validade atual do atributo processador intersecte todo o intervalo de validade atual do atributo valor.

- **`EQUAL`**: operador utilizado para verificar a identidade/igualdade entre dois intervalos.

A identidade entre intervalos é verificada quando o instante inicial do intervalo um coincide com o instante inicial do intervalo dois, e o instante final do intervalo um é o mesmo que o instante final do intervalo dois.

FIGURA 4.3 – Igualdade entre intervalos

- **`BEFORE:`** operador utilizado para verificar se um intervalo precede outro intervalo ou se um instante antecede um intervalo.

  Considera-se que um instante *i* antecede um intervalo quando o instante inicial do intervalo é maior que o instante *i*. Entre intervalos, assume-se que um intervalo precede outro quando o instante inicial e o instante final do intervalo um são menor que o instante inicial do intervalo dois.

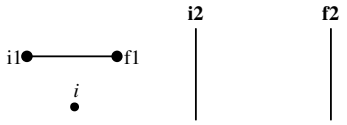

FIGURA 4.4 – Intervalo um e instante *i* precedem o intervalo dois

- **`INTO:`** operador utilizado para constatar se um instante está contido em um intervalo ou se um intervalo está contido em outro intervalo.

  Um intervalo está "dentro" de outro quando o instante inicial do intervalo um é maior que o instante inicial do intervalo dois, e o instante final do intervalo um é menor que o instante final do intervalo dois. Para um instante estar contido em um intervalo, ele deve ser maior que o instante inicial e menor que o instante final do intervalo.

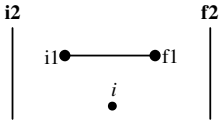

FIGURA 4.5 – Intervalo um e instante *i* contidos no intervalo dois

- **`AFTER:`** operador utilizado para verificar se um instante sucede um intervalo ou se um intervalo sucede outro intervalo.

  Se o instante final de um intervalo for menor que o instante *i*, tem-se que o instante sucede o intervalo. Caso o instante final do intervalo dois seja menor que o instante inicial e o instante final do intervalo um, diz-se que o intervalo um sucede o intervalo dois.

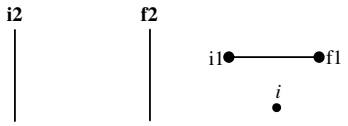

FIGURA 4.6 – Intervalo um e instante *i* sucedem o intervalo dois

Para ilustrar a utilização dos operadores `BEFORE`, `INTO` e `AFTER`, é apresentada a seguinte consulta com uma condição composta:

```
SELECT EVER valor
FROM Computador
WHERE PRESENT(valor.viInstant BEFORE [..10-10-2001]) AND
      processador.vInterval INTO [05-10-2001..10-10-2001] AND
      PRESENT(DataFabricao AFTER processador.vInterval)
```

A condição da consulta apresenta três possibilidades de emprego dos operadores. A primeira verifica se um instante obtido por uma propriedade predefinida para recuperação de instantes antecede um determinado intervalo. Na segunda, está sendo identificado se um intervalo está contido dentro de outro. Por fim, é verificado se uma propriedade temporal sucede um dado intervalo.

Além dos operadores, esta seção discutiu as demais características temporais da TVQL. Para concluir, são apresentadas as possibilidades de condições temporais a partir da combinação das propriedades predefinidas para instantes e intervalos com os operadores (**TABELA 4.2**). Destaca-se que o elemento `propriedade` das colunas *Propriedade* e *Operando* equivale a um atributo definido pelo usuário com o tipo de dado correspondente a um instante.

TABELA 4.2 – Condições temporais (propriedades e operadores)

Condição Temporal		
Propriedade	**Operador**	**Operando**

`tiInstant`
`tfInstant`
`viInstant`
`vfInstant`
`propriedade`
	`=`
	`<>`
	`>`
	`<`
	`>=`
	`<=`
		`NOW`
		`instante`
		`propridade`

`tInterval`
`vInterval`
	`BEFORE`
	`INTO`
	`AFTER`
		`[instante1..instante2]`
		`[..instante]`
		`[instante..]`
		`tInterval`
		`vInterval`

	`INTERSECT`
	`OVERLAP`
	`EQUAL`	

1.15 Características de Versionamento

As funcionalidades de versionamento suportadas pela TVQL são aplicáveis em suas três cláusulas principais: `SELECT-FROM-WHERE`. Além das propriedades e das funções para tratar aspectos de versão (utilizáveis nas cláusulas `SELECT-WHERE`), a TVQL define um comportamento especial para cláusula `FROM`.

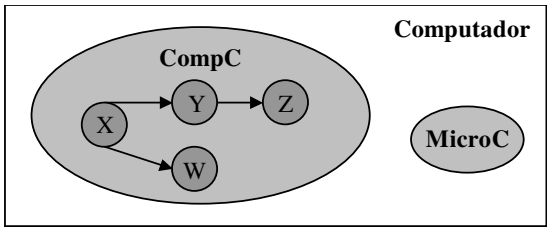

FIGURA 4.7 – Objetos das classe Computador

Nos exemplos de consulta mostrados até agora, os nomes das classes cujas instâncias de objetos deveriam ser consultadas eram especificados na cláusula `FROM`. Com o adendo do conceito de versão, além dos objetos, as versões desses objetos também devem ser analisadas. Considerando a **FIGURA 4.7**, que representa objetos e versões da classe `Computador`, têm-se duas formas de compor a cláusula `FROM`:

- **Definição do nome da classe:** quando apenas o nome da classe é especificado, são retornados os objetos que não apresentam versões (`MicroC`) e a versão corrente dos objetos que possuem versões (considera-se que `Z` seja versão corrente do objeto `CompC`).

```
FROM Computador
```

- **Definição do nome da classe e da palavra reservada `versions`:** quando o nome da classe é seguido pela propriedade `versions`, todas as versões são recuperadas (versões `X`,`Y`,`Z`,`W` do objeto `CompC`), assim como aqueles objetos que não apresentam versões (`MicroC`). Tanto para o TVM quanto para a TVQL, um objeto sem versões é tratado como uma versão.

```
FROM Computador.versions
```

Com a possibilidade de recuperar as versões de objetos, foram definidos três grupos de funções para retornar as informações referentes aos objetos versionados e suas versões:

- **Funções que analisam o estado de uma versão:** uma versão pode assumir os estados *Working*, *Stable*, *Consolidated* e *Deactivated*. Para verificar o estado, são definidas funções de mesmo nome com a adição do prefixo *is*: **`isWorking`**, **`isStable`**, **`isConsolidated`** e **`isDeactivated`**.

```
SELECT processador
FROM Computador.versions c
WHERE c.isWorking AND processador.viInstant <> Now
```

A consulta acima demonstra a forma de utilização das funções. O retorno da função é sempre um valor lógico. Nesse caso, se está analisando se as versões de objetos da classe `Computador` (representadas pelo alias `c`) estão em trabalho. Em caso afirmativo, a expressão é retornada como verdadeira.

Ainda existe uma variação no uso dessas funções. Com o acréscimo do prefixo `At`, torna-se possível verificar o estado das versões num dado momento, através da passagem de um instante como parâmetro:

```
SELECT processador
FROM Computador.versions v
WHERE v.isConsolidatedAt("25-08-2001")
```

- **Funções que analisam a hierarquia de herança:** para ilustrar o uso das funções que atuam sobre a hierarquia de herança, considera-se que a classe `Computador` estabelece um relacionamento de herança por extensão com a classe `Notebook` (**FIGURA 4.8**). Dessa forma, os objetos e as versões instanciados nos dois níveis da hierarquia são tratados de forma independente e são estabelecidas correspondências (mapeamentos) entre versões de um objeto em uma classe e versões de seu ascendente na superclasse. Obrigatoriamente, uma versão criada no nível inferior deve corresponder a um ascendente na superclasse. Todavia, uma versão não necessariamente precisa estar relacionada com um descendente. Admite-se também que uma versão possua múltiplas correspondências, ou seja, vários ascendentes e descendentes.

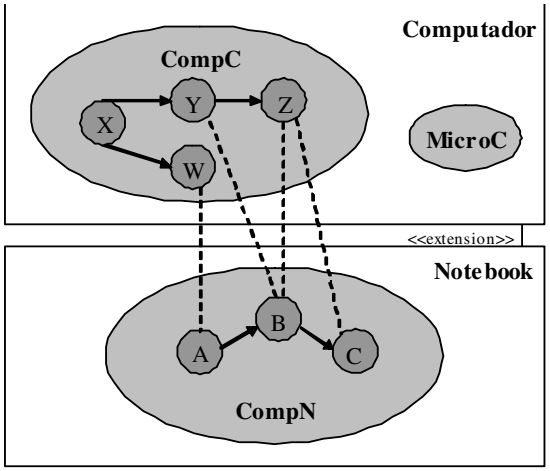

FIGURA 4.8 – Relacionamento de herança por extensão entre classes

Para verificar a existência de correspondências entre versões, são definidas duas funções na TVQL: **`isAscendantOf`** e **`isDescendantOf`**. A primeira analisa se uma versão é ascendente de outra versão; a segunda examina se a versão que invoca a função é descendente daquela passada como parâmetro:

```
SELECT c.valor
FROM Computador.versions c, Notebook
WHERE c.isAscendantOf(Notebook)
```

```
SELECT n.bateria
FROM Computador.versions c, Notebook.version n
WHERE n.isDescendantOf(c)
```

Na consulta que utiliza a função `isAscendantOf`, será retornado o atributo `valor` daquelas versões que são ascendentes de `Notebook`. Como apenas o nome da classe foi especificado, serão considerados todos os objetos sem versões e as versões correntes dos objetos versionados. No exemplo existe um único objeto versionado cuja versão atual é `C`. Assim, será retornado o atributo `valor` da versão ascendente de `C`, que, neste caso, é a versão `Z`. Na segunda consulta, supondo que a classe `Notebook` define um atributo `bateria`, deseja-se retornar este atributo para aquelas versões que são descendentes das versões de objetos da classe `Computador`. Como todas são descendentes de alguma versão, serão recuperados os valores do atributo `bateria` das versões `A`, `B` e `C`.

- **Funções que analisam a hierarquia de derivação:** a hierarquia de derivação restringe-se à evolução do conjunto de versões de um objeto versionado. Para a análise dessa hierarquia, são definidas as funções: **`isFirst`** (verifica se a versão é a primeira criada na hierarquia), **`isLast`** (verifica se é a última criada), **`isSuccessorOf`** (verifica se versão é sucessora de outra), **`isPredecessorOf`** (verifica se a versão é antecessor de outra), **`isCurrent`** (verifica se a versão é a corrente), **`isUserCurrent`** (verifica se a versão corrente foi definida pelo usuário) e **`isConfiguration`** (verifica se a versão faz parte de uma configuração). Com exceção das funções `isPredecessorOf` e `isConfiguration`, as demais possuem uma variante a partir do acréscimo do sufixo `At`. Esta variante, através da passagem de um parâmetro, permite verificar a hierarquia de derivação em um instante específico determinado pelo usuário. Para exemplificar o uso das funções de análise da hierarquia, é apresentada a seguinte consulta:

```
SELECT EVER c.processador
FROM Computador.versions c
WHERE c.isLastAt(30-10-2001) AND PRESENT(c.isPrecessorOf(Y)) AND
      PRESENT(c.isCurrent) AND PRESENT(c.isFirst)
```

Cabe destacar neste exemplo, além do uso das funções de derivação, a combinação de funções temporais com funções de versionamento. Almejou-se retornar nesta consulta o histórico do atributo `processador` da versão, que era a última em 30-10-2001 e que atualmente seja a primeira versão, a versão corrente e a antecessora da versão `Y`.

Além das funções, a TVQL apresenta um conjunto de propriedades que pode ser utilizado tanto na cláusula `WHERE` quanto na cláusula `SELECT`. Tais propriedades permitem recuperar informações do controle do objeto versionado que normalmente não estão acessíveis ao usuário:

- **`nickname`**: recupera o nome de um objeto e/ou versão;
- **`entity`**: manipula as entidades armazenadas;

- **`firstVersion`**: recupera o tvOID da primeira versão do objeto;
- **`lastVersion`**: recupera o tvOID da última versão do objeto;
- **`currentVersion`**: recupera o tvOID da versão corrente do objeto;
- **`configurationCount`**: obtém o número de versões configuradas;
- **`versionCount`**: obtém o número de versões do objeto;
- **`nextVersionNumber`**: obtém o número da próxima versão;
- **`userCurrentFlag`**: retorna um valor lógico indicando se a versão corrente foi especificada pelo usuário.

No exemplo abaixo é mostrada uma forma de emprego das propriedades. Destaca-se o uso das propriedades `nickname` e `firstVersion` na cláusula `SELECT` e a comparação entre duas propriedades que retornam `tvOID`:

```
SELECT c.nickname, c.firstVersion
FROM Computador.versions c
WHERE c.versionCount = 4 AND c.currentVersion =  c.lastVersion
```

Para finalizar esta subseção, é apresentado um quadro-resumo (**TABELA 4.3**), destacando todas as funções e as propriedades com características de versionamento.

TABELA 4.3 – Propriedades e funções com características de versão

Tipo de função/propriedade	Forma de utilização

Análise do estado	`version.isWorking: boolean`
	`version.isWorkingAt(instant): boolean`
	`version.isStable: boolean`
	`version.isStableAt(instant): boolean`
	`version.isConsolidated: boolean`
	`version.isConsolidatedAt(instant): boolean`
	`version.isDeactivated: boolean`
	`version.isDeactivatedAt(instant): Boolean`

Análise da hierarquia de extensão	`version.isAscendantOf(version):boolean` `version.isDescendantOf(version):boolean`

Análise da hierarquia de derivação	`version.isFirst: boolean`
	`version.isFirstAt(instant): boolean`
	`version.isLast: boolean`
	`version.isLastAt(instant): boolean`
	`version.isPredecessorOf(version): boolean`
	`version.isSuccessorOf(version): boolean`
	`version.isSuccessorOfAt(version, instant): boolean`
	`version.isCurrent: boolean`
	`version.isCurrentAt(instant): boolean`
	`version.isUserCurrent: boolean`
	`version.isUserCurrentAt(instant): boolean`
	`version.isConfiguration: boolean`

Manipulação do controle do objeto versionado	`version.configurationCount: integer`
	`version.currentVersion: tvOID`
	`version.firstVersion: tvOID`
	`version.lastVersion: tvOID`
	`version.nextVersionNumber: integer`
	`version.userCurrentFlag: boolean`
	`version.versionCount: integer`
	`version.nickname: string`
	`version.entity: string`

1.16 Considerações Finais

Com base nas particularidades do Modelo Temporal de Versões, foi definida uma linguagem de consulta suficientemente capaz de consultar aplicações modeladas a

partir do TVM. A linguagem busca unir o conceito de tempo com o conceito de versão em suas cláusulas, propriedades, funções e operadores. Até onde se tem conhecimento, trata-se de uma das linguagens pioneiras na união desses conceitos.

A TVQL baseia sua estrutura na SQL, o que permite uma assimilação mais rápida da forma de utilização de suas novas funcionalidades. A respeito do tratamento de aspectos temporais, além das propriedades definidas especificamente de acordo com o modelo, cabe destacar o emprego das palavras reservadas `EVER` e `PRESENT`. As duas permitem flexibilizar a definição de consultas, visto que todo o histórico ou apenas o valor atual dos dados pode ser considerado em qualquer ponto da consulta, seja na seleção dos valores, seja na especificação da condição.

Considerando os aspectos de versionamento, foram definidas funções capazes de analisar o comportamento das versões dentro da hierarquia de derivação, assim como na hierarquia de extensão. A possibilidade de recuperar apenas objetos ou os objetos e suas versões pode ser analisada como um variante de destaque dentro da linguagem.

Até o momento foram definidas as características básicas para suportar o modelo. Existe a necessidade de criação de novas operações, que permitam expressar consultas mais complexas. Suportar diferentes granularidades de tempo, dentre outras questões, é algo ainda não definido e que deverá ser considerado em trabalhos futuros.

5 Representação de Instância do TVM em XML

Quando existe referência ao gerenciamento de versões em documentos XML, o que se está tratando são instâncias completas e não partes de documentos. Algumas pesquisas sobre versões em XML [CHI 2000, CHI 2001] abordam técnicas de armazenagem e recuperação de versões de documentos que possuem mudanças estruturais significativas. Entretanto, ainda não se tem conhecimento da aplicação de versionamento em nível de elementos e de atributos dentro de um documento.

Já a inclusão de aspectos temporais para a manutenção do histórico de elementos e atributos [AMA 2000, MAN 2001] tem sido pesquisada, sendo que alguns conceitos definidos no modelo proposto por Amagasa estão presentes na representação criada especificamente para este trabalho.

A representação discutida nesta seção busca especificar possíveis objetos e versões, instanciadas a partir do Modelo Temporal de Versões em um formato XML. O trabalho mais próximo ao que está sendo proposto aqui é o mapeamento entre esquemas orientados a objetos e esquemas XML [BOU 2001]. Porém, o autor não se preocupa com a representação de instâncias.

Neste capítulo, não são apresentadas regras para o mapeamento de objetos do TVM para XML. Para que isso ocorresse, previamente, uma aplicação modelada a partir do TVM deveria ser mapeada para um banco de dados relacional. Posteriormente, os dados presentes na base relacional povoariam um documento XML com base na representação proposta nessa pesquisa. Um mapeamento bastante simplificado foi definido sobre o banco de dados IBM DB2, contudo a concepção do documento XML não foi atingida (ver capítulo 7).

O presente tópico inicia descrevendo a escolha realizada entre dois modelos de esquemas XML. Segue detalhando o vocabulário definido para a representação, tanto de objetos convencionais quanto de objetos temporais e versionados, sendo finalizado com algumas considerações acerca das deficiências e das vantagens da representação criada.

1.17 DTD ou XML Schema?

Desde o princípio dos estudos realizados nesta pesquisa, destacando aqueles relacionados com a definição de um modelo XML representativo para dados originários de aplicações modeladas a partir do Modelo Temporal de Versões, sempre se estabeleceu a dúvida entre qual das propostas seria a mais adequada para descrever a estrutura do documento XML que iria representar as instâncias do TVM.

Em quase todo o desenvolvimento do trabalho foi assumida a DTD como alternativa para alcançar o objetivo. Essa escolha se deveu, principalmente, à simplificação adotada para representar as instâncias. Como pode ser constatado no

Anexo 2 DTD da representação, tanto as propriedades de controle do modelo quanto os atributos definidos para a aplicação estão representados em uma gramática bastante concisa e de fácil compreensão. Assim sendo, considerou-se que os recursos disponibilizados pela DTD eram suficientes para a definição de um modelo eficaz de representação.

Contudo, a limitação no controle de restrições e a quase inexistência de tipos de dados impediam que a representação fosse refinada. Buscando enriquecer um pouco mais esse modelo de representação, foi adotada a XML Schema. Além do suporte muito mais qualificado para tipos de dados, principalmente aqueles de concepção temporal, e da flexibilidade para criação de restrições e tipos de dados simples, ela descreve de forma mais inteligível o vocabulário proposto.

1.18 Discussão da XML Schema Criada para a Representação

O esquema modelado (

Anexo 3 XML Schema da representação) é discutido, em detalhes, nesta subseção. As classes do Modelo Temporal de Versões são comentadas juntamente com exemplos de possíveis instâncias. Concomitantemente, a representação em XML e o trecho da XML Schema também são mostrados e explicados.

1.18.1 Diferenciação entre instâncias

O principal objetivo da descrição em XML era representar instâncias com características de tempo e de versão. No entanto, nem todas as instâncias de uma aplicação necessariamente são temporais e versionadas. Essas instâncias convencionais (sem tempo e sem versão) também devem estar representadas no documento XML.

Para fins de pré-seleção, as instâncias foram divididas em dois grupos: os objetos sem tempo e versão são agrupados sob o elemento `NonTemporalObjects`, diferentemente daquelas instâncias da classe `TemporalVersion`, que estão congregadas sob o elemento `TemporalObjects`. Acima desses elementos que separam os objetos de acordo com suas características, encontra-se o elemento `ObjectInstances`, raiz do documento.

Num primeiro momento, foi considerada a possibilidade de representar em documentos distintos cada grupo de instâncias, com o que seria evidenciada a diferenciação entre elas. Contudo, analisando o processamento de consultas, se o usuário desejasse, por exemplo, consultar todas as instâncias independentemente de suas características, ele necessariamente precisaria considerar a união dos dois documentos como escopo da consulta. A união de fontes, como se sabe, é um processo caro em banco de dados e ainda mais acentuado em documentos XML.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <xsd:schema xmlns:xsd="http://www.w3.org/2000/10/XMLSchema">
3  <xsd:element name="ObjectInstances">
4     <xsd:complexType>
5        <xsd:sequence>
6           <xsd:element name="NonTemporalObjects minOccurs="0"
7                        maxOccurs="1" type="ObjectsType"/>
8           <xsd:element name="TemporalObjects" minOccurs="0"
9                        maxOccurs="1" type="ObjectsType"/>
10       </xsd:sequence>
11    </xsd:complexType>
12 </xsd:element>
```

O trecho acima representa a parte inicial do esquema, destacando as regras de definição dos elementos que agrupam os dois diferentes conjuntos de instâncias. As duas primeiras linhas são padrões na descrição de um Schema[2]. A primeira caracteriza a declaração XML, que especifica a versão que está sendo usada (nesse caso a versão 1.0 da XML). Na segunda linha, é indicado o tipo de documento que está sendo declarado, ou seja, do tipo XML Schema.

Nota-se que todas as instruções de declaração presentes nesse trecho são precedidas pelo prefixo `xsd`. As instruções que possuem esse prefixo de *namespace* estão definidas sob a URI especificada na declaração do tipo de documento (linha 2). Se o usuário desejasse definir uma nova estrutura com nome `element`, para que se evite

[2] Quando houver referência ao nome "Schema" considera-se uma alusão ao documento com extensão .xsd, que modela a representação das instâncias do TVM em XML.

conflitos, ele deve vincular a esse nome um prefixo de *namespace* que represente o conjunto de declarações feitas por ele. O uso do prefixo é importante no processamento do Schema para que o processador valide o documento com a instrução correta.

Na linha 3 é definido o elemento raiz do documento, cujo nome é `ObjectInstances`. Esse é o único elemento sempre presente no documento XML proposto neste trabalho. Como o elemento `ObjectInstances` pode possuir subelementos, ele é declarado como um `xsd:complexType` (linha 4). Como seus dois subelementos podem coexistir, eles são definidos como integrantes de uma `xsd:sequence` (linha 5). Os elementos `NonTemporalObjects` e `TemporalObjects` são modelados, respectivamente, nas linhas 6-7 e 8-9. Excetuando seus nomes, eles apresentam as mesmas características, ou seja, podem ou não aparecer sob o elemento raiz. Se ocorrerem, eles só podem constar uma única vez.

Nesse fragmento do Schema, considerando o comparativo DTD/XML Schema, todas as funcionalidades poderiam ser representadas numa DTD, observando, obviamente, a diferença de sintaxe existente entre as duas abordagens. Independentemente do uso de DTD ou de XML Schema, a estrutura do documento XML começaria, assim, a ser concebida:

```
<ObjectInstances>
   <NonTemporalObjects>
      <!-- Instâncias sem características de tempo e de versão -->
   </NonTemporalObjects>
   <TemporalObjects>
      <!-- Instâncias temporais e versionadas -->
   </TemporalObjects>
</ObjectInstances>
```

1.18.2 Instâncias não-temporais e não-versionadas

Tanto o elemento `NonTemporalObjects` quanto o elemento `TemporalObjects` utilizam o tipo complexo `ObjectsType` como base. Ele define que os dois elementos que reúnem as instâncias podem ter um ou mais subelementos `Object`, sendo que cada subelemento associa as propriedades de uma instância:

```
<xsd:complexType name="ObjectsType">
   <xsd:sequence>
      <xsd:element name="Object" type="ObjectType"
                   minOccurs="1" maxOccurs="unbounded"/>
   </xsd:sequence>
</xsd:complexType>
```

O elemento `Object` também está associado a um tipo complexo, que define, além de elementos, atributos. O tipo `ObjectType` delimita a estrutura XML para representar objetos convencionais e objetos temporais versionados. A estrutura desse tipo é assim descrita:

```
<xsd:complexType name="ObjectType">
   <xsd:choice>
      <xsd:element name="TemporalObject" type="TemporalObjectType"/>
      <xsd:element name="Attributes" type="AttributesType"/>
   </xsd:choice>
   <xsd:attribute name="tvOID" type="BaseTVOIDType" use="required"/>
```

```
   <xsd:attribute name="name" type="xsd:string"/>
   <xsd:attribute name="class" type="xsd:string" use="required"/>
   <xsd:attribute name="type" type="xsd:string"/>
</xsd:complexType>
```

Os quatro atributos XML definidos por elementos `xsd:attribute` foram especificados para controlar algumas metainformações indispensáveis para um bom entendimento da representação e para facilitar o processo de consulta.

O atributo XML `type`, a princípio, não é utilizado quando da representação de objetos convencionais; justamente por isso ele não é declarado como obrigatório. Em objetos temporais e versionados, ele tem a função de indicar que o tipo de instância está caracterizada: uma versão ou um objeto versionado.

Também declarado como opcional, o atributo `name` pode definir um nome que identifique o objeto. Sua aplicação é relevante no acesso a um objeto quando não se tem conhecimento exato do tvOID que o identifica.

Para acessar objetos que apresentam a mesma classe como origem, foi definido o atributo XML `class`. Esse atributo foi especificado como obrigatório pelo simples fato de que todas instâncias descendem de classes. O atributo torna mais direto o processo de recuperação dos objetos e das versões, dispensando a verificação do tvOID para identificar à qual classe pertencem.

Dos quatro atributos, o mais representativo é o `tvOID`, que permite identificar, além da classe, a entidade e a versão de uma instância. Como em objetos convencionais não há o conceito de versão, o identificador da versão recebe `null`. Para validar o atributo, foi definido um tipo simples cuja estrutura é a seguinte:

```
<xsd:simpleType name="BaseTVOIDType">
   <xsd:restriction base="xsd:string">
      <xsd:pattern value="([0-9]+,[0-9]+,(([0-9]+)|null))|null"/>
   </xsd:restriction>
</xsd:simpleType>
```

O `BaseTVOIDType` tem como base o tipo `xsd:string`, que segue um padrão para a representação do tvOID. O tipo permite que o valor seja apresentado no formato 1,1,1 ou `null`. No caso do atributo `tvOID`, necessariamente o valor do tvOID deve ser informado; logo, o formato utilizado é o 1,1,1. Como esse tipo também é aplicado em elementos XML e, nesses, o valor pode ser omitido ou não existir, a representação `null` é utilizada para indicar ausência de valor.

Com esses atributos XML é possível atingir algumas informações sobre as instâncias. Contudo, a diferenciação explícita na representação dos objetos passa a existir a partir do emprego do elemento `xsd:choice`. Nos tipos mostrados até agora, só foram utilizadas construções com o elemento `xsd:sequence`, nas quais os elementos só poderiam aparecer em uma determinada seqüência dentro do documento. Com o uso do elemento `xsd:choice`, surge a alternativa de estruturar o documento de diferentes maneiras. Nessa situação específica, o elemento `Object` pode assumir duas formatações:

- se for definido `TemporalObject` como subelemento, configura-se a estruturação de um objeto temporal e versionado (detalhamento nas seções subseqüentes);

- se for definido `Attributes` como subelemento, um objeto convencional está sendo representado.

Como pode ser analisado na especificação do tipo `ObjectType`, não há referência a alguma forma de controle de relacionamentos e de métodos. Essas duas importantes características não foram consideradas na presente pesquisa. O enfoque foi totalmente voltado para o controle dos dados (atributos/propriedades) e para as relações dinâmicas dentro da hierarquia de extensão. Vale ressaltar que relacionamentos, por si só, representam grande quantidade de informação, o que justifica incondicionalmente sua adição numa futura extensão dessa representação.

Entretanto, o controle dos atributos de aplicação[3] é destacado no nível atual de representação através do uso do elemento `Attributes`. Este elemento não é unicamente utilizado para representar os atributos de aplicação de objetos convencionais; também tem sua aplicação em instâncias temporais e versionadas. Como uma instância convencional não possui atributos de controle, sua representação em XML parte diretamente para o tratamento dos valores de suas propriedades.

O tipo `AttributesType` norteia a definição do elemento `Attributes`. Esse tipo descreve uma listagem de todos os atributos de aplicação que podem ocorrer no documento, não importando se são temporais ou não. Cada atributo de aplicação definido pelo tipo pode ocorrer no máximo uma vez sob o elemento `Attributes`.

Para exemplificar a definição do tipo `AttributeType`, considera-se a existência de duas classes **não temporais e não versionadas**: Computador e Notebook. Os atributos `processador` e `discoRigido` são da classe Computador e as propriedades `dispApontador` e `duracaoBateria` pertencem a classe Notebook. A especificação do tipo `AttributesType` com base nesses atributos de aplicação seria a seguinte:

```
<xsd:complexType name="AttributesType">
   <xsd:sequence>
      <xsd:element name="processador" type="processadorType"
                   minOccurs="0" maxOccurs="1"/>
      <xsd:element name="discoRigido" type="discoRigidoType"
                   minOccurs="0" maxOccurs="1"/>
      <xsd:element name="duracaoBateria" type="duracaoBateriaType"
                   minOccurs="0" maxOccurs="1"/>
      <xsd:element name="dispApontador" type="dispositivoApontadorType"
                   minOccurs="0" maxOccurs="1"/>
   </xsd:sequence>
</xsd:complexType>
```

Todos os atributos de aplicação são modelados com o mínimo de ocorrências igual a zero. Caso fossem obrigatórios, todos os elementos que representam propriedades seriam apresentados à medida que um elemento `Attributes` fosse declarado. Isso não é o ideal visto que, por exemplo, para representar as instâncias da classe Computador, são necessários apenas os dois primeiros elementos, sendo descartados os dois últimos, que correspondem à classe Notebook. Nesse ponto, o usuário deve ter o bom senso e o cuidado de definir quais elementos são úteis na representação de cada uma das classes.

[3] Para diferenciar o uso da palavra "atributo", visto que em momentos está se tratando de atributos de elmentos XML e em outros de atributos modelados a partir do TVM, condicionou-se o uso da expressão "atributo de aplicação" para destacar esse último tipo e a expressão "atributo XML" para o primeiro.

Cada elemento definido no tipo `AttributesType` também tem como base um tipo que é especificado no atributo XML `type`. Por exemplo, o tipo `processadorType`:

```
<xsd:complexType name="processadorType">
   <xsd:sequence>
      <xsd:element name="Value" type="ValueStringType"/>
   </xsd:sequence>
</xsd:complexType>
```

Os tipos `processadorType`, `discoRigidoType`, `duracaoBateriaType` e `dispositivoApontadorType` definem que seus respectivos elementos possuirão um subelemento denominado `Value`. Nele estará contido o valor do atributo da aplicação. Coincidentemente, os quatros tipo utilizam `ValueStringType` como tipo base (ver

Anexo 3 XML Schema da representação). No entanto, outros tipos podem ser definidos visando tornar mais íntegra a representação dos valores.

Depois de apresentar esses vários níveis de tipos simples e tipos complexos que definem a estrutura de objetos convencionais, cabe mostrar o resultado no documento XML. Para isso serão consideradas uma instância da classe Computador e uma instância da classe Notebook:

Computador	
entidade	1
classe	5
versão	null
nome	
processador	**1 Ghz**
discoRigido	**20 Gb**

Notebook	
entidade	1
classe	10
versão	null
nome	N2001
duracaoBateria	**240 min**
dispApontador	**accupoint**

FIGURA 5.1 – Objetos com metainformações e atributos de aplicação

A representação dessas duas instâncias não temporais e não versionadas, desconsiderando a existência de outras instâncias temporais e versionadas, é feita da seguinte maneira:

```
<ObjectInstances>
   <NonTemporalObjects>
      <Object tvOID="1,5,null" class="Computador">
         <Attributes>
            <processador>
               <Value>1000</Value>
            </processador>
            <discoRigido>
               <Value>20</Value>
            </discoRigido>
         </Attributes>
      </Object>
      <Object tvOID="1,10,null" name="N2001" class="Notebook">
         <Attributes>
            <duracaoBateria>
               <Value>240</Value>
            </duracaoBateria>
            <dispApontador>
               <Value>accupoint</Value>
            </dispApontador>
         </Attributes>
      </Object>
   </NonTemporalObjects>
</ObjectInstances>
```

Alguns pontos podem ser destacados nessa breve representação:

- a inexistência de instâncias temporais e versionadas não invalida a representação. Caso existissem, elas poderiam ser definidas normalmente no mesmo documento, porém sob um elemento `TemporalObjects`;

- nesse exemplo, o elemento `NonTemporalObjects` comporta apenas duas instâncias. Todavia, ele pode representar *um* ou *n* objetos. Ressalta-se que todos os objetos, independentemente da classe, são descritos sob o mesmo elemento `NonTemporalObjects`;
- o `tvOID` dos dois objetos possui `null` para o identificador da versão, o que se justifica por se tratar de instâncias não temporais. Por possuírem o mesmo identificador de entidade, obviamente eles representam uma mesma entidade, contudo em níveis diferentes;
- a instância de Computador não possui um nome vinculado, diferente da instância de Notebook, cujo nome é `N2001`. Essa situação evidencia uma intervenção do usuário: se houver necessidade, define-se um nome; ele não é fundamental, pode ser preterido;
- o atributo XML `type` não é utilizado porque se trata de instâncias convencionais. Caso fossem temporais e versionadas, seria importante informar se a instância é uma versão ou é um objeto versionado;
- como foi anteriormente descrito, o tipo complexo `AttributesType` especifica todos os atributos de aplicação que podem aparecer no documento. No entanto, cabe ao usuário definir quais são pertinentes em cada objeto;
- o valor da propriedade está representado no elemento `Value`, que está inserido em cada elemento que indica um atributo de aplicação. Esse elemento adicional foi utilizado para manter a integridade entre atributos de aplicação normais e propriedades temporais. O detalhamento sobre o porquê dessa escolha é discutido na próxima subseção.

1.18.3 Instâncias temporais e versionadas

Todas as instâncias temporais e versionadas são representadas sob o elemento `TemporalObjects`. Da mesma forma que objetos convencionais, o elemento `Object` agrupa as propriedades de cada uma das instâncias. Porém, nem toda a instância temporal e versionada é um objeto (nesse caso um objeto versionado). Uma instância também pode ser uma versão, cuja diferença para um objeto versionado pode ser verificada pelo tvOID ou pelo atributo XML `type`.

Na definição do tipo `ObjectType`, os subelementos passíveis de utilização são: `Attributes` e `TemporalObject`. Como visto na seção anterior, quando um elemento `Attributes` é diretamente utilizado, está configurado um objeto não temporal e não versionado. Mas, se o usuário define `TemporalObject` como filho do elemento `Object`, o que se está representando é um objeto temporal e versionado.

O nome `TemporalObject`, assim como `Object`, não foi esporadicamente definido. Ele se origina da hierarquia de classes do Modelo Temporal de Versões, bem como `TemporalVersion` e `VersionedObjectControl`. Não apenas o nome é derivado do modelo, mas também os atributos definidos em suas classes, os quais são representados no Schema e no documento XML com o mínimo de alterações possíveis.

A classe `TemporalObject` define apenas uma propriedade denominada `alive`. Esse atributo de controle foi especificado no tipo complexo `TemporalObjectType` como um elemento que utiliza o tipo `aliveType` como base:

```
<xsd:complexType name="aliveType">
  <xsd:sequence>
    <xsd:element name="Value" type="ValueBooleanType"
                 minOccurs="1" maxOccurs="unbounded" default="false"/>
  </xsd:sequence>
</xsd:complexType>
```

Por se tratar de um atributo de controle temporal, não foi modelado como um atributo XML. Por ser temporal, pode assumir vários valores durante sua história, situação não suportada por um atributo XML, que representa apenas um único valor. Justamente o atributo `maxOccurs` do tipo `aliveType` determina a possibilidade de representar múltiplos valores que estarão contidos em diferentes elementos `Value`.

O elemento `Value` pode ser definido partindo de diferentes tipos complexos cuja variação se encontra apenas no tipo de dados escolhido. O atributo de controle `alive` é do tipo booleano no modelo, sendo aqui representado por um tipo complexo `ValueBooleanType`, que possui como tipo base um `xsd:boolean`:

```
<xsd:complexType name="ValueBooleanType">
  <xsd:simpleContent>
    <xsd:restriction base="xsd:boolean">
      <xsd:attribute name="tTimei" type="xsd:date"/>
      <xsd:attribute name="tTimef" type="xsd:date"/>
      <xsd:attribute name="vTimei" type="xsd:date"/>
      <xsd:attribute name="vTimef" type="xsd:date"/>
    </xsd:restriction>
  </xsd:simpleContent>
</xsd:complexType>
```

Os demais tipos complexos utilizados para especificar o elemento `Value` possuem a mesma estrutura que a apresentada acima. A diferença ocorre na definição do tipo base que é realizada no atributo XML `base`. No

Anexo 3 XML Schema da representação, estão descritos os demais tipos complexos que foram definidos e utilizados nesse trabalho.

Para representar os valores de atributos de aplicação não temporais, o elemento `Value` dispensa o uso de atributos XML especiais. No entanto, para propriedades temporais, como é o caso da propriedade `alive`, mostra-se indispensável utilizar os atributos XML definidos no tipo complexo. Eles foram especificados para representar os rótulos temporais de cada informação presente em um elemento `Value`. Nesta pesquisa, eles foram definidos com o tipo `xsd:date` para representar os tempos de transação e de validade da informação. De acordo com a aplicação, o usuário pode desejar modificar a granularidade temporal. Para isso, basta alterar o tipo de dado temporal no atributo XML `type`.

Para ilustrar o resultado no documento XML, considera-se que o atributo de controle `alive` teve o valor `true` por um determinado período e, posteriormente, passou a ter valor `false` como válido. O elemento `alive`, onde essas informações estão armazenadas, teria a seguinte formatação.

```
<alive>
   <Value tTimei="2001-06-21" tTimef="9999-12-31"
          vTimei="2001-06-21" vTimef="2001-06-29">true</Value>
   <Value tTimei="2001-06-21" tTimef="9999-12-31"
          vTimei="2001-06-30" vTimef="9999-12-31">false</Value>
</alive>
```

O atributo de controle `alive` é um dos elementos definidos pelo tipo complexo `TemporalObjectType`, que, necessariamente, deve aparecer na representação XML. O restante da representação de `TemporalObject` depende da natureza da instância temporal da qual se está tratando. Respeitando a hierarquia do Modelo Temporal de Versões, a classe `TemporalObject` pode ser especializada em `TemporalVersion` ou em `VesionedObjectControl`. Essa possibilidade também é prevista no elemento `xsd:choice` do tipo complexo `TemporalObjectType`:

```
<xsd:complexType name="TemporalObjectType">
   <xsd:sequence>
      <xsd:element name="alive" type="aliveType"/>
      <xsd:choice>
         <xsd:sequence>
            <xsd:element name="TemporalVersion"
                         type="TemporalVersionType"/>
            <xsd:element name="Attributes" type="AttributesType"/>
         </xsd:sequence>
         <xsd:element name="VersionedObjectControl"
                      type="VersionedObjectControlType"/>
      </xsd:choice>
   </xsd:sequence>
</xsd:complexType>
```

Se a instância a ser representada é uma versão ou um objeto versionado, devem ser utilizados os elementos `TemporalVersion` e `Attributes`, nessa seqüência. Caso seja um controle de objeto versionado, o elemento `VersionedObjectControl`.é aquele que deve ser empregado.

Objetos versionados e versões têm sua representação regulada pelo tipo `TemporalVersionType` que especifica, respeitando as diferenças entre os modelos, a classe `TemporalVersion`. Não existem diferenças estruturais em representar um objeto versionado e uma versão; o que existe é um enfoque conceitual diferenciado, no qual; o objeto versionado representa a versão corrente com o mesmo formato utilizado pelas demais versões. Como apresentado anteriormente, através do atributo XML `tvOID` ou através do atributo XML `type` do elemento `Object` é possível diferenciar um objeto versionado de uma versão.

O tipo `TemporalVersionType` determina que todos os atributos de controle definidos na classe `TemporalVersion` sejam representados na forma de elementos sob `TemporalVersion`. Excetuando os elementos `predecessor` e `configuration`, os demais são de natureza temporal, o que inviabiliza suas representações na forma de atributos XML. Esses dois atributos de controle destacados até poderiam ser declarados como atributos XML. No entanto, haveria uma descaracterização na apresentação, visto que atributos de controle seriam representados em formatos diferenciados. Para tornar a representação mais homogênea nesse ponto, todos os atributos de controle foram projetados como elementos XML.

```
<xsd:complexType name="TemporalVersionType">
   <xsd:sequence>
      <xsd:element name="ascendant" type="ascendantType"/>
      <xsd:element name="descendant" type="descendantType"/>
      <xsd:element name="predecessor" type="predecessorType"/>
      <xsd:element name="successor" type="successorType"/>
      <xsd:element name="configuration" type="configurationType"/>
      <xsd:element name="status" type="statusType"/>
   </xsd:sequence>
   <xsd:attribute name="vocOID" type="BaseTVOIDType"/>
</xsd:complexType>
```

Cada elemento é declarado com um tipo especificamente criado para atender as suas necessidades. Todos esses tipos podem ser vistos no

Anexo 3 XML Schema da representação, contudo foram destacados dois, `predecessorType` e `statusType`, para ilustrar suas funcionalidades gerais:

```
<xsd:complexType name="predecessorType">
   <xsd:sequence>
      <xsd:element name="Value" type="ValueTVOIDType" default="null"/>
   </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="statusType">
   <xsd:sequence>
      <xsd:element name="Value" type="ValueStatusType"
                   maxOccurs="unbounded" default="Working"/>
   </xsd:sequence>
</xsd:complexType>
```

No Modelo Temporal de Versões, o atributo de controle `predecessor` indica o `tvOID` da versão que antecede aquela que está sendo tratada. Como uma versão possui apenas um predecessor em toda a sua vida, o atributo de controle foi definido como não temporal. Na representação em XML, ele é tratado da mesma maneira. No tipo `predecessorType` não são vistos os atributos XML `minOccurs` e `maxOccurs` justamente porque apenas um `tvOID` estará representado, o que dispensa a definição de restrições para o número de ocorrências.

A mesma situação não é encontrada no tipo `statusType`. Durante sua história, uma versão pode assumir diferentes estados, passando pelo status `Working` e chegando, até, ao estado `Consolidated`. Para manter o histórico das variações de *status*, é indispensável que o elemento `Value` possa ocorrer múltiplas vezes. Por esse motivo, o atributo `maxOccurs` é definido como `unbounded`. Tal especificação é encontrada em todos os demais tipos complexos que definem atributos temporais de controle.

A modificação mais significativa dentre os tipos complexos restringe-se ao tipo de dados base. No tipo `ValueStatusType` foi definido um novo tipo simples como base. O tipo `BaseStatusType` foi projetado para delimitar o escopo de valores possíveis para representar o estado de uma versão. A sintaxe na qual foi declarado aparece a seguir:

```
<xsd:simpleType name="BaseStatusType">
   <xsd:restriction base="xsd:string">
      <xsd:enumeration value="Working"/>
      <xsd:enumeration value="Stable"/>
      <xsd:enumeration value="Consolidated"/>
      <xsd:enumeration value="Deactivated"/>
   </xsd:restriction>
</xsd:simpleType>
```

Além dos elementos, tipos simples e tipos complexos declarados para representar instâncias da classe `TemporalVersion`, foi definido um atributo XML denominado `vocOID` para indicar o relacionamento entre objetos das classes `TemporalVersion` e `VersionedObjectControl` do Modelo Temporal de Versões. O atributo está presente em toda representação de versão ou de objeto versionado, sendo que o valor nele armazenado é o `tvOID` do controle do objeto versionado. A função desse implemento é formalizar a interligação entre objetos de controle e objetos de aplicação.

Com a declaração do atributo XML `vocOID` está completa a declaração do elemento `TemporalVersion`. Todavia, para representar um objeto versionado ou versão, não basta realizar o tratamento dos atributos de controle do modelo. É necessário que os atributos de aplicação estejam igualmente representados. Tal representação é realizada sob o elemento `Attributes`. A definição das propriedades é feita da mesma forma que para objetos não temporais, ressaltando que instâncias temporais e versionadas podem ter atributos de aplicação temporais e não temporais definidos lado a lado.

No

Anexo 4 Instâncias do TVM representadas em XML estão representados objetos versionados, controles de objeto versionado e versões. Para ilustrar um exemplo de instância de `TemporalObject`, considera-se a versão com `tvOID` 10,1,1, sobre a qual podem ser realizados os seguintes comentários:

- assim como em objetos não temporais, a representação da instância estabelece-se a partir do elemento `Object`. A primeira diferença está no elemento posicionado em um nível acima na hierarquia, nesse caso, `TemporalObjects`;
- através do `tvOID` é possível estabelecer que a instância representada é uma versão, cujo identificador é o algarismo 1. Caso estivesse sendo tratado um objeto versionado, o algarismo 0 seria o identificador. Além do `tvOID`, o atributo `type` estabelece de forma mais explícita que a instância tratada é uma versão;
- seguindo a hierarquia do modelo, toda a instância temporal e versionada é um objeto da classe `TemporalVersion` ou uma versão de objeto. No entanto, a classe `TemporalVersion` é uma especialização da classe `TemporalObject`, sendo que esta última não é vista pelo usuário no modelo, mas aqui é representada por um elemento de mesmo nome. A principal função do elemento `TemporalObject` dentro da representação é agregar os elementos `alive`, `TemporalVersion` e `Attributes` que caracterizam um objeto versionado e/ou versão;
- como explicado anteriormente, o atributo de controle `alive` é representado através de um elemento. Isso porque um objeto versionado e/ou versão, depois de criado, pode ser desativado e "ressuscitado" várias vezes. Essas informações, representadas por valores booleanos, não devem ser descartadas. Logo, todo o histórico deve ser mantido e a forma escolhida foi utilizar um elemento XML dada a impossibilidade de um atributo XML representar múltiplos valores;
- após, o elemento `alive` está representado o elemento `TemporalVersion` e agrupa os elementos que representam os atributos de controle da classe `TemporalVersion`. No mesmo elemento `TemporalVersion` está definido o atributo XML `vocOID`, que faz referência ao controle do objeto versionado, nesse exemplo, 30,3,1;
- todo atributo de controle possui sua representação em um elemento de mesmo nome, no qual os valores estão armazenados em elementos `Value`. Para aqueles de natureza temporal, o elemento `Value` possui atributos XML para representar os rótulos temporais vinculados à informação. `Predecessor` e `Configuration` não são temporais, portanto os rótulos são desnecessários;
- terminada a representação dos atributos de controle, é feita a descrição dos atributos de aplicação, que estão reunidos sob o elemento `Attributes`, o qual aparece logo após o elemento `TemporalVersion`. No exemplo, todas as propriedades são temporais. Caso houvesse alguma propriedade não temporal, ela seria representada no mesmo nível que as demais, porém os rótulos não estariam presentes no elemento, assim como foi visto nos atributos.não temporais de controle.

A representação de objetos versionados e versões segue o modelo que foi comentado acima, mas nem sempre uma instância temporal é uma versão ou um objeto versionado. Um controle de objeto versionado também é uma instância temporal, contudo a classe da qual origina é `VersionedObjectControl`, e não `TemporalVersion`, como os demais objetos temporais e versionados já discutidos.

Para regular as instâncias da classe `VersionedObjectControl`, foi definido o tipo complexo `VersionedObjectControlType`. Para representar um controle de objeto versionado são necessários apenas atributos de controle, diferentemente de versões, em que os atributos de aplicação são fundamentais para caracterizá-las. Dessa forma, o tipo complexo especificado abaixo é suficiente para validar a estrutura de um objeto de controle:

```
<xsd:complexType name="VersionedObjectControlType">
   <xsd:sequence>
      <xsd:element name="configurationCount"
                   type="configurationCountType"/>
      <xsd:element name="currentVersion" type="currentVersionType"/>
      <xsd:element name="firstVersion" type="firstVersionType"/>
      <xsd:element name="lastVersion" type="lastVersionType"/>
      <xsd:element name="nextVersionNumber"
                   type="nextVersionNumberType"/>
      <xsd:element name="userCurrentFlag" type="userCurrentFlagType"/>
      <xsd:element name="versionCount" type="versionCountType"/>
   </xsd:sequence>
</xsd:complexType>
```

Também são definidos tipos para cada atributo de controle a ser representado. Dentre eles apenas `nextVersionNumber` não é temporal, sendo que o tratamento é o mesmo visto para atributos de controle da classe `TemporalVersion`.

São poucas as diferenças encontradas entre o núcleo (atributos de controle) da representação de `TemporalVersion` e a representação de `VersionedObjectControl`. Sugere-se o objeto de `tvOID` 30,3,1 (mesmo `tvOID` referenciado pela versão comentada anteriormente) para delinear um controle de objeto versionado representado em XML:

- uma instância de `VersionedObjectControl` também é representada pelo elemento `Object`, entretanto os atributos `name` e `type` são descartados. O primeiro porque não existe necessidade de nomear um objeto que não é acessado diretamente pelo usuário. Já o segundo só é utilizado para diferenciar instâncias de `TemporalVersion`, sendo desnecessário nesse contexto;
- como se trata de uma instância temporal, também possui os elementos `TemporalObject` e `alive`. O elemento `Attributes` não ocorre, pois a classe não possui atributos de aplicação. Ela apenas define atributos que controlam as versões de um objeto versionado;
- definido após o elemento `alive`, `VersionedObjectControl` une os elementos responsáveis pela representação dos atributos de controle das versões. Respeitando certas diferenças, o funcionamento é o mesmo do apresentado para `TemporalVersion`.

1.19 Considerações Finais

Neste capítulo, foi explanada a representação de objetos, objetos versionados, versões e controles de objeto versionado em um formato XML. O principal objetivo foi desenvolver uma representação, a mais objetiva possível, mantendo ao máximo as características do Modelo Temporal de Versões. Dessa forma, instâncias podem ser intercambiadas entre bancos de dados modelados a partir do TVM, bem como podem ser apresentadas de diferentes formas com o uso de folhas de estilo. Além dessas, outras aplicações podem ser enumeradas. Todavia, como o enfoque do trabalho é a extensão de uma linguagem de consulta, destaca-se como principal aplicação a recuperação de informações através de consultas sobre o documento XML.

No que tange à descrição de atributos de controle pertinentes ao modelo e à descrição de atributos de aplicação definidos pelo usuário, é possível considerar que se atingiu um bom resultado. Com os tipos de dados disponibilizados pela XML Schema, não há uma perda excessiva de representatividade de informações, considerando os tipos de dados disponíveis em bancos de dados convencionais. Além disso, a possibilidade de criação de tipos derivados torna a XML Schema ainda mais eficiente para definir restrições sobre os atributos de controle e de aplicação.

Contudo, algumas limitações podem ser verificadas. Mesmo com o bom suporte a tipos de dados proporcionados pela XML Schema, os de caráter temporal não são tão vastos quanto os disponibilizados em bancos de dados. Na representação propriamente dita, não são abordados métodos e relacionamentos. Porém, em extensões vindouras, é fundamental a adição dessas estruturas. E, para finalizar, as regras que apontam o mapeamento de objetos presentes em bancos de dados para a representação XML não foram apresentadas. Sugere-se que em trabalhos futuros essa lacuna seja preenchida, tornando completo o ambiente de geração e consulta de documentos com características de tempo e de versão.

6 Extensão da Linguagem de Consulta

Como objetivo principal da pesquisa, neste tópico são apresentadas e discutidas as implementações das funções de tempo e de versão que permitem recuperar, de forma mais direta e perceptível, as informações representadas em documentos cuja estrutura segue o esquema definido e explicado na seção anterior.

A idéia que norteou o desenvolvimento da pesquisa após a escolha da linguagem XQuery como base para a extensão foi desconsiderar qualquer modificação na sintaxe da linguagem. Feita a análise sobre um grupo de linguagens de consulta, viu-se que algumas disponibilizavam o que se chama de "funções extensíveis", dentre as quais a própria XQuery. Também se viu que os recursos propostos para a definição de novas funções na XQuery eram bastante avançados e permitiriam realizar a extensão sem que houvesse qualquer alteração em sua sintaxe original. Além disso, outra grande vantagem do uso de funções na XQuery foi a total independência de uma linguagem de programação para a implementação, visto que as sintaxes das funções são consultas definidas na própria linguagem com o acréscimo de expressões condicionais e funções da XPath.

Mantida a sintaxe padrão da XQuery, igualmente se buscou manter uma padronização das funções de extensão com os conceitos definidos na TVQL. A linguagem do Modelo Temporal de Versões foi definida em paralelo a esse trabalho com um enfoque voltado para a orientação a objetos. Mesmo assim, as propriedades e funções nela definidas foram adaptadas às necessidades de consulta a dados XML. Assim, manteve-se uma relação entre as linguagens, o que propicia seus usos em um ambiente de consulta temporal de versão integrado, com consulta a dados armazenados no banco de dados e consulta a dados XML.

Neste capítulo, as funções são descritas respeitando três grupos: funções auxiliares, funções temporais e funções de versão. A sintaxe e sua respectiva discussão são encontradas em cada uma das subseções. Exemplos de utilização não são aqui mostrados, mas serão largamente explorados no capítulo subseqüente.

1.20 Funções Auxiliares

Algumas funções auxiliares foram definidas visando facilitar a recuperação de objetos, versões e propriedades representadas no documento através de elementos e atributos. Dessa forma, o usuário não precisa saber a expressão de caminho completa para atingir um elemento que indica, por exemplo, o valor de uma propriedade. Basta, para uma situação como esta, utilizar a função `tv:value` em conjunto com a função `tv:property`. Essas e outras funções adicionais são apresentadas a seguir, destacando-se seus parâmetros de entrada, suas respectivas implementações e os resultados retornados.

1.20.1 tv:document

Qualquer documento a ser consultado pela XQuery deve ser atribuído a uma variável através da função `document` para ser manipulado. Buscando tornar direto o acesso às informações mais relevantes representadas em documentos com

características de tempo e de versão, foi proposta a função `tv:document`, que utiliza a função original como base.

```
define FUNCTION tv:document ($in_document) RETURNS objects
{
   for $objects in document($in_document)
                          /ObjectInstances/TemporalObjects/Object
   return $objects
}
```

Essa função é aplicada nas cláusulas `LET` e `FOR` e recebe, através do parâmetro `in_document`, o caminho onde se encontra o documento-alvo da consulta. Esse documento é previamente selecionado destacando apenas os elementos `Object`, que representam objetos temporais e versionados. Os elementos retornados serão utilizados nas demais cláusulas da consulta (`WHERE` e `RETURN`).

1.20.2 tv:now

A função `tv:now` apenas recupera o tempo atual do sistema. De acordo com a especificação da XQuery utilizada neste trabalho, existe apenas suporte a datas como unidade temporal. Logo, o emprego da função `tv:now` resulta na data atual do sistema. Como pode ser visto em sua brevíssima implementação, nenhum tipo de parâmetro de entrada é previsto.

```
define FUNCTION tv:now () RETURNS currentDate
{
   date()
}
```

Para recuperar a data atual do sistema, bastaria o emprego da função `date` disponibilizada pela XQuery. Todavia, para manter a relação com a TVQL, também foi definida na extensão a função `tv:now` (que simplesmente utiliza o retorno da função `date`).

1.20.3 tv:nickname

A função `tv:nickname` retorna um determinado objeto desde que seu nome seja informado como parâmetro de entrada (`in_objname`). O outro parâmetro, `in_tvObj`, corresponde a uma variável de iteração que armazena um conjunto de objetos que podem ou não ter sido previamente selecionados.

```
define FUNCTION tv:nickname ($in_objName, $in_tvObj) RETURNS object
{
   for $object in $in_tvObj[@name=$in_objName]
   return $object
}
```

A função basicamente “varre“ todos os objetos presentes na variável `in_tvObj`, verificando aqueles que possuem a string indicada pelo parâmetro `in_objName` coincidente ao valor representado pelo atributo XML `name`. O retorno será o elemento, com seus descendentes, que representa o objeto com o nome em questão.

1.20.4 tv:objects

Para recuperar todos os objetos de uma classe, sugere-se a função `tv:objects`. Através da variável `in_className` é informado o nome da classe da qual os objetos retornados devem ser instâncias. A variável `in_tvObj` possui o mesmo comportamento apresentado anteriormente, ou seja, armazena um conjunto de objetos que serão analisados pela função.

```
define FUNCTION tv:objects ($in_className, $in_tvObj) RETURNS objects
{
   for $objects in $in_tvObj[@class=$in_className and
                             @type="VersionedObject"]
   return $objects
}
```

Como resultado da função são apresentados elementos `<Object>` cujo atributo `class` confere com o nome da classe passado por parâmetro. Além disso, os objetos retornados devem possuir o valor do atributo `type` equivalente a "`VersionedObject`", configurando, assim, o retorno apenas de objetos versionados.

1.20.5 tv:property

Através da função `tv:property` são atingidos os atributos de aplicação representados no documento sob o elemento `Attributes` e recuperados todos os elementos `Value` que armazenam os valores assumidos pelas propriedades. O nome do atributo de aplicação requerido é passado à função através do parâmetro `in_property`. Já a recuperação dos valores dessa propriedade depende de uma verificação aos objetos indicados pela variável `in_tvObj`.

```
define FUNCTION tv:property ($in_property, $in_tvObj, $in_searchType)
       RETURNS setOfValues
{
   for $setOfValues in $in_tvObj//node()[name(.)=$in_property]/Value
   where if ($in_searchType = "EVER") then
            $setOfValues
         else
            $setOfValues[@vTimei <= date() and @vTimef >= date() and
                         @tTimef = "9999-12-31"])
   return $setOfValues
}
```

Para recuperar elementos `Value` é necessário, primeiro, ter acesso ao elemento que representa a propriedade, cujo nome é indicado pela variável `in_property`. No entanto, para alcançar a propriedade é preciso utilizar algumas funções adicionais disponibilizadas pela XPath. A primeira é a função `node()`, que seleciona todo e qualquer nodo dentro do contexto estipulado (descendentes dos elementos indicados por `in_tvObj`). Cumprida a primeira etapa, basta comparar o elemento correspondente à propriedade (através da função `name(.)`) com o nome da propriedade passada por parâmetro. Concluída essa etapa, o simples emprego da expressão `/Value` retorna todos elementos `Value`.requisitados como retorno da função.

O parâmetro `$in_searchType` e a cláusula `WHERE` definida no corpo da função foram utilizados visando à recuperação temporal das informações e serão explicados na seção que trata das funções temporais.

1.20.6 tv:value

A função `tv:value` só tem sua aplicação justificada quando utilizada em conjunto com a função `tv:property`. Tem como objetivo apresentar os elementos `Value` recuperados pela `tv:property` sem os atributos XML que representam os rótulos temporais.

```
define FUNCTION tv:value ($in_tvProperty) RETURNS values
{
   for $values in $in_tvProperty
   return <Value>{string($values)}</Value>
}
```

A função percorre cada um dos elementos `Value` da propriedade passada como parâmetro (`in_tvProperty`) e retorna o valor contido no elemento. Cada valor servirá de conteúdo para um novo elemento `Value` criado para fins de apresentação.

1.21 Funções Temporais

As funções temporais definidas na TVQL foram implementadas respeitando as peculiaridades do modelo de dados XML e da representação proposta no trabalho. Contudo, dois pontos importantes da TVQL tiveram de ser tratados de maneira bastante diferente daquela vista na linguagem do TVM. Os dois pontos referidos são a palavra reservada `EVER` e a função `PRESENT`.

A função poderia ser implementada normalmente na XQuery, diferentemente da palavra reservada. Para que `EVER` fosse implementada de maneira mais próxima àquela vista na TVQL, seria necessário modificar a sintaxe padrão da XQuery adicionando, por exemplo, a palavra reservada às cláusulas `WHERE` e `RETURN`. Entretanto, não foi objetivo desse trabalho alterar a sintaxe base da linguagem e, sim, estendê-la utilizando os recursos por ela disponibilizados, ou seja, as funções.

Para solucionar esse problema, a palavra reservada `EVER` e a função `PRESENT` foram levadas para cada uma das funções que manipulam diretamente valores temporais. A solução encontrada foi aquela vista, porém não comentada na função `tv:property`. Quando o parâmetro `in_searchType` recebe o valor `"EVER"`, todo o histórico será considerado; caso seja informado qualquer outro valor ou mesmo deixado vazio, assume-se o valor corrente como base de pesquisa. Para fazer a distinção do retorno que se dará, a implementação é tratada particularmente para cada função, mas normalmente são acrescidos alguns predicados à cláusula `WHERE`.

1.21.1 tv:tiInstant, tv:tfInstant, tv:viInstant e tv:vfInstant

Para recuperar o tempo de transação inicial, o tempo de transação final, o tempo de validade inicial e o tempo de validade final, foram implementadas, respectivamente, as funções `tv:tiInstant`, `tv:tfInstant`, `tv:viInstant` e `tv:vfInstant`. Todas elas recebem como parâmetro de entrada (`in_tvProperty`) os elementos `Value` recuperados pela função `tv:property`.

```
define FUNCTION tv:tiInstant ($in_tvProperty) RETURNS tiInstant
{
   for $tiInstant in $in_tvProperty
   return <tiInstant>{string($tiInstant/@tTimei)}</tiInstant>
```

```
}

define FUNCTION tv:tfInstant ($in_tvProperty) RETURNS tfInstant
{
   for $tfInstant in $in_tvProperty
   return <tfInstant>{string($tfInstant/@tTimef)}</tfInstant>
}

define FUNCTION tv:viInstant ($in_tvProperty) RETURNS viInstant
{
   for $viInstant in $in_tvProperty
   return <viInstant>{string($viInstant/@vTimei)}</viInstant>
}

define FUNCTION tv:vfInstant ($in_tvProperty) RETURNS vfInstant
{
   for $vfInstant in $in_tvProperty
   return <vfInstant>{string($vfInstant/@vTimef)}</vfInstant>
}
```

A implementação das quatro funções é semelhante variando apenas no atributo que representa cada um dos rótulos temporais e no elemento construído como resultado da função. Dependendo dos valores passados como parâmetro, a saída pode ser um conjunto de nodos ou um único elemento com o valor do rótulo temporal requisitado. O uso dessas funções restringe-se às cláusulas `WHERE` e `RETURN`.

1.21.2 tv:tInterval e tv:vInterval

Da mesma forma que as funções apresentadas anteriormente, as funções `tInterval` e `vInterval` também recuperam rótulos temporais, porém no formato de intervalos. Respectivamente, elas retornam o intervalo de transação e o intervalo de validade de uma propriedade temporal passada através do parâmetro `in_tvProperty`.

```
define FUNCTION tv:tInterval ($in_tvProperty) RETURNS tInterval
{
   for $tInterval in $in_tvProperty
   return if (string($tInterval/@tTimef) = "9999-12-31") then
             <tInterval>{string($tInterval/@tTimei)}..</tInterval>
          else
             <tInterval>{string($tInterval/@tTimei)}..
                        {string($tInterval/@tTimef)}</tInterval>
}

define FUNCTION tv:vInterval ($in_tvProperty) RETURNS vInterval
{
   for $vInterval in $in_tvProperty
   return if (string($vInterval/@vTimef) = "9999-12-31") then
             <vInterval>{string($vInterval/@vTimei)}..</vInterval>
          else
             <vInterval>{string($vInterval/@vTimei)}..
                        {string($vInterval/@vTimef)}</vInterval>
}
```

Os corpos das duas funções são basicamente iguais diferindo apenas nos atributos utilizados. O resultado da função é um conjunto de elementos construídos a

partir da combinação, dependendo da função, dos tempos inicial e final de validade ou de transação. Na cláusula `RETURN`, é definida a criação dos intervalos de acordo com duas condições possíveis:

- tempo final igual a "`9999-12-31`" – esta data foi determinada para representar o infinito futuro, sendo que o intervalo construído segue o formato "`datainicial..`". Não há nenhuma expressão condicional prevendo que a data inicial seja igual a "`0001-01-01`" justamente porque, no momento de inserção de uma informação, o tempo inicial sempre é informado, o que torna desnecessária uma comparação dessa ordem;
- tempo final diferente a "`9999-12-31`" – isto indica que os dois tempos são conhecidos; logo, podem ser recuperados do documento seguindo a notação "`datainicial..datafinal`".

1.21.3 tv:iLifeTime e tv:fLifeTime

A funções `iLifeTime` e `fLifeTime` recuperam o tempo de vida inicial e o tempo de vida final de objetos versionados, versões e controles de objeto versionado representados em elementos `Object` no documento. Como se trata de tempo de vida, só serão retornados os tempos daqueles objetos cujo valor do elemento `alive` seja igual a `true`, ou seja, tempos de objetos válidos.

```
define FUNCTION tv:iLifeTime ($in_tvObj) RETURNS iLifeTime
{
   for $iLifeTime in $in_tvObj
   return
     <iLifeTime>
       {string($iLifeTime/TemporalObject/alive/Value[.="true"]/@vTimei)}
     </iLifeTime>
}

define FUNCTION tv:fLifeTime ($in_tvObj) RETURNS fLifeTime
{
   for $fLifeTime in $in_tvObj
   return
     <fLifeTime>
       {string($fLifeTime/TemporalObject/alive/Value[.="true"]/@vTimef)}
     </fLifeTime>
}
```

As funções recebem como parâmetro de entrada um ou mais elementos `Object`. Na cláusula `RETURN` é verificado cada um dos elementos `Object`, sendo que será retornado o tempo inicial (`vTimei`) ou o tempo final (`vTimef`) daquele objeto cujo elemento `Value` da propriedade `alive` seja igual a `true`. Um novo elemento (`iLifeTime` ou `fLifeTime`) é apresentado como saída da função.

1.21.4 tv:equal

A função `tv:equal` verifica a existência de igualdade entre intervalos. Ela pode receber como parâmetro de entrada dois intervalos, um conjunto de intervalos e um intervalo ou dois conjuntos de intervalos, desde que eles sejam resultado de funções `tv:tInterval` ou `tv:vInterval`.

```
define FUNCTION tv:equal ($in_intervalA, $in_intervalB) RETURNS boolean
{
   not(empty(
      let $intervalA := $in_intervalA,
          $intervalB := $in_intervalB
      where $intervalA = $intervalB
      return $intervalA))
}
```

Os intervalos são atribuídos a duas variáveis de iteração (`intervalA` e `intervalB`) que são comparadas até que todos o elementos presentes nelas tenham sido verificados (processo semelhante ao produto cartesiano). Se, em alguma dessas verificações, for constatada igualdade, o elemento da variável `intervalA` é retornado. Contudo, o retorno da função não é um elemento e, sim, um valor booleano. Para atingir tal resultado, são empregadas as funções `not` e `empty` da XQuery.

Se há igualdade, ao menos um elemento representando um intervalo é retornado. Aplicando a função `empty` sobre esse resultado, obtém-se o valor `false`. Agora, usando a função `not` sobre o resultado anterior, é alcançada a real saída da função, nesse caso, o valor booleano `true`.

1.21.5 tv:intersect

A função `tv:intersect` analisa se um intervalo intersecta, em ao menos um ponto, um outro intervalo. Nela estão presentes dois parâmetros de entrada que seguem as mesmas possibilidades definidas para a função `tv:equal`. O detalhe na implementação da `tv:intersect` está na definição de uma função que visa desmembrar os intervalos em instantes inicial e final. Essa função adicional, cuja implementação é mostrada abaixo, denomina-se `tv:instantsFromInterval`.

```
define FUNCTION tv:instantsFromInterval ($in_interval) RETURNS interval
{
   if (substring($in_interval,1,2) = "..") then
   (
      <interval>
         <iInstant>0001-01-01</iInstant>
         <fInstant>{substring($in_interval,3,10)}</fInstant>
      </interval>
   )
   else
      if ((substring($in_interval,11,2) = "..") and
          (substring($in_interval,13,1) = "")) then
      (
         <interval>
            <iInstant>{substring($in_interval,1,10)}</iInstant>
            <fInstant>9999-12-31</fInstant>
         </interval>
      )
      else
         if ((substring($in_interval,11,2) = "..") and
             (substring($in_interval,13,1) != "")) then
         (
            <interval>
               <iInstant>{substring($in_interval,1,10)}</iInstant>
```

```
            <fInstant>{substring($in_interval,13,10)}</fInstant>
         </interval>
      )
      else
      (
         <interval>
            <iInstant>{$in_interval}</iInstant>
            <fInstant>{$in_interval}</fInstant>
         </interval>
      )
}
```

Para realizar a comparação entre os intervalos passados por parâmetro, é indispensável o conhecimento do instante inicial e do instante final que limitam cada um deles. A função auxiliar recebe um intervalo em uma das notações possíveis (`..instFinal` | `instInicial..` | `instInicial..instFinal` | `instante`) e o representa em elementos `iInstant` e `fInstant` sob o elemento `interval`. Dessa forma, utilizando os recursos básicos de comparação disponibilizados pela XQuery, facilmente se trata a questão da intersecção de intervalos com a definição dos predicados apresentados a seguir no corpo da função.

```
define FUNCTION tv:intersect ($in_intervalA, $in_intervalB)
       RETURNS boolean
{
   not(empty(
      let $intA := tv:instantsFromInterval($in_intervalA)
      let $intB := tv:instantsFromInterval($in_intervalB)
      where ($intA/iInstant <= $intB/fInstant) and
            ($intA/fInstant >= $intB/iInstant) and
            ($intA/iInstant <= $intA/fInstant) and
            ($intB/iInstant <= $intB/fInstant)
      return $intA))
}
```

O retorno da função `tv:intersect` também é um valor booleano, cuja obtenção é alcançada através da mesma combinação das funções `not` e `empty` apresentada na função `tv:equal`. Todavia, as variáveis de iteração não recebem diretamente os intervalos passados como parâmetro. Esses servem de entrada para a função auxiliar que atribuirá às variáveis os intervalos construídos na forma de elementos. Essa construção torna direta a comparação entre os intervalos, o que pode ser visto na cláusula `WHERE` da implementação.

1.21.6 tv:overlap

A função `tv:overlap` é uma variação da função `tv:intersect` apresentada anteriormente. Configura-se uma situação de sobreposição quando um intervalo intersecta completamente um outro. Para verificar uma intersecção, ao menos um ponto em comum deve existir entre dois intervalos. Para verificar uma sobreposição, o intervalo B deve estar contido no intervalo A.

```
define FUNCTION tv:overlap ($in_intervalA, $in_intervalB)
       RETURNS boolean
{
   not(empty(
```

```
      let $intA := tv:instantsFromInterval($in_intervalA)
      let $intB := tv:instantsFromInterval($in_intervalB)
      where ($intA/iInstant <= $intB/iInstant) and
            ($intA/fInstant >= $intB/fInstant) and
            ($intA/iInstant <= $intA/fInstant) and
            ($intB/iInstant <= $intB/fInstant)
      return $intA))
}
```

Ressalta-se a ordem na qual os intervalos devem ser passados como parâmetro: o intervalo sobreposto é o B; o intervalo que sobrepõe é A. As variáveis recebem os intervalos na forma de elementos que são comparados na cláusula `WHERE`. Verificada a sobreposição, é retornado o primeiro intervalo que sofre a ação das funções `empty` e `not`, resultando no saída lógica esperada para a função.

1.21.7 tv:before, tv:after e tv:into

As funções `tv:before`, `tv:after` e `tv:into` permitem verificar a posição relativa de um intervalo ou de um instante com relação a um intervalo base. Respectivamente, elas verificam se um instante ou um intervalo está antes do, após o, ou está contido no intervalo base. As três funções recebem dois parâmetros como entrada: o primeiro pode ser tanto um intervalo quanto um instante; o segundo é o intervalo base para a comparação.

```
define FUNCTION tv:before ($in_intervalA, $in_intervalB) RETURNS boolean
{
   not(empty(
      let $intA := tv:instantsFromInterval($in_intervalA)
      let $intB := tv:instantsFromInterval($in_intervalB)
      where ($intA/iInstant < $intB/iInstant) and
            ($intA/fInstant < $intB/iInstant) and
            ($intA/iInstant <= $intA/fInstant) and
            ($intB/iInstant <= $intB/fInstant)
      return $intA))
}

define FUNCTION tv:after ($in_intervalA, $in_intervalB) RETURNS boolean
{
   not(empty(
      let $intA := tv:instantsFromInterval($in_intervalA)
      let $intB := tv:instantsFromInterval($in_intervalB)
      where ($intA/iInstant > $intB/fInstant) and
            ($intA/fInstant > $intB/fInstant) and
            ($intA/iInstant <= $intA/fInstant) and
            ($intB/iInstant <= $intB/fInstant)
      return $intA))
}

define FUNCTION tv:into ($in_intervalA, $in_intervalB) RETURNS boolean
{
   not(empty(
      let $intA := tv:instantsFromInterval($in_intervalA)
      let $intB := tv:instantsFromInterval($in_intervalB)
      where ($intA/iInstant > $intB/iInstant) and
            ($intA/fInstant < $intB/fInstant) and
            ($intA/iInstant <= $intA/fInstant) and
```

```
            ($intB/iInstant <= $intB/fInstant)
      return $intA))
}
```

Desconsiderando a cláusula `WHERE`, na qual os predicados definidos são característicos para cada uma das funções, o restante do corpo é idêntico para as três, sendo que o funcionamento é o mesmo do explicado para as funções `tv:intersect` e `tv:overlap`.

1.22 Funções de Versão

Assim como as temporais, as funções que tratam aspectos de versionamento de objetos e versões buscam manter ao máximo a compatibilidade com os conceitos definidos na TVQL. A mudança mais significativa em algumas funções de versão é o adendo do parâmetro `in_searchType`, que define o raio de ação temporal da função e, conseqüentemente, da consulta.

São apresentadas, a seguir, as implementações de funções que recuperam o estado das versões, que permitem navegar na hierarquia de herança e na hierarquia de derivação e que acessam, de forma transparente, os controles de objeto versionado.

1.22.1 tv:versions

Com o objetivo de recuperar as versões de um objeto versionado, foi criada a função `tv:versions`, cujo uso é restrito às cláusulas `LET` e `FOR`. Esta recebe dois parâmetros como entrada:

- a variável `in_obj`, que recebe o objeto versionado (trata-se do elemento que representa o objeto versionado e não a string que indica seu nome);
- a variável `in_tvObj`, que concentra o conjunto de objetos e versões utilizado no processamento da função.

```
define FUNCTION tv:versions ($in_obj, $in_tvObj) RETURNS versions
{
   for $versions in $in_tvObj[@type="Version"]
   where $versions/TemporalObject/TemporalVersion/@vocOID =
         $in_obj/TemporalObject/TemporalVersion/@vocOID
   return $versions
}
```

A função, inicialmente, separa as versões dentre aqueles objetos presentes na variável `in_tvObj`. Posteriormente, cada uma das versões pré-selecionada tem o valor do atributo `vocOID` comparado com o valor do atributo de mesmo nome do objeto versionado, representado pela variável `in_obj`. Constatada a igualdade entre os atributos, o elemento que representa a versão em questão é retornado e apresentado como saída da função `tv:versions`.

1.22.2 tv:isWorking, tv:isStable, tv:isConsolidated e tv:isDeactivated

Para analisar o estado das versões, foram definidas quatro funções: `tv:isWorking`, `tv:isStable`, `tv:isConsolidated` e `tv:isDeactivated`. Elas recebem, através do parâmetro `in_tvObj`, as versões que servem de base para a verificação. O processo de constatação do estado pode considerar todo o histórico ou

apenas o valor atual. Para diferenciar essas duas situações, é utilizado o parâmetro `in_searchType` (uso análogo ao da função `tv:property`).

```
define FUNCTION tv:isWorking ($in_tvObj, $in_searchType) RETURNS boolean
{
   not(empty(tv:status("Working", $in_tvObj, $in_searchType)))
}

define FUNCTION tv:isStable ($in_tvObj, $in_searchType) RETURNS boolean
{
   not(empty(tv:status("Stable", $in_tvObj, $in_searchType)))
}

define FUNCTION tv:isConsolidated ($in_tvObj, $in_searchType)
       RETURNS boolean
{
   not(empty(tv:status("Consolidated", $in_tvObj, $in_searchType)))
}

define FUNCTION tv:isDeactivated ($in_tvObj, $in_searchType)
       RETURNS boolean
{
   not(empty(tv:status("Deactivated", $in_tvObj, $in_searchType)))
}
```

Como todas as funções utilizam, quase que em sua totalidade, o mesmo corpo na implementação, foi definida uma função auxiliar denominada `tv:status`, visando à reutilização de código. Além dos parâmetros `in_tvObj` e `in_searchType`, essa função recebe na variável `in_status` a string que representa o estado que se está querendo verificar durante o processamento da função.

```
define FUNCTION tv:status ($in_status, $in_tvObj, $in_searchType)
       RETURNS objects
{
   for $objects in $in_tvObj
   where if ($in_searchType = "EVER") then
            $objects/TemporalObject/TemporalVersion/status/Value =
            $in_status
         else
            $objects/TemporalObject/TemporalVersion/status/Value
            [@vTimei <= date() and @vTimef >= date() and
             @tTimef = "9999-12-31"] = $in_status
   return $objects
}
```

Definido o tipo de *status*, o conjunto de versões e o escopo temporal, a função auxiliar simplesmente analisa cada uma das versões buscando aquelas cujo valor armazenado no subelemento `Value` do elemento `status` seja idêntico ao passado à função pelo parâmetro `in_status`. Verificada a identidade, o elemento correspondente à versão é retornado à função que invocou `tv:status`. Posteriormente, o método de combinação das funções `not` e `empty` é novamente empregado para redundar no valor booleano de saída da função invocante.

1.22.3 tv:isWorkingAt, tv:isStableAt, tv:isConsolidatedAt e tv:isDeactivatedAt

Da mesma forma que as anteriores, estas quatro funções analisam o estado de uma ou mais versões. A primeira variação ocorre no próprio nome das funções, sendo acrescido o sufixo `At` a cada uma delas. O objetivo dessa variante é permitir que seja especificado o exato instante no qual o *status* deve ser verificado. Esse instante é passado à função através do parâmetro `in_instant`. Com o uso desse parâmetro, extingue-se o emprego de `in_searchType`, visto que todo o histórico passa a ser considerado. Por outro lado, o parâmetro `in_tvObj` continua com sua utilização normal..

```
define FUNCTION tv:isWorkingAt ($in_tvObj, $in_instant) RETURNS boolean
{
   not(empty(tv:statusAt("Working", $in_tvObj, $in_instant)))
}

define FUNCTION tv:isStableAt ($in_tvObj, $in_instant) RETURNS boolean
{
   not(empty(tv:statusAt("Stable", $in_tvObj, $in_instant)))
}

define FUNCTION tv:isConsolidatedAt ($in_tvObj, $in_instant)
       RETURNS boolean
{
   not(empty(tv:statusAt("Consolidated", $in_tvObj, $in_instant)))
}

define FUNCTION tv:isDeactivatedAt ($in_tvObj, $in_instant)
       RETURNS boolean
{
   not(empty(tv:statusAt("Deactivated", $in_tvObj, $in_instant)))
}
```

Novamente é definida uma função auxiliar, denominado `tv:statusAt`, visando reaproveitar estruturas comuns a todas as funções. Ela recebe o *status*, o conjunto de versões e o instante. Caso o *status* informado coincida com o valor válido no instante passado como parâmetro, ela retorna a versão. O que pode ser destacado na função é a condição imposta ao elemento `Value` na cláusula `WHERE`. No mais, a concepção é a mesma vista na função `tv:status`, desconsiderando, obviamente, a expressão condicional.

```
define FUNCTION tv:statusAt ($in_status, $in_tvObj, $in_instant)
       RETURNS objects
{
   for $objects in $in_tvObj
   where $objects/TemporalObject/TemporalVersion/status/Value
        [@vTimei <= $in_instant and @vTimef >= $in_instant and
         @tTimef = "9999-12-31"] = $in_status
   return $objects
}
```

1.22.4 tv:ascendantOf e tv:isAscendantOf

A função `tv:ascendantOf` recupera os elementos que representam as versões que são ascendentes daquela passada pelo parâmetro `in_version`. Os parâmetros `in_tvObj` e `in_searchType` possuem o mesmo funcionamento e uso das implementações anteriores.

```
define FUNCTION tv:ascendantOf ($in_version, $in_tvObj,
                                $in_searchType)
       RETURNS ascendantVersion
{
   let $version := $in_version//TemporalVersion/ascendant/Value
   for $ascVersion in $in_tvObj
   where if ($in_searchType = "EVER") then
            $ascVersion[@tvOID=$version]
         else
            $ascVersion[@tvOID=$version[@vTimei <= date() and
                        @vTimef >= date() and @tTimef = "9999-12-31"]]
   return $ascVersion
}
```

Na cláusula `LET`, a variável `version` recebe os valores correspondentes aos `tvOID` das versões ascendentes da versão presente na variável `in_version`. Esses `tvOID` são comparados com os `tvOID` das versões armazenadas em `ascVersion`. Verificada a igualdade, a versão presente em `ascVersion` é recuperada e apresentada como saída da função.

Já a função `tv:isAscendantOf` é empregada para verificar se `in_ascTvObj` é ascendente da versão `in_version`. O retorno da função, como todas aquelas que possuem o prefixo `is`, é um valor lógico.

```
define FUNCTION tv:isAscendantOf ($in_ascTvObj, $in_version,
                                  $in_searchType)
       RETURNS boolean
{
   not(empty(for $ascTvObj in $in_ascTvObj,
                 $version in $in_version
             where
                if ($in_searchType = "EVER") then
                   $version//ascendant/Value = $ascTvObj/@tvOID
                else
                   $version//ascendant/Value
                            [@vTimei <= date() and @vTimef >= date() and
                             @tTimef = "9999-12-31"] = $ascTvObj/@tvOID
             return $ascTvObj))
}
```

A variável `ascTvObj` recebe a versão supostamente ascendente, e a variável `version`, a versão que possivelmente sofre ascendência de `ascTvObj`. Na cláusula `WHERE` são comparados os valores dos ascendentes de `version` com o `tvOID` da possível versão ascendente. Constatada a relação de ascendência, a versão comprovadamente ascendente é retornada e transformada na saída booleana esperada para a função.

1.22.5 tv:descendantOf e tv:isDescendantOf

As funções `tv:descendantOf` e `tv:isDescendantOf` tratam da relação de descendência entre versões. A primeira recupera as instâncias descendentes de uma versão; já a segunda verifica se está estabelecida a relação de descendência entre duas versões. As implementações das duas funções são muito semelhantes às vistas nas funções `tv:ascendantOf` e `tv:isAscendantOf`. A diferenciação encontrada está no nome das variáveis de iteração utilizadas e nas expressões de caminho, que, ao invés de alcançarem elementos `ascendant`, atingem elementos `descendant`.

```
define FUNCTION tv:descendantOf ($in_version, $in_tvObj, $in_searchType)
       RETURNS descendantVersion
{
   let $version := $in_version//TemporalVersion/descendant/Value
   for $descVersion in $in_tvObj
   where if ($in_searchType = "EVER") then
            $descVersion[@tvOID=$version]
         else
            $descVersion[@tvOID=$version[@vTimei <= date() and
                        @vTimef >= date() and @tTimef = "9999-12-31"]]
   return $descVersion
}

define FUNCTION tv:isDescendantOf ($in_descTvObj, $in_version,
                                   $in_searchType)
       RETURNS boolean
{
   not(empty(for $descTvObj in $in_descTvObj,
                 $version in $in_version
             where if ($in_searchType = "EVER") then
                      $version//TemporalVersion/descendant/Value =
                      $descTvObj/@tvOID
                   else
                      $version//TemporalVersion/descendant/Value
                      [@vTimei <= date() and @vTimef >= date() and
                       @tTimef = "9999-12-31"] = $descTvObj/@tvOID
             return $descTvObj))
}
```

1.22.6 tv:firstVersion, tv:isFirst e tv:isFirstAt

A função `tv:firstVersion` recupera a primeira versão do objeto desde que aplicada sobre o elemento que representa um objeto versionado. O parâmetro `in_firstObj` recebe um objeto versionado na qual a obtenção da primeira versão é relevante. Os demais parâmetros de entrada já têm aplicação conhecida.

```
define FUNCTION tv:firstVersion ($in_firstObj, $in_tvObj,
                                 $in_searchType)
RETURNS firstVersion
{
   for $voc in $in_tvObj[@tvOID=$in_firstObj//TemporalVersion/@vocOID],
       $firstVersion in $in_tvObj
   where if ($in_searchType = "EVER") then
            $firstVersion/@tvOID =
            $voc//VersionedObjectControl/firstVersion/Value
```

```
            else
               $firstVersion/@tvOID =
               $voc//VersionedObjectControl/firstVersion/Value
                     [@vTimei <= date() and @vTimef >= date() and
                      @tTimef = "9999-12-31"]
   return $firstVersion
}
```

Inicialmente, a variável voc recebe o controle do objeto versionado e a variável `firstVersion` recebe o conjunto de elementos de onde redundará a primeira versão. O valor do `tvOID` de cada uma das versões presentes na variavel `firstVersion` é comparado com os `tvOID` obtidos a partir do subelemento `Value` do elemento `firstVersion`. Se os `tvOID` forem compatíveis, o elemento que representa a primeira versão é recuperado da variável `firstVersion`.

A função `tv:isFirst` verifica se a versão passada como parâmetro é a primeira dentro do seu objeto versionado. A versão a ser analisada é fornecida à função através do parâmetro `in_isFirst`. O `tvOID` da versão é comparado com os valores de `firstVersion` no controle do objeto versionado. Comprovada a identidade entre os `tvOID`, as funções `not` e `empty` tratam o objeto recuperado e retornam o valor booleano correspondente.

```
define FUNCTION tv:isFirst ($in_isFirst, $in_tvObj, $in_searchType)
       RETURNS boolean
{
   not(empty(
      for $isFirstObj in $in_isFirst,
          $voc in $in_tvObj[@tvOID=$isFirstObj//TemporalVersion/@vocOID]
      where if ($in_searchType = "EVER") then
               $voc//VersionedObjectControl/firstVersion/Value =
               $isFirstObj/@tvOID
            else
               $voc//VersionedObjectControl/firstVersion/Value
                     [@vTimei <= date() and @vTimef >= date() and
                      @tTimef = "9999-12-31"] = $isFirstObj/@tvOID
      return $isFirstObj))
}
```

A função `tv:isFirstAt` possui funcionamento semelhante à anterior, no entanto, ao invés do parâmetro `in_searchType`, é informado o instante em que supostamente a versão passada como parâmetro era a primeira no seu objeto versionado. Nota-se que a cláusula `WHERE` é mais simplificada que a função anterior, pois não há necessidade de utilização de expressão condicional e, sim, um breve predicado analisando os tempos de validade inicial e validade final dos valores de `firstVersion`.

```
define FUNCTION tv:isFirstAt($in_isFirst, $in_tvObj, $in_instant)
       RETURNS boolean
{
   not(empty(
      for $isFirstObj in $in_isFirst,
          $voc in $in_tvObj[@tvOID=$isFirstObj//TemporalVersion/@vocOID]
      where $voc//VersionedObjectControl/firstVersion/Value
                  [@vTimei <= $in_instant and @vTimef >= $in_instant and
                   @tTimef = "9999-12-31"] =
```

```
            $isFirstObj/@tvOID
      return $isFirstObj))
}
```

1.22.7 tv:lastVersion, tv:isLast e tv:isLastAt

A aplicação da função `tv:lastVersion` resulta no conjunto de elementos que representou ou ainda representa a última versão de um objeto versionado. Comparada com a função `tv:firstVersion`, a alteração mais significativa concentra-se no uso da aresta `lastVersion` no lugar da aresta `firstVersion` nas expressões de caminho. No restante, o funcionamento é o mesmo da função `tv:firstVersion`, anteriormente discutida.

```
define FUNCTION tv:lastVersion ($in_lastObj, $in_tvObj, $in_searchType)
       RETURNS lastVersion
{
   for $voc in $in_tvObj[@tvOID=$in_lastObj//TemporalVersion/@vocOID],
       $lastVersion in $in_tvObj
   where if ($in_searchType = "EVER") then
            $lastVersion/@tvOID =
            $voc//VersionedObjectControl/lastVersion/Value
         else
            $lastVersion/@tvOID =
            $voc//VersionedObjectControl/lastVersion/Value
                  [@vTimei <= date() and @vTimef >= date() and
                   @tTimef = "9999-12-31"]
   return $lastVersion
}
```

As funções `tv:isLast` e `tv:isLastAt` verificam se a versão passada como parâmetro é a última dentre as versões do objeto versionado. A segunda, como se sabe, recebe o instante no qual a condição é verificada. As duas têm saída lógica que segue a mesma linha das demais funções que possuem esse tipo de retorno.

```
define FUNCTION tv:isLast($in_isLast, $in_tvObj, $in_searchType)
       RETURNS boolean
{
   not(empty(
      for $isLastObj in $in_isLast,
          $voc in $in_tvObj[@tvOID=$isLastObj//TemporalVersion/@vocOID]
      where if ($in_searchType = "EVER") then
               $voc//VersionedObjectControl/lastVersion/Value =
               $isLastObj/@tvOID
            else
               $voc//VersionedObjectControl/lastVersion/Value
                     [@vTimei <= date() and @vTimef >= date() and
                      @tTimef = "9999-12-31"] = $isLastObj/@tvOID
      return $isLastObj))
}

define FUNCTION tv:isLastAt($in_isLast, $in_tvObj, $in_instant)
       RETURNS boolean
{
   not(empty(
      for $isLastObj in $in_isLast,
```

```
            $voc in $in_tvObj[@tvOID=$isLastObj/ /TemporalVersion/@vocOID]
        where $voc//VersionedObjectControl/lastVersion/Value
                    [@vTimei <= $in_instant and @vTimef >= $in_instant and
                     @tTimef = "9999-12-31"] =
              $isLastObj/@tvOID
        return $isLastObj))
}
```

1.22.8 tv:predecessorOf e tv:isPredecessorOf

A função `tv:predecessorOf` obtém a versão antecessora àquela passada pelo parâmetro `in_version`. O valor do `tvOID` da versão antecessora é atribuído à variável `version`. Então, o identificador é comparado com o `tvOID` de cada um dos elementos armazenados na variável `predVersion`. No momento em que os identificadores forem iguais, será retornado o elemento em `predVersion`, que representa a versão predecessora.

```
define FUNCTION tv:predecessorOf ($in_version, $in_tvObj)
       RETURNS predecessorVersion
{
   let $version := $in_version//TemporalVersion/predecessor/Value
   for $predVersion in $in_tvObj
   where $predVersion/@tvOID = $version
   return $predVersion
}
```

A função `tv:isPredecessorOf` examina se a versão `in_predTvObj` é antecessora à versão `in_tvObj`. Sua implementação aparenta ser idêntica à anterior. Porém, na atual função, a varredura do elemento `predecessor` é feita sobre o conjunto de versões (segundo parâmetro) e não sobre a única versão indicada pelo primeiro parâmetro. Já, na função `tv:predecessorOf`, a varredura é feita sobre a versão cujo predecessor é requerido (primeiro parâmetro).

```
define FUNCTION tv:isPredecessorOf($in_predTvObj, $in_tvObj)
       RETURNS boolean
{
   not(empty(for $tvObj in $in_tvObj//TemporalVersion/predecessor/Value,
                 $predTvObj in $in_predTvObj
             where $predTvObj/@tvOID = $tvObj
             return $predTvObj))
}
```

1.22.9 tv:successorOf, tv:isSuccessorOf e tv:isSuccessorOfAt

Operações considerando versões sucessoras dentro da hierarquia de derivação são realizadas a partir da utilização das funções `tv:successorOf`, `tv:isSuccessorOf` e `tv:isSuccessorOfAt`. A primeira função retorna os elementos que representam as versões sucessoras; as outras estabelecem comparações entre grupos de versões, verificando se existe relação de sucessão entre as versões.

```
define FUNCTION tv:successorOf ($in_version, $in_tvObj, $in_searchType)
       RETURNS successorVersion
{
   let $version := $in_version//TemporalVersion/successor/Value
```

```
   for $sucVersion in $in_tvObj
   where if ($in_searchType = "EVER") then
            $sucVersion/@tvOID = $version
         else
            $sucVersion/@tvOID = $version[@vTimei <= date() and
                                          @vTimef >= date() and
                                          @tTimef = "9999-12-31"]
   return $sucVersion
}

define FUNCTION tv:isSuccessorOf($in_succTvObj, $in_tvObj,
                                 $in_searchType)
RETURNS boolean
{
   not(empty(for $tvObj in $in_tvObj//TemporalVersion/successor/Value,
                 $succObj in $in_succTvObj
             where if ($in_searchType = "EVER") then
                      $tvObj = $succObj/@tvOID
                   else
                      $tvObj[@vTimei <= date() and @vTimef >= date() and
                             @tTimef = "9999-12-31"] = $succObj/@tvOID
             return $succObj))
}
```

Salientando pequenas alterações nas expressões de caminho, as funções `tv:successorOf` e `tv:isSuccessorOf` são respectivamente iguais às funções `tv:predecessorOf` e `tv:isPredecessorOf`. Entretanto, no contexto da relação predecessor-sucessor, a função `tv:isSuccessorOfAt` possui um caráter distinto. Ela tem seu retorno associado a um instante, o que não é visto nas funções que trabalham com predecessores. Tal fato se justifica porque uma versão sempre possui uma, e única, antecessora. Em qualquer instante em que uma função for aplicada, o resultado sempre será o mesmo, visto que o atributo de controle, e elemento, `predecessor` não é temporal.

```
define FUNCTION tv:isSuccessorOfAt($in_succTvObj, $in_tvObj,
                                   $in_instant)
       RETURNS boolean
{
   not(empty(
      for $tvObj in $in_tvObj//TemporalVersion/successor/Value
                  [@vTimei <= $in_instant and @vTimef >= $in_instant and
                   @tTimef = "9999-12-31"],
          $succObj in $in_succTvObj
      where $tvObj = $succObj/@tvOID
      return $succObj))
}
```

1.22.10tv:currentVersion, tv:isCurrent, tv:isCurrentAt

A função `tv:currentVersion` permite recuperar a versão corrente do objeto versionado indicado através do parâmetro `in_currObj`. Em `in_tvObj`, é indispensável que os controles de objeto versionado sejam informados para atingir o controle referente ao objeto e, por conseqüência, a versão corrente do objeto.

```
define FUNCTION tv:currentVersion ($in_currObj, $in_tvObj,
```

```
                                        $in_searchType)
        RETURNS currentVersion
{
   for $voc in $in_tvObj[@tvOID=$in_currObj//TemporalVersion/@vocOID],
       $currentVersion in $in_tvObj
   where if ($in_searchType = "EVER") then
            $currentVersion/@tvOID =
            $voc//VersionedObjectControl/currentVersion/Value
         else
            $currentVersion/@tvOID =
            $voc//VersionedObjectControl/currentVersion/Value
                  [@vTimei <= date() and @vTimef >= date() and
                   @tTimef = "9999-12-31"]
   return $currentVersion
}
```

A função `tv:isCurrent` constata, ou não, que a versão passada através do parâmetro `in_isCurrent` se trata da versão corrente. Os controles de objeto versionado e o escopo temporal da função são esperados para a execução da função.

```
define FUNCTION tv:isCurrent($in_isCurrent, $in_tvObj, $in_searchType)
       RETURNS boolean
{
   not(empty(
      for $isCurrObj in $in_isCurrent,
          $voc in $in_tvObj[@tvOID=$isCurrObj//TemporalVersion/@vocOID]
      where if ($in_searchType = "EVER") then
               $voc//VersionedObjectControl/currentVersion/Value =
               $isCurrObj/@tvOID
            else
               $voc//VersionedObjectControl/currentVersion/Value
                     [@vTimei <= date() and @vTimef >= date() and
                      @tTimef = "9999-12-31"] = $isCurrObj/@tvOID
      return $isCurrObj))
}
```

Assim como a anterior, a função `tv:isCurrentAt` retorna um valor booleano que é alcançado através da comparação entre o atributo de controle `currentVersion` e a versão passada à função. A diferença está na explicitação do instante no qual a versão deveria ser a corrente dentro do seu objeto versionado. O instante em questão é transmitido à função pelo parâmetro `in_instant`.

```
define FUNCTION tv:isCurrentAt($in_isCurrent, $in_tvObj, $in_instant)
       RETURNS boolean
{
   not(empty(
      for $isCurrObj in $in_isCurrent,
          $voc in $in_tvObj[@tvOID=$isCurrObj//TemporalVersion/@vocOID]
      where $voc//VersionedObjectControl/currentVersion/Value
                  [@vTimei <= $in_instant and @vTimef >= $in_instant and
                   @tTimef = "9999-12-31"] = $isCurrObj/@tvOID
      return $isCurrentObj))
}
```

1.22.11tv:isConfiguration

Um objeto pode possuir diferentes configurações, sendo uma versão apenas a cada nível da hierarquia de herança. Para verificar se uma versão é componente de uma configuração, é definida e implementada a função `tv:isConfiguration`.

```
define FUNCTION tv:isConfiguration($in_isConfig) RETURNS boolean
{
   not(empty(
      for $isConfigObj in $in_isConfig
      where $isConfigObj//TemporalVersion/configuration/Value = "true"
      return $isConfigObj))
}
```

Através do parâmetro `in_isConfig` é informada a versão cujo atributo de controle `configuration` é analisado. Se o valor for `true`, a versão é integrante de uma configuração. Nessa função, despreza-se o uso do parâmetro `in_tvObj`, pois não é preciso realizar acesso a qualquer outra versão, objeto ou controle para obter o resultado esperado para a função.

1.23 Considerações Finais

Nesta seção foram apresentadas as funções que caracterizam a extensão da linguagem de consulta. Considerando documentos modelados a partir do paradigma de representação descrito nesta pesquisa, as funções proporcionam a recuperação de informações relacionadas com tempo e versão, de forma que o usuário não precise ter um conhecimento elevado acerca da estrutura da visão XML que está consultando. Se o usuário possui uma razoável noção sobre o funcionamento das cláusulas `FLWR` da XQuery, um bom conhecimento das características de aplicações modeladas a partir do TVM e das funcionalidades proporcionadas pela TVQL, ele encontrará poucas dificuldades em recuperar informações presentes na representação.

Com as funções apresentadas neste trabalho, não é necessário, sequer, que o usuário tenha conhecimento da linguagem XPath para que consulte e recupere dados. A simples combinação de funções temporais com funções de versão permite atingir atributos de aplicação e atributos de controle sem grandes problemas. No entanto, se o usuário detém um bom conhecimento de expressões de caminho, funções e *wildcards* da XPath, ele poderá refinar ainda mais o processo de obtenção de informação, sem considerar a possibilidade de construção de saídas mais elaboradas através da cláusula `RETURN`.

Apesar da facilidade de definição de consultas com o uso das funções auxiliares, temporais e de versão, a extensão proposta apresenta significativas limitações. Na versão utilizada, a XQuery apresenta um suporte limitado a tipos temporais, sendo que apenas datas possuem um tratamento aceitável. Logo, como pôde ser visto nas implementações das funções, o suporte prestado na atual extensão se dá apenas para datas e na notação `aaaa-mm-dd`. Ainda, sobre a especificação da XQuery está prevista validação para os parâmetros de entrada de funções. Contudo, na prática isso ainda não está funcionando adequadamente. Finalizando, o tratamento de erros de execução das funções é inexistente, pois a XQuery não apresenta recursos comuns em linguagens de programação para o tratamento de exceções. No atual estágio da pesquisa, qualquer erro relacionado às novas funções resulta em um retorno vazio para a consulta.

7 Exemplos de Consultas

O objetivo deste capítulo é apresentar, através de consultas, alguns exemplos de utilização das funções propostas para a extensão da XQuery. Idealiza-se mostrar como as novas funções podem ser aplicadas sem que haja modificações acentuadas na forma padrão de utilização da linguagem. Além disso, deseja-se mostrar que o emprego das novas funções facilita a obtenção de informações relacionadas com tempo e com versão para documentos que seguem a proposta de representação vista neste trabalho.

Os exemplos destacados neste capítulo se baseiam em uma pequena aplicação, cujos dados estão representados em um documento XML que segue o modelo de representação anteriormente descrito. As consultas são realizadas sobre o documento XML, com o emprego de uma ferramenta que implementa a especificação da XQuery e suporta a adição de novas funções. Ainda, é proposto um ambiente de utilização da XQuery estendida, considerando documentos com características de tempo e de versão presentes em diretórios do sistema de arquivos ou gerenciados por um banco de dados.

1.24 Aplicação

O exemplo de aplicação escolhido para este trabalho não tem como objetivo destacar as várias alternativas de modelagem previstas pelo Modelo Temporal de Versões, mas deseja disponibilizar uma quantidade razoável de dados representados em um documento XML, que possam ser consultados pela XQuery e pelas novas funções de tempo e de versão propostas nesta pesquisa.

A aplicação consiste no controle de configurações de computadores, mais especificamente, *notebooks*. Na classe Computador, são definidas as propriedades processador, disco rígido e valor, que são genéricas para qualquer tipo de máquina (*desktop* ou portátil). As três propriedades são temporais e representam valores numéricos. Já a classe Notebook define dois atributos temporais, duração da bateria (numérico) e dispositivo apontador (caractere).

A distribuição das propriedades em duas classes permite que variantes de configurações de computador sejam criadas aproveitando versões já definidas dentro da hierarquia. A temporalização dos atributos permite manter todas as informações referentes a cada modelo de computador e/ou de *notebook* sempre acessíveis através de consultas temporais.

A **FIGURA 7.1** representa dois modelos de computadores e dois modelos específicos de *notebook*. Cada um desses modelos passou por alterações em suas configurações que foram mantidas pela temporalidade das propriedades. Mudanças com um grau de relevância alto originavam novas versões dos modelos, sendo que a correspondência entre essas versões também era mantida a cada nova derivação.

Nessa figura está representado apenas o estado atual de cada uma das versões no momento em que elas foram criadas. Ressalta-se que os dados dessa aplicação seguem o modelo de representação definido nesta pesquisa. Todo o histórico das propriedades, versão a versão, está descrito no

Anexo 4 Instâncias do TVM representadas em XML. Esses dados servirão de base para as consultas a serem discutidas na próxima seção.

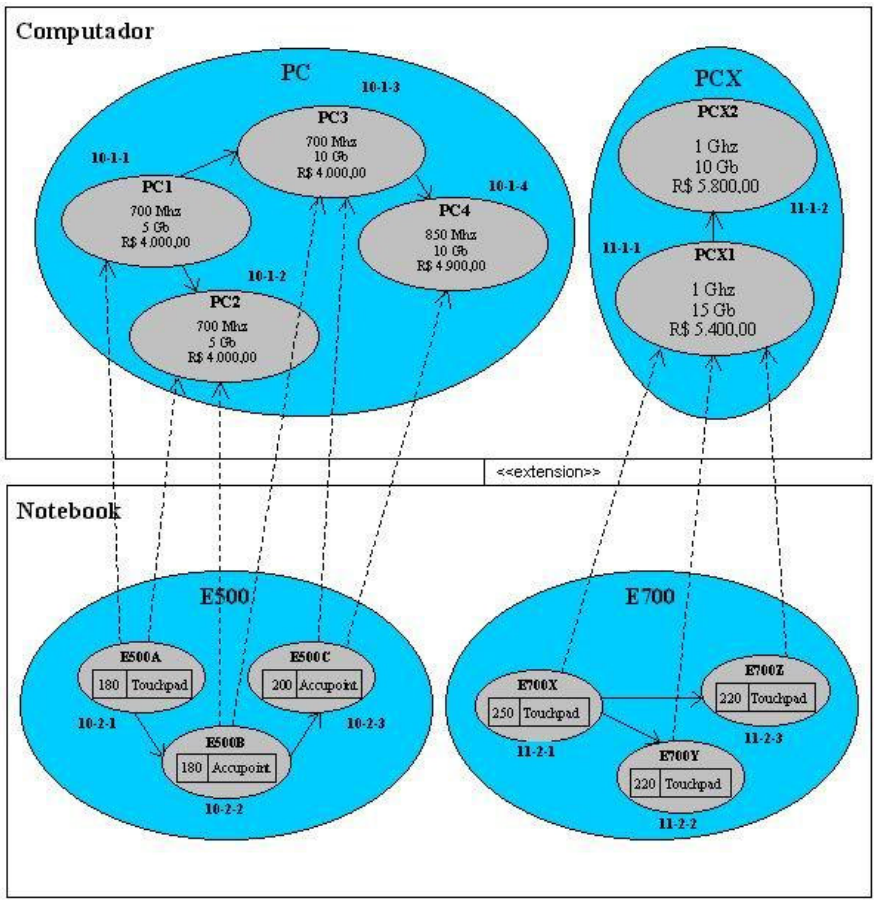

FIGURA 7.1 – Representação gráficas dos objetos e das versões

1.25 Consultas e Resultados

Nesta seção, são discutidas quinze consultas mostrando algumas das possíveis combinações de uso das funções definidas na extensão. Cada consulta possui: um enunciado definindo o objetivo a ser alcançando, a sintaxe de uma possível solução para o problema apresentado no enunciado, o resultado da consulta e a análise da solução encontrada.

1.25.1 Consulta A – tv:document

Recuperar todos os objetos versionados, versões e controles de objeto versionado presentes no documento `x-tv-instances`.

```
let $tvObj := tv:document("x-tv-instances.xml")
return $tvObj
```

Resultado:

```
<?xml version="1.0"?>
<xql:result xmlns:xql="http://metalab.unc.edu/xql/">
  <Object tvOID="10,1,0" name="PC" class="Computador"
          type="VersionedObject">
    <TemporalObject>
      ...
  </Object>
  ...
  <Object tvOID="11,1,2" name="PCX2" class="Computador" type="Version">
    <TemporalObject>
      ...
      <Attributes>
        <processador>
          <Value tTimei="2001-10-01" tTimef="9999-12-31"
                 vTimei="2001-10-01" vTimef="9999-12-31">1000</Value>
        </processador>
        ...
  </Object>
  ...
  <Object tvOID="35,3,1" class="VersionedObjectControl">
    <TemporalObject>
      ...
      <VersionedObjectControl>
        <configurationCount>
          <Value tTimei="2001-06-21" tTimef="9999-12-31"
                 vTimei="2001-06-21" vTimef="9999-12-31">0</Value>
        </configurationCount>
        <currentVersion>
          <Value tTimei="2001-10-01" tTimef="9999-12-31"
                 vTimei="2001-10-01" vTimef="9999-12-31">11,1,2</Value>
        </currentVersion>
        ...
  </Object>
  ...
</xql:result>
```

Uma rápida comparação com o documento `x-tv-instances` apresentado no

Anexo 4 Instâncias do TVM representadas em XML permite constatar que o resultado descrito não é apresentado em sua totalidade. O principal objetivo dessa abreviação é demonstrar a saída da função `tv:document`. Ela recebe o documento como parâmetro e recupera todos elementos `Object` que representam versões, objetos versionados e controles de objeto versionado.

Convenciona-se utilizar essa função na cláusula `LET` da XQuery, pois em todas consultas com aspectos de tempo e de versão são utilizados os objetos recuperados pela função `tv:document`. Porém, poucas vezes se realiza algum tipo de iteração sobre a variável que recebe os valores retornados da função. As iterações são realizadas em variáveis derivadas da inicial, que são definidas, posteriormente, na cláusula `FOR`.

1.25.2 Consulta B – tv:nickname

Recuperar a versão cujo nome ou nickname corresponde a `PCX2`.

```
let $tvObj := tv:document("x-tv-instances.xml")
return tv:nickname("PCX2", $tvObj)
```

Resultado:

```
<?xml version="1.0"?>
<Object tvOID="11,1,2" name="PCX2" class="Computador" type="Version">
  <TemporalObject>
    <alive>
      <Value tTimei="2001-10-01" tTimef="9999-12-31"
             vTimei="2001-10-01" vTimef="9999-12-31">true</Value>
    </alive>
    <TemporalVersion vocOID="35,3,1">
      <ascendant>
        <Value tTimei="2001-10-01" tTimef="9999-12-31"
               vTimei="2001-10-01" vTimef="9999-12-31">null</Value>
      </ascendant>
      <descendant>
        <Value tTimei="2001-10-01" tTimef="9999-12-31"
               vTimei="2001-10-01" vTimef="9999-12-31">null</Value>
      </descendant>
      ...
    </TemporalVersion>
    <Attributes>
      <processador>
        <Value tTimei="2001-10-01" tTimef="9999-12-31"
               vTimei="2001-10-01" vTimef="9999-12-31">1000</Value>
      </processador>
      <discoRigido>
        <Value tTimei="2001-10-01" tTimef="9999-12-31"
               vTimei="2001-10-01" vTimef="9999-12-31">10</Value>
      </discoRigido>
      <valor>
        <Value tTimei="2001-10-01" tTimef="9999-12-31"
               vTimei="2001-10-01" vTimef="9999-12-31">5800.00</Value>
      </valor>
    </Attributes>
  </TemporalObject>
</Object>
```

A consulta B exemplifica a utilização da função auxiliar `tv:nickname` na cláusula `RETURN`. Normalmente, essa função é utilizada em condições na cláusula `WHERE` e na atribuição de valores a variáveis na cláusula `FOR`. Entretanto, como o resultado

desejado é a própria versão retornada da função, seu emprego é totalmente justificado na cláusula `RETURN`, sendo desnecessária a definição de qualquer variável auxiliar para obter o resultado esperado (foram omitidos certos elementos que representam atributos de controle, pois o resultado real é relativamente extenso).

1.25.3 Consulta C – tv:objects, tv:property e tv:value

Retornar, a partir dos objetos versionados da classe `Computador`, os valores das propriedades `processador` e `valor`.

```
let $tvObj := tv:document("x-tv-instances.xml")
for $versObjs in tv:objects("Computador", $tvObj)
return <ObjetoVersionado tvOID={$versObjs/@tvOID}>
          {tv:property("processador", $versObjs, "PRESENT")}
          {tv:value(tv:property("valor", $versObjs, "PRESENT"))}
       </ObjetoVersionado>
```

Resultado:

```
<?xml version="1.0"?>
<xql:result xmlns:xql="http://metalab.unc.edu/xql/">
  <ObjetoVersionado tvOID="10,1,0">
    <Value tTimei="2001-06-21" tTimef="9999-12-31"
           vTimei="2001-06-23" vTimef="9999-12-31">900</Value>
    <Value>4200.00</Value>
  </ObjetoVersionado>
  <ObjetoVersionado tvOID="11,1,0">
    <Value tTimei="2001-10-01" tTimef="9999-12-31"
           vTimei="2001-10-01" vTimef="9999-12-31">1000</Value>
    <Value>5800.00</Value>
  </ObjetoVersionado>
</xql:result>
```

Para recuperar os objetos versionados de uma determinada classe, é aplicada a função `tv:objects`. Os objetos retornados pela função são atribuídos à variável `versObjs`, que servirá de entrada para as funções que recuperarão os valores das propriedades. Na cláusula `RETURN` é criado o elemento denominado `ObjetoVersionado` para agrupar as propriedades de acordo com o seu respectivo objeto versionado. Nesse elemento é definido um atributo XML de nome `tvOID`, que identifica cada um dos objetos versionados originários da variável `versObjs`. Sob o novo elemento são listados o valor e os rótulos temporais da propriedade `processador` (resultado da aplicação da função `tv:property`) e o conteúdo da propriedade `valor`. Quando a função `tv:value` é combinada com a `tv:property`, os valores retornados não são associados aos rótulos temporais, o que pode ser visto no resultado da consulta C.

1.25.4 Consulta D – tv:versions, tv:tiInstant, tv:tfInstant e tv:viInstant

Obter o nome da versão e os tempos de transação da propriedade `duracaoBateria` quando seu valor foi igual a `180 min`. O valor requerido precisa ter começado a valer em `2001-08-10`.

```
let $tvObj := tv:document("x-tv-instances.xml")
for $versObjs in tv:versions(tv:objects("Notebook", $tvObj), $tvObj),
    $propDurBat in tv:property("duracaoBateria",$versObjs, "EVER")
where tv:viInstant($propDurBat) > "2001-08-10" and $propDurBat = 180 and
```

```
      tv:tfInstant($propDurBat) = "9999-12-31"
return <TempoDeTransacao versao={$versObjs/@name}>
          {tv:tiInstant($propDurBat)}
          {tv:tfInstant($propDurBat)}
       </TempoDeTransacao>
```

Resultado:

```
<?xml version="1.0"?>
<TempoDeTransacao versao="E700Y">
  <tiInstant>2001-09-09</tiInstant>
  <tfInstant>9999-12-31</tfInstant>
</TempoDeTransacao>
```

Na variável `versObjs` são armazenadas as versões dos objetos instanciados da classe `Notebook` (`duracaoBateria` é uma propriedade da classe `Notebook`). A variável `propDurBat` recebe os valores referentes à propriedade `duracaoBateria`, que são recuperados pela função `tv:property`. Essa função tem como entrada o conjunto de versões presentes na variável `versObjs`. Como o enunciado está considerando uma data passada, a função deve utilizar o parâmetro `EVER` para direcionar o processamento da função visando tratar todo o histórico de valores da propriedade.

Na cláusula `WHERE` é utilizada a função `tv:viInstant` para recuperar o tempo de validade inicial dos valores contidos em `propDurBat`. Esses instantes são comparados com a data `2001-08-10` e o conteúdo de `propDurBat`, comparado com o valor `180`. A condição `tv:tfInstant($propDurBat) = "9999-12-31"` implica a análise do histórico do estado atual da representação (base de dados). Se a condição for verdadeira, será construído o elemento `TempoDeTransacao` com um atributo indicando o nome da versão. O conteúdo do elemento é resultado da aplicação das funções `tv:tiInstant` e `tv:tf:Instant`, que retornam os tempos de transação da propriedade.

1.25.5 Consulta E – tv:iLifeTime, tv:fLifeTime e tv:now

Retornar as configurações de disco rígido com capacidade superior a 8 Gb identificando o computador a que pertencem. O modelo de computador ainda deve ser válido e a data inicial deve ser superior a 2001-07-05.

```
let $tvObj := tv:document("x-tv-instances.xml")
for $versObjs in tv:versions(tv:objects("Computador", $tvObj), $tvObj),
    $propDiscRig in tv:property("discoRigido",$versObjs, "EVER")
where tv:iLifeTime($versObjs) > "2001-07-05" and
      tv:fLifeTime($versObjs) >= tv:now() and $propDiscRig > 8 and
      tv:tfInstant($propDiscRig) = "9999-12-31"
return <DiscoRigido Gb={$propDiscRig} computador={$versObjs/@name} />
```

Resultado:

```
<?xml version="1.0"?>
<xql:result xmlns:xql="http://metalab.unc.edu/xql/">
  <DiscoRigido Gb="10" computador="PC4" />
  <DiscoRigido Gb="15" computador="PCX1" />
  <DiscoRigido Gb="10" computador="PCX1" />
  <DiscoRigido Gb="10" computador="PCX2" />
</xql:result>
```

As versões de `Computador` são atribuídas a `versObjs` e os valores da propriedade `discoRigido` à variável `propDiscRig`. Como se deseja recuperar os discos de computadores ainda válidos, o valor do tempo de vida da versão deve ser igual ou superior à data atual. A data atual é recuperada através da função `tv:now` e o tempo final de vida da versão, através da função `tv:fLifeTime` (recebe cada versão contida em `versObjs`). O tempo inicial de vida da versão utilizada na comparação é retornado pela função `tv:iLifeTime`. Concluindo a condição imposta na cláusula `WHERE`, os valores presentes na variável `propDiscRig` devem ser superiores a 8 Gb, seguindo o solicitado no enunciado. Como resultado da consulta, são apresentados elementos vazios de nome `DiscoRigido` que possuem dois atributos: `computador`, indicando o modelo da máquina, e `Gb`, listando o valor da propriedade `discoRigido`.

1.25.6 Consulta F – tv:tInterval, tv:vinterval e tv:equal

Retornar o dispositivo apontador juntamente com o intervalo de validade daqueles cuja transação ocorreu exatamente no intervalo `2001-05-05..2001-06-04`.

```
let $tvObj := tv:document("x-tv-instances.xml")
for $versObjs in tv:versions(tv:objects("Notebook", $tvObj), $tvObj),
    $propDispAp in tv:property("dispositivoApontador",$versObjs, "EVER")
where tv:equal(tv:tInterval($propDispAp), "2001-05-05..2001-06-07")
return <Dispositivo>
         {tv:value($propDispAp)}
         {tv:vInterval($propDispAp)}
       </Dispositivo>
```

Resultado:

```
<?xml version="1.0"?>
<Dispositivo>
  <Value>Touchpad</Value>
  <vInterval>2001-05-05..</vInterval>
</Dispositivo>
```

Para verificar a ocorrência de igualdade entre intervalos, é utilizada a função `tv:equal`. No exemplo, o intervalo "`2001-05-05..2001-06-04`" e os intervalos de transação dos valores da propriedade `dispositivoApontador` (`propDispAp`) recuperados pela função `tv:tInterval` são comparados. Nesse tipo de consulta, em que todo o histórico da representação é verificado, despreza-se a condição `tv:tfInstant($propDurBat) = "9999-12-31"`. Assim, qualquer informação que foi inserida será efetivamente analisada, não só aquelas referentes ao estado atual da base.

O valor do dispositivo apontador e o seu respectivo intervalo de validade são retornados no elemento `Dispositivo` construído na cláusula `RETURN`. O intervalo em questão é obtido e encapsulado em um elemento `vInterval` através da função `tv:vInterval`.

1.25.7 Consulta G – tv:intersect e tv:overlap

Obter as propriedades `processador` e `valor` cujos intervalos de validade se intersectam e que o intervalo de validade da propriedade `valor` sobreponha completamente o intervalo "`2001-07-01..2002-07-01`".

```
let $tvObj := tv:document("x-tv-instances.xml")
```

```
for $versObjs in tv:versions(tv:objects("Computador", $tvObj), $tvObj),
    $propPrc in tv:property("processador",$versObjs, "PRESENT"),
    $propVlr in tv:property("valor",$versObjs, "PRESENT")
where tv:intersect(tv:vInterval($propPrc), tv:vInterval($propVlr)) and
      tv:overlap(tv:vInterval($propVlr), "2001-07-01..2002-07-01")
return <Propriedades>
         {tv:value($propPrc)}{tv:value($propVlr)}
       </Propriedades>
```

Resultado:

```
<?xml version="1.0"?>
<xql:result xmlns:xql="http://metalab.unc.edu/xql/">
  <Propriedades>
    <Value>700</Value>
    <Value>4000.00</Value>
  </Propriedades>
  <Propriedades>
    <Value>900</Value>
    <Value>4200.00</Value>
  </Propriedades>
</xql:result>
```

Como a consulta busca recuperar duas propriedades, seus valores atuais devem ser atribuídos a duas variáveis de iteração (`propPrc` e `propVlr`). Para verificar se os tempos de validade das duas propriedades intersectam, são aplicadas as funções `tv:intersect` e `tv:vInterval` de forma combinada. A função `tv:overlap` é utilizada para verificar se o intervalo de validade da propriedade `valor` sobrepõe o intervalo "`2001-07-01..2002-07-01`". O resultado da consulta é apresentado em pares de elementos `Value` agrupados sob o elemento `Propriedades`. Cada ocorrência do elemento `Propriedades` caracteriza os valores das propriedades `processador` e `valor` de uma única versão.

1.25.8 Consulta H – tv:into, tv:before e tv:after

Retornar o tvOID e o nome da versão de computador cuja configuração de disco rígido tenha valido em junho de 2001 e que o tempo inicial de vida da versão seja anterior ao intervalo "`2001-07-10..`" e o tempo final posterior ao mês de dezembro de 2001.

```
let $tvObj := tv:document("x-tv-instances.xml")
for $versObjs in tv:versions(tv:objects("Computador", $tvObj), $tvObj),
    $propDiscRig in tv:property("discoRigido",$versObjs, "EVER")
where tv:into(tv:vInterval($propDiscRig), "2001-06-01..2001-06-30") and
      tv:before(tv:iLifeTime($versObjs), "2001-07-10..") and
      tv:after(tv:fLifeTime($versObjs), "2001-12-01..2001-12-31") and
      tv:tfInstant($propDiscRig) = "9999-12-31"
return <Computador>
         <tvOID>{$versObjs/@tvOID}</tvOID>
         <nome>{$versObjs/@name}</nome>
       </Computador>
```

Resultado:

```
<?xml version="1.0"?>
<Computador>
```

```
  <tvOID tvOID="10,1,3" />
  <nome name="PC3" />
</Computador>
```

Para verificar se um disco rígido valeu no mês de junho, utilizam-se a função `tv:vInterval` para recuperar o intervalo de validade da propriedade `discoRigido` e a função `tv:into` para examinar se esse intervalo de validade está dentro do intervalo "`2001-06-01..2001-06-30`", que representa o mês de junho. Nessa consulta, as funções `tv:before` e `tv:after`, respectivamente, analisam se os tempos de vida da versão a ser recuperada estão antes e após aos intervalos propostos. Como saída da consulta, um elemento `Computador` é criado com os subelementos que representam o `tvOID` e o nome da versão retornada.

1.25.9 Consulta I – tv:isStable, tv:isDeactivated e tv:isWorkingAt

Retornar uma listagem dos tvOID das versões que nunca estiveram desativadas, que estiveram em trabalho no dia 25 de agosto de 2001 e que se encontram estáveis atualmente.

```
let $tvObj := tv:document("x-tv-instances.xml"),
    $versComp := tv:versions(tv:objects("Computador", $tvObj), $tvObj),
    $versNote := tv:versions(tv:objects("Notebook", $tvObj), $tvObj)
for $versions in $versComp union $versNote
where not(tv:isDeactivated($versions, "EVER")) and
      tv:isWorkingAt($versions, "2001-08-25") and
      tv:isStable($versions, "PRESENT")
return $versions/@tvOID
```

Resultado:

```
<xql:result xmlns:xql="http://metalab.unc.edu/xql/">
  11,1,1
</xql:result>
```

Como a consulta deseja examinar todas as versões, independentemente do objeto de que fazem parte, foi utilizada a função da XQuery denominada `union` para agrupar esses dois conjuntos de versões. Na função `tv:isDeactivated` é utilizado o parâmetro `EVER` porque todo o histórico do atributo de controle `status` é verificado, diferentemente da função `tv:isStable`, que considera apenas o valor atual. Já a função `tv:isWorkingAt` independe desse parâmetro, pois recebe um instante que define o escopo de atuação da função. O retorno da consulta é uma simples listagem de `tvOID` (nesse caso, `11,1,1`) que segue a construção-padrão de elementos definida pelo protótipo.

1.25.10Consulta J – tv:isConsolidated e tv:descendantOf

Recuperar o tempo inicial de vida dos descendentes das versões de computadores que atualmente estão consolidadas.

```
let $tvObj := tv:document("x-tv-instances.xml")
for $versComp in tv:versions(tv:objects("Computador", $tvObj), $tvObj)
where tv:isConsolidated($versComp, "PRESENT")
return
   for $versDesc in tv:descendantOf($versComp, $tvObj, "PRESENT")
```

```
   return <resultado>
            <versao>{$versComp/@name}</versao>
            <descendente>{$versDesc/@name}</descendente>
            {tv:iLifeTime($versDesc)}
          </resultado>
```

Resultado:

```
<?xml version="1.0"?>
<resultado>
  <versao name="PC4" />
  <descendente name="E500C" />
  <iLifeTime>2001-07-02</iLifeTime>
</resultado>
```

Na primeira etapa da consulta é aplicada a função `tv:isConsolidated` sobre o conjunto de versões de computadores. Só servem de entrada para a consulta aninhada aquelas versões cujo *status* atual seja a `Consolidated`. Na segunda etapa, a variável `versDesc` recebe as atuais versões descendentes de `versComp`. O resultado da consulta é construído com o elemento `versão` (valor recuperado na primeira etapa) e os elementos `descendente` e `iLifeTime` (valores referentes, nesse exemplo, a versão descendente recuperada na segunda fase).

1.25.11Consulta K – tv:ascendantOf, tv:isFirst e tv:isLastAt

Retornar o histórico da propriedade `discoRigido` da versão de computador que é a primeira e que foi ou é ascendente da versão do objeto versionado `E700` que já foi a última em 20 de setembro de 2001.

```
let $tvObj := tv:document("x-tv-instances.xml")
for $versNote in tv:versions(tv:nickname("E700", $tvObj), $tvObj),
    $versComp in tv:versions(tv:objects("Computador", $tvObj), $tvObj),
    $propDiscRig in tv:property("discoRigido", $versComp, "EVER")
where tv:isLastAt($versNote, $tvObj, "2001-09-20") and
      tv:isAscendantOf($versComp, $versNote, "EVER") and
      tv:tfInstant($versNote//ascendant/Value) = "9999-12-31" and
      tv:isFirst($versComp, $tvObj, "PRESENT") and
      tv:tfInstant($propDiscRig) = "9999-12-31"
return tv:value($propDiscRig)
```

Resultado:

```
<xql:result xmlns:xql="http://metalab.unc.edu/xql/">
  <Value>15</Value>
  <Value>5</Value>
  <Value>10</Value>
</xql:result>
```

A variável `versNote` recebe o conjunto de versões do objeto `E700`, e a variável `versComp`, todas as versões dos objetos da classe `Computador`. Do grupo de versões representadas em `versComp` são obtidos os valores da propriedade `discoRigido`. Esses valores são atribuídos à variável `propDiscRig`, que é utilizada tanto na cláusula `WHERE` quanto na `RETURN`. Na cláusula `WHERE`, a função `tv:isLastAt` analisa se uma das versões do Notebook E700 foi a última em 20 de setembro de 2001. A função `tv:isAscendantOf` constata se os valores de `versComp` e de `versNote` estabelecem

uma relação de ascendência a cada iteração; já `tv:isFirst` examina se a versão presente em `versComp` é atualmente a primeira. As duas condições que utilizam a função `tv:tfInstant` são empregadas para recuperar os valores do estado atual da "base". O estado atual é verificado através do tempo de transação final de cada informação, que deve ser igual a `null` (na aplicação, `9999-12-31`).

1.25.12Consulta L – tv:lastVersion, tv:predecessorOf e tv:isConfiguration

Retornar o atual processador da versão antecessora a última do objeto `PC`, desde que esta não faca parte de uma configuração.

```
let $tvObj := tv:document("x-tv-instances.xml")
for $versLast in tv:lastVersion(tv:nickname("PC", $tvObj),
                                $tvObj,
                                "PRESENT"),
    $propProc in tv:property("processador",
                             tv:predecessorOf($versLast, $tvObj),
                             "PRESENT")
where not(tv:isConfiguration($versLast, $tvObj))
return <processador>{string($propProc)}</processador>
```

Resultado:

```
<?xml version="1.0"?>
<processador>850</processador>
```

Através da função `tv:lastVersion` aplicada na cláusula `FOR` é recuperada a atual última versão do objeto `PC`. O elemento que representa a última versão é atribuído à variável `versLast`, que é passada como parâmetro à função `tv:isConfiguration`. Essa função verifica se a versão faz parte de uma configuração. Voltando à cláusula `FOR`, a variável `propProc` recebe o valor atual da propriedade `processador`. Desse valor é retirada a estrutura do elemento e só é retornada a string com a velocidade do processador. A string retornada é conteúdo do elemento `processador` construído na cláusula `RETURN` para fins de apresentação do resultado.

1.25.13Consulta M – tv:successorOf e tv:isCurrentAt

Obter o nome das atuais versões sucessoras da versão do objeto `E500` que foi corrente em 30 de junho de 2001.

```
let $tvObj := tv:document("x-tv-instances.xml")
for $versNote in tv:versions(tv:nickname("E500", $tvObj), $tvObj)
where tv:isCurrentAt($versNote, $tvObj, "2001-06-30")
return
   for $versSucc in tv:successorOf($versNote, $tvObj, "PRESENT")
   return <sucessor>{string($versSucc/@name)}</sucessor>
```

Resultado:

```
<?xml version="1.0"?>
<sucessor>E500C</sucessor>
```

Inicialmente, as versões do objeto `E500` são atribuídas à variável `versNote`. Na primeira fase da consulta é examinado se alguma das versões presentes em `versNote`

foi a corrente em 30 de junho de 2001. Se essa condição for verdadeira, passa-se a processar a consulta aninhada. Nela, a variável `versSucc` recebe as atuais sucessoras da versão que satisfez a condição imposta na primeira etapa. No resultado, a string obtida da expressão de caminho `versSucc/@name` é definida como conteúdo do elemento `sucessor`.

1.25.14Consulta N – tv:into, tv:after e tv:descendantOf

Retornar o histórico da propriedade `dispositivoApontador` das versões descendentes daquelas cujo instante inicial de vida está definido no mês de agosto e o instante final é posterior ao mesmo mês.

```
let $tvObj := tv:document("x-tv-instances.xml")
for $versComp in tv:versions(tv:objects("Computador", $tvObj), $tvObj)
where tv:into(tv:iLifeTime($versComp), "2001-08-01..2001-08-31") and
      tv:after(tv:fLifeTime($versComp), "..2001-08-31")
return
   for $descComp in tv:descendantOf($versComp, $tvObj, "PRESENT"),
       $propDispAp in tv:property("dispositivoApontador",
                                  $descComp,
                                  "EVER")
   where tv:tfInstant($propDispAp) = "9999-12-31"
   return <Computador modelo={$versComp/@name}>
            <Notebook modelo={$descComp/@name}>
              <Dispositivo>
                {tv:value($propDispAp)}
              </Dispositivo>
            </Notebook>
          </Computador>
```

Resultado:

```
<?xml version="1.0"?>
<Computador modelo="PCX1">
  <Notebook modelo="E700Z">
    <Dispositivo>
      <Value>Touchpad</Value>
    </Dispositivo>
  </Notebook>
</Computador>
```

Em sua primeira parte, a consulta recupera as versões de objetos da classe `Computador`. Dessas, são apenas consideradas na segunda parte da consulta aquelas cujo tempo de vida inicial está dentro do intervalo `"2001-08-01..2001-08-31"` (`tv:into`), e o tempo de vida final é posterior ao intervalo, cujo tempo final é igual à data que marca o fim do mês de agosto (`tv:after`). Na segunda parte, os atuais descendentes das versões de `Computador` avaliadas na primeira etapa são armazenados na variável `descComp`. Dessa variável é recuperado o histórico do estado atual da propriedade `dispositivoApontador`, que, por sua vez, é apresentado como resultado da consulta. O resultado é construído em um elemento `Computador` que possui como subelemento o Notebook descendente. Sob o elemento `Notebook` é definido o elemento `Dispositivo`, que agrupa os valores do histórico de `dispositivoApontador`.

1.25.15Consulta O – tv:intersect, tv:firstVersion,

Obter, da primeira versão de todos os notebooks, os valores de todas suas propriedades no intervalo "2001-07-01..2001-12-31".

```
let $tvObj := tv:document("x-tv-instances.xml")
for $objVers in tv:objects("Notebook", $tvObj),
    $firstVers in tv:firstVersion($objVers, $tvObj, "PRESENT")
return
    <Notebook tvOID={$firstVers/@tvOID}>
      <DuracaoBateria>
        {for $propDurBat in tv:property("duracaoBateria",
                                        $firstVers,
                                        "EVER")
         where tv:tfInstant($propDurBat) = "9999-12-31" and
               tv:intersect(tv:vInterval($propDurBat),
                            "2001-07-01..2001-12-31")
         return tv:value($propDurBat)}
      </DuracaoBateria>
      <DispositivoApontador>
        {for $propDispAp in tv:property("dispositivoApontador",
                                        $firstVers,
                                        "EVER")
         where tv:tfInstant($propDispAp) = "9999-12-31" and
               tv:intersect(tv:vInterval($propDispAp),
                            "2001-07-01..2001-12-31")
         return tv:value($propDispAp)}
      </DispostivoApontador>
    </Notebook>
```

Resultado:

```
<?xml version="1.0"?>
<xql:result xmlns:xql="http://metalab.unc.edu/xql/">
  <Notebook tvOID="10,2,1">
    <DuracaoBateria>
      <Value>180</Value>
    </DuracaoBateria>
    <DispositivoApontador>
      <Value>Accupoint</Value>
    </DispositivoApontador>
  </Notebook>
  <Notebook tvOID="11,2,1">
    <DuracaoBateria>
      <Value>220</Value>
    </DuracaoBateria>
    <DispositivoApontador>
      <Value>Touchpad</Value>
    </DispositivoApontador>
  </Notebook>
</xql:result>
```

O passo inicial a ser dado para resolver essa consulta é recuperar as primeiras versões de `Notebook`. Em `objVers` são atribuídos os objetos versionados da classe `Notebook`. Essa variável é a principal entrada para a função `tv:firstVersion`, que visa recuperar as atuais primeiras versões de cada objeto. Passada essa primeira fase, já é iniciada a construção do resultado. Nesse exemplo, a construção é um pouco mais

elaborada pois existem duas subconsultas aninhadas à principal. Inicialmente, é definido o elemento `Notebook`, onde um atributo representa o `tvOID` da versão. Sob esse elemento são, respectivamente, definidos os subelementos `DuracaoBateria` e `DispositivoApontador`. Em cada um desses subelementos está presente uma consulta que recupera o histórico dos valores da respectiva propriedade. Como é de interesse retornar os valores no intervalo `"2001-07-01..2001-12-31"`, a função `tv:intersect` está presente em ambas subconsultas.

1.26 Ferramenta Utilizada nos Testes

Desde a definição da XQuery como a linguagem base para extensão, não foi estabelecida a implementação de um protótipo de processador/interpretador da linguagem como objetivo da pesquisa. Muitas ferramentas que implementam as várias especificações da XQuery foram e estão sendo criadas. Logo, optou-se por utilizar ferramentas disponibilizadas por empresas e por centros de pesquisa para validar as funções que neste trabalho foram definidas. Dentre algumas ferramentas analisadas, podem ser destacadas as seguintes:

- **Microsoft XML Query Language Demo:** Disponível em http://xmlqueryservices.com/. Esse protótipo é baseado na especificação de 7 de junho de 2001 (penúltima especificação da XQuery), porém inviabiliza qualquer tentativa de teste das novas funções visto que as consultas nele escritas só podem acessar documentos predefinidos;
- **Galax:** Disponível em: http://db.bell-labs.com/galax/. Protótipo desenvolvido em um projeto de mesmo nome coordenado pelos pesquisadores Mary Fernandez, Jérôme Siméon and Philip Wadler em Bell Labs. Suporta a maioria das especificações da XQuery, inclusive a atual, 20 de dezembro de 2001. Apenas realiza consultas referentes ao estudo de caso definido pelo W3C XML Query Working Group;
- **X-Hive XQuery Demo:** Disponível em: http://www.x-hive.com/xquery. Protótipo acessa documentos armazenados em um banco de dados XML X-Hive/DB povoado com documentos utilizados no estudo de caso do W3C. É baseado, também, na última especificação da XQuery e, igualmente aos protótipos anteriores, não permite consultas a novos documentos;
- **Software AG QuiP:** Disponível em http://www.softwareag.com/ developer/quip/. Protótipo desenvolvido para plataforma Windows 32 bit que disponibiliza uma interface gráfica para a escrita de consulta e visualização de resultados. Possui integração com o banco de dados XML Software AG Tamino e segue a especificação da XQuery de 7 de junho de 2001.

Dentre as ferramentas descritas, as três primeiras não permitem a definição de consultas sobre documentos criados pelo usuário. Logo, optou-se pela ferramenta Software AG Quip, a que permite que consultas sejam definidas sobre documentos variados. Além disso, permite a incorporação de novas funções, fato que viabiliza o teste das funções da extensão. A implementação de uma interface gráfica para a entrada e a saída das consultas também é um ponto relevante dessa implementação, pois torna ágil o processo de iteração com os recursos fornecidos.

Destaca-se que as consultas apresentadas e discutidas na seção anterior foram todas testadas utilizando a ferramenta Software AG Quip. Todas as funções propostas e implementadas foram integradas ao ambiente por meio de uma biblioteca seguindo os moldes por ela definidos. Os resultados dos exemplos de consultas são apresentados na íntegra, respeitando a saída disponibilizada pela ferramenta.

1.27 Uma Proposta de Ambiente para Aplicação da XQuery Estendida

O ambiente proposto de aplicação da linguagem XQuery com as novas funções (**FIGURA 7.2**) é composto por três partes: o sistema de arquivos (SA), o banco de dados (BD) e o motor de consulta da linguagem XQuery. A linguagem estendida deve trabalhar tanto sobre arquivos XML localizados em diretórios no sistema de arquivos quanto sobre documentos XML gerenciados pelo banco de dados, neste caso, o DB2.

O sistema de arquivos varia de acordo com o sistema operacional utilizado. No entanto, a função de manipular os arquivos armazenados em disco é comum para todos os sistemas. Basicamente, os arquivos armazenados em disco e controlados pelo SA são do tipo binário ou texto. Documentos XML armazenados em diretórios são exemplos de arquivos do tipo texto. Realizar consultas sobre arquivos dessa ordem (modo um) caracteriza-se como uma das quatro maneiras de utilização da XQuery com as novas funções. Esses arquivos XML podem representar qualquer tipo de informação, ou seja, não são gerados e utilizados por uma única aplicação. Mais do que isso, esses documentos podem ou não apresentar características de tempo e de versão. Independentemente desses fatores, a extensão de linguagem deve satisfazer às necessidades de consulta para cada caso.

Os documentos considerados no modo um não são controlados por uma aplicação em especial. No entanto, alguns deles podem estar sendo gerenciados pelo banco de dados, o que é possível através da integração de uma ferramenta ao banco de dados. A ferramenta em questão é a *XML Extender* [IBM 00], que faz parte de um conjunto de *extenders* projetados para trabalhar com tipos de dados (som, imagem, vídeo, XML, etc.) não adequadamente gerenciados pelos bancos de dados convencionais. Considerando o ambiente, essa possibilidade é apresentada no modo dois.

Ainda, o *XML Extender* permite que um documento seja integralmente armazenado numa coluna de uma tabela. Para isso, a ferramenta disponibiliza novos tipos de dados e funções de manipulação. Contudo, mesmo que o documento esteja numa coluna, é necessário que haja uma referência, partindo do BD, ao arquivo XML que está presente em algum diretório do SA. Para consultar esses documentos, é preciso criar tabelas auxiliares que representem aqueles elementos e atributos do documento armazenado na coluna. Isso se mostra importante, pois as tabelas auxiliares podem ser indexadas para melhorar a performance das consultas. Feito isso, consultas SQL podem

ser realizadas integrando, quando necessário, as tabelas auxiliares com a tabela base.

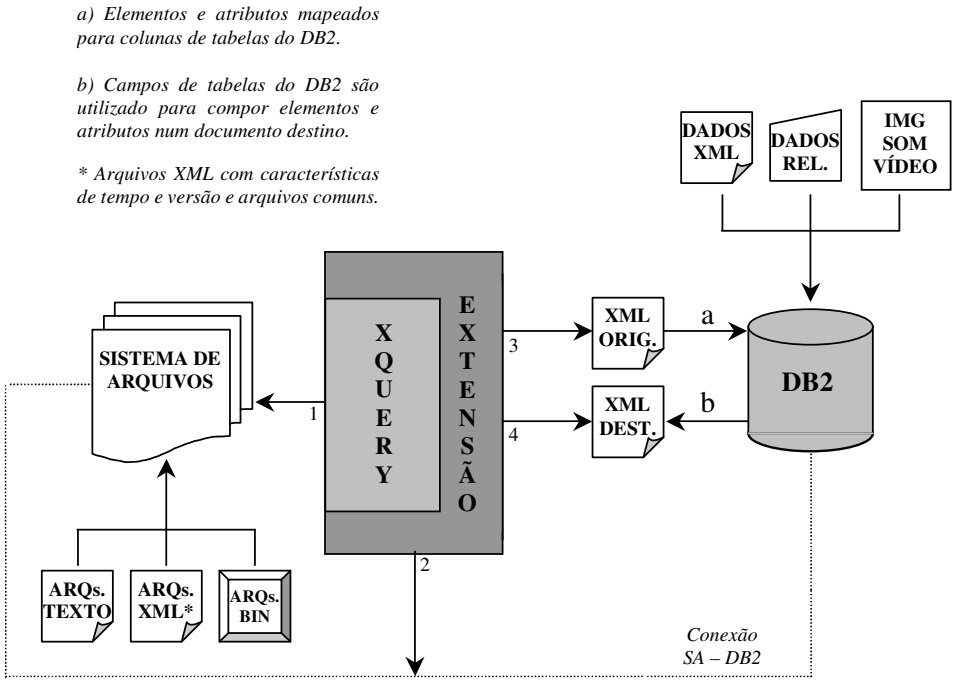

FIGURA 7.2 – Ambiente de utilização da XQuery com as novas funções

Por esse ponto de vista, não se justifica completamente o uso da XQuery, visto que os dados estão sendo controlados e podem ser consultados através de SQL. Entretanto, isso exige a criação de tabelas auxiliares e regras de mapeamento para qualquer documento que se deseja armazenar. Com a XQuery, nada impede que o documento continue sendo gerenciado pelo BD, todavia consultas podem ser aplicadas de forma direta ao documento, sem necessidade de definição de métodos complementares.

Da mesma forma que documentos XML padrão podem ser armazenados em colunas, documentos XML com características de tempo e de versão também podem. Seus atributos e elementos também podem ser mapeados para tabelas auxiliares, permitido, assim, que sejam consultados. No entanto, as consultas com aspectos temporais deverão ser definidas com predicados longos e complexos, e as consultas considerando aspectos de versão não poderão ser explicitamente definidas porque SQL não dá suporte à manipulação de versões. Caso existisse esse suporte de tempo e de versão na SQL, nada impediria a realização de consultas sobre documentos com tempo e versão armazenados em colunas.

Nessa lacuna aberta pela limitação da linguagem de consulta do banco de dados é que entra a XQuery com as funções de tempo e de versão. Consultas com aspectos temporais e de versão podem ser realizadas diretamente e os documentos continuarão sendo controlados pelo BD. Enquanto não houver uma linguagem com essa capacidade dentro do BD, justifica-se a aplicação da XQuery estendida.

Os modos três e quatro são apresentados para demonstrar a utilização de outras técnicas para a construção e decomposição de documentos XML. Levando em conta o aspecto da consulta, não se altera nada do que foi mencionado sobre os pontos um e

dois. O que se está querendo mostrar no modo três é que documentos XML padrão ou com características de tempo e de versão não precisam ser armazenados inteiramente em colunas. Eles podem ser decompostos, e seus atributos e elementos podem ser armazenados normalmente em colunas de tipo de dados convencionais (numérico, caracter, etc.). Para isso, é preciso definir um arquivo especial, proposto pelo *XML Extender*, chamado arquivo DAD (*Document Access Definition*). Ele mapeia a estrutura do documento para tabelas do BD, o que permite tanto a composição quanto a decomposição dos documentos.

Assim, os dados mapeados podem sofrer qualquer tipo de manipulação dentro do BD e, posteriormente, podem ser utilizados para compor um documento XML. No momento da composição do documento, o arquivo DAD deve ser conjugado com uma DTD para a construção de um documento bem formado. Se as tabelas no BD estiverem bem projetadas para trabalhar de acordo com o modelo TVM, o uso da representação apresentada neste trabalho permite compor documentos com características de tempo e de versão que poderão ser consultados com a XQuery estendida. Destaca-se que os documentos que se originam desse ciclo (modos 3 e 4) podem ser consultados normalmente pela linguagem estendida.

O problema encontrado para o funcionamento integral desse ambiente foi a composição de documentos XML a partir de dados presentes na base (não apenas documentos com características de tempo e de versão). Esse processo não obteve os resultados esperados visto que a composição não era efetivada por problemas relacionados ao driver JDBC (*Java Database Connector*) requisitado pelo *XML Extender*. Portanto, o documento que representa instâncias de uma aplicação do TVM precisou ser criado sem os recursos do *extender*, não confirmando, em sua totalidade, o ciclo proposto no ambiente.

1.28 Considerações Finais

Este capítulo abordou alguns exemplos de consultas utilizando a extensão proposta. Para isso, foi criada uma pequena aplicação cujos dados foram representados em um documento XML. As consultas realizadas sobre esse documento foram testadas utilizando a ferramenta Software AG Quip. Concluindo a seção, foi descrita uma proposta de ambiente de utilização da biblioteca das novas funções integrada à ferramenta escolhida para realização dos testes.

Quanto ao ambiente, a impossibilidade de realização de testes sobre documentos gerados a partir de dados presentes no banco pôde ser considerada um ponto de relativa frustração. A modelagem de uma aplicação através do TVM em um banco de dados objeto-relacional (IBM DB2) é realizável. Com a integração do *XML Extender* viu-se a possibilidade de criar visões XML da base de dados de uma maneira automatizada através do recurso de coleções XML. Todavia, por razões que não foram integralmente reconhecidas, não foi efetivada a geração do documento XML, diferentemente do processo de colunas XML, que foi testado com sucesso. Ressalta-se, com exceção dessa faceta, que as demais possibilidades de aplicação da XQuery foram contempladas sem maiores dificuldades.

Com relação à ferramenta escolhida e que foi largamente utilizada, não se salienta um ponto problemático. A funcionalidade de bibliotecas de funções é algo que poderia ser mais bem tratado nessa implementação. Ela possui essa característica, no entanto apenas para documentação de funções, visto que as funções armazenadas na

biblioteca não são visíveis ao ambiente de consulta. Postergando essa questão, seu uso foi de fundamental importância para validar as funções implementadas nesse trabalho.

A respeito da aplicação, não foi objetivo criar um exemplo elaborado, mas, sim, um que disponibilizasse uma boa quantidade de dados passíveis de consulta. Consultas que foram apresentadas em um número restrito, contudo, abordaram quase todas as funções disponíveis. Numa análise inicial, pode não ser constatada uma grande vantagem no uso dessas funções. Mas, examinando a estrutura do documento e as peculiaridades de consultas com tempo e com versão, viu-se que o uso apenas dos recursos tradicionais da XQuery tornaria a escrita excessivamente longa e com expressões de caminho relativamente complicadas. É inegável, também, a complexidade de uso da XQuery com as novas funções. Entretanto, elas tornaram o processo de obtenção de informações temporais e de versão até certo ponto independentes de um conhecimento do documento. Se o usuário sabe quais objetos estão representados e como eles estão relacionados, as funções permitem abreviar por demais a escrita de consultas.

8 Conclusões

A XML surgiu com o objetivo de ser uma linguagem de fácil entendimento por parte do usuário, para ser amplamente difundida através da Internet, metas que alcançou com êxito em um curto espaço de tempo. Todavia, seu emprego não se restringiu apenas à *World Wide Web*. Sua sintaxe concisa permite que seja utilizada em muitas outras aplicações visto que o processamento de documentos XML não requer programas avançados.

A expansão de seu uso é tão considerável que atingiu fortemente a área de banco de dados. Os principais SGBDs comerciais já apresentam suporte ao que se pode considerar como "um novo tipo de dado". Além disso, é comum o uso de ferramentas que disponibilizam dados relacionais ou objeto-relacionais em um formato XML.

A relação com a área de banco de dados atingiu um tal ponto de consolidação que muitas empresas já desenvolvem bancos de dados exclusivamente XML, onde os dados são documentos e as consultas, realizadas através de linguagens especiais como XQL, XPath, XQuery, são aplicadas unicamente sobre esses documentos.

Considerando apenas os grandes bancos de dados objeto-relacionais, uma utilização emergente relacionada a XML é a materialização de visões nesse novo formato. Essas visões podem servir de entrada para diversas aplicações, além de serem publicadas na *World Wide Web*.

Como já foi mencionado durante o trabalho, esses mesmos bancos de dados não suportam o Modelo Temporal de Versões, cabendo a realização de mapeamentos para que os dados de aplicações modeladas a partir do TVM sejam adequadamente controlados pelo banco. Se o mapeamento for realizado de forma eficiente, a materialização de visões a partir de uma base TVM pode ser normalmente configurada. No entanto, principalmente em aplicações do TVM, mostra-se importante que essas visões sigam um paradigma comum.

Nessa pesquisa foi apresentado um modelo de representação de dados de aplicações TVM em XML. Objetivou-se congregar em um mesmo documento objetos, versões e controles de objeto versionado que seriam descritos através de elementos e atributos XML. A estrutura utilizada para a representação das instâncias do TVM buscou seguir ao máximo a concepção adotada nas classes do Modelo Temporal de Versões. Assim, o usuário que possui conhecimento acerca do modelo não teria dificuldades para interpretar o documento XML e identificaria facilmente propriedades de controle e atributos de aplicação de versões e objetos.

No contexto do trabalho, a definição desse modelo de representação foi relevante, porém ocupa um *status* secundário dentro do todo. Esse esquema de representação foi fundamental para definir o formato-padrão para um estilo de documento XML com características de tempo e de versão que poderia ser alvo de consultas temporais e versionadas.

Realizar esse tipo de consultas foi o real objetivo da pesquisa. Para tanto, partiu-se para a definição da melhor linguagem para dados XML que atendesse a essa finalidade. Constatou-se que nenhuma das linguagens analisadas apresentava suporte a consultas temporais, muito menos a consultas que tratassem de versões. A partir desse instante, passou-se a examinar qual delas melhor suportaria uma extensão. Foram

analisadas várias funcionalidades pertinentes às linguagens de consulta XML, sendo que a forma visualizada e eleita para evitar modificações na especificação original foi o emprego de funções extensíveis, ponto onde se destacaram a XML-QL, a XQL e, principalmente, a XQuery. Além de apresentar uma especificação mais consistente nesse ponto, a XQuery já apresentava vantagens sobre as demais em outros tópicos, circunstâncias que justificaram seu emprego como base para a extensão.

Cumprida a etapa de escolha da linguagem, concentraram-se esforços para definir que pontos uma linguagem temporal versionada de consulta deveria atender para ter seu uso justificado. Concomitantemente a esta análise, estava sendo proposta pelo Grupo de Pesquisa em Versões e Tempo a especificação da linguagem de consultas para o Modelo Temporal de Versões, denominada *Temporal Version Query Language* (TVQL). Este estudo resultou em um relatório de pesquisa que facilitou em muito o processo de escolha das funcionalidades que deveriam integrar a extensão [MORa 01].

Obviamente, alguns ajustes precisaram ser feitos, pois a TVQL tem orientação voltada para banco de dados, diferentemente da extensão proposta, voltada para documentos XML. Funções auxiliares para o tratamento de aspectos inerentes a dados XML foram desenvolvidas, mas o núcleo da extensão descendeu de técnicas definidas para a TVQL.

Para realização de testes da extensão, foi proposta uma aplicação resumida, com dados de uma aplicação temporal versionada representados em um documento XML que seguia o paradigma criado nessa pesquisa. Para efetivação dos testes, foram vinculadas, através de uma biblioteca, as funções temporais e de versão a uma ferramenta já existente, que implementa a especificação da linguagem XQuery. Depois da análise de quinze consultas considerando as novas funções, alguns pontos puderam ser destacados:

- fica a cargo do usuário a utilização ou não das funções para recuperar informações temporais e de versão. Tais dados podem ser retornados sem o uso das funções, resultando, porém, na especificação de consultas extensas e complexas sejam especificadas;
- a utilização das funções dispensa um conhecimento profundo sobre a estrutura do documento. Conhecendo o funcionamento do TVM, sabendo quais as propriedades que estão representadas no documento, a combinação de funções permite atingir resultados até certo ponto especializados;
- tendo conhecimento superficial da linguagem XPath e da estrutura do documento, mostra-se possível construir saídas ainda mais qualificadas a partir da utilização das funções disponibilizadas pela extensão;
- existe um bom suporte para a recuperação de informações referentes a versões. Porém, considerando os aspectos temporais, novas funções devem ser implementadas. Por exemplo, novos operadores devem ser definidos para tratar instantes temporais;
- a incorporação da palavra reservada `EVER` e da função `PRESENT` em funções que acessam o histórico das propriedades foi uma maneira eficiente para diferenciar o escopo de atuação temporal das funções. Pode não ser a mais estética, mas apresentou os resultados esperados;

- quase todas as funções têm sua implementação fundamentada na estrutura definida para o documento. Logo, se um novo modelo for criado para representar instâncias temporais versionadas, é sabido que as funções não retornarão os dados esperados sem que mudanças sejam executadas.

Partindo dos pontos comentados, sugere-se que novos recursos sejam incorporados à extensão e, por conseqüência, ao trabalho. Dentre eles:

- um novo modelo que represente, além das informações já consideradas, os relacionamentos existentes entre instâncias;
- após a padronização da especificação da linguagem XQuery, prover, além das datas, novas opções de granularidades temporais;
- analisar a possibilidade de separar em documentos diferentes os atributos de controle, previstos no modelo, das propriedades de aplicação, definidas pelo usuário;
- adaptar as funções existentes a diferentes granularidades temporais;
- implementar um protótipo de apoio à especificação de consultas com recursos visuais para escolha das versões e dos objetos a serem analisados na consulta e das funções aptas a operar sobre as instâncias previamente selecionadas;
- incorporar em um ambiente integrado de consultas uma implementação da TVQL, uma ferramenta de criação de visões XML a partir da base TVM e uma implementação da XQuery estendida;
- estabelecer um estudo de caso mais complexo e testar um número significativamente maior de consultas.

Anexo 1 Listel.dtd e Listel.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT listatelefones (contato+)>
<!ELEMENT contato (nome, telefone, endereco?)>
<!ATTLIST contato tipo CDATA #REQUIRED>
<!ELEMENT nome (#PCDATA)>
<!ELEMENT telefone (#PCDATA)>
<!ELEMENT endereco (rua, numero?, cidade)>
<!ELEMENT rua (#PCDATA)>
<!ELEMENT numero (#PCDATA)>
<!ELEMENT cidade (#PCDATA)>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE listatelefones SYSTEM "listel.dtd">
<listatelefones>
    <contato tipo = "Pessoa Fisica">
        <nome>Maria</nome>
        <telefone>334-3421</telefone>
        <endereco>
            <rua>Av. Ipiranga</rua>
            <cidade>Porto Alegre</cidade>
        </endereco>
    </contato>
    <contato tipo = "Pessoa Fisica">
        <nome>Pedro</nome>
        <telefone>321-8971</telefone>
    </contato>
    <contato tipo = "Pessoa Fisica">
        <nome>Paula</nome>
        <telefone>271-1999</telefone>
        <endereco>
            <rua>Alberto Bins</rua>
            <numero>1856</numero>
            <cidade>Caxias do Sul</cidade>
        </endereco>
    </contato>
    <contato tipo = "Pessoa Juridica">
        <nome>Enterprise</nome>
        <telefone>111-1212</telefone>
        <endereco>
            <rua>Bento Gonçalves</rua>
            <numero>14</numero>
            <cidade>Porto Alegre</cidade>
        </endereco>
    </contato>
</listatelefones>
```

Anexo 2 DTD da representação

```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT ObjectInstances (NonTemporalObjects?, TemporalObjects?)>

<!ELEMENT NonTemporalObjects (Object+)>
<!ELEMENT TemporalObjects (Object+)>

<!ELEMENT Object (TemporalObject | Attributes)>
<!ATTLIST Object  tvOID CDATA #REQUIRED
                  name CDATA #IMPLIED
                  class CDATA #REQUIRED
                  type CDATA #IMPLIED>

<!ELEMENT TemporalObject (alive, ((TemporalVersion, Attributes) | VersionedObjectControl))>
<!ELEMENT alive (Value+)>

<!ELEMENT VersionedObjectControl (configurationCount, currentVersion, firstVersion, lastVersion,
          nextVersionNumber, userCurrentFlag, versionCount)>
<!ELEMENT configurationCount (Value+)>
<!ELEMENT currentVersion (Value+)>
<!ELEMENT versionCount (Value+)>
<!ELEMENT firstVersion (Value+)>
<!ELEMENT lastVersion (Value+)>
<!ELEMENT nextVersionNumber (Value)>
<!ELEMENT userCurrentFlag (Value+)>

<!ELEMENT TemporalVersion (ascendant, descendant, predecessor, successor, configuration, status)>
<!ATTLIST TemporalVersion vocOID CDATA #IMPLIED>
<!ELEMENT ascendant (Value+)>
<!ELEMENT descendant (Value+)>
<!ELEMENT predecessor (Value)>
<!ELEMENT successor (Value+)>
<!ELEMENT configuration (Value)>
<!ELEMENT status (Value+)>

<!ELEMENT Attributes (processador?, discoRigido?, valor?, duracaoBateria?, dispositivoApontador?)>
<!ELEMENT processador (Value+)>
<!ELEMENT discoRigido (Value+)>
<!ELEMENT valor (Value+)>
<!ELEMENT duracaoBateria (Value+)>
<!ELEMENT dispositivoApontador (Value+)>

<!ELEMENT Value (#PCDATA)>
<!ATTLIST Value tTimei CDATA #IMPLIED
                tTimef CDATA #IMPLIED
                vTimei CDATA #IMPLIED
                vTimef CDATA #IMPLIED>
```

Anexo 3 XML Schema da representação

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2000/10/XMLSchema" elementFormDefault="qualified">
    <xsd:element name="ObjectInstances">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element name="NonTemporalObjects" minOccurs="0" maxOccurs="1" type="ObjectsType"/>
                <xsd:element name="TemporalObjects" minOccurs="0" maxOccurs="1" type="ObjectsType"/>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
    <xsd:complexType name="ObjectsType">
        <xsd:sequence>
            <xsd:element name="Object" type="ObjectType" minOccurs="1" maxOccurs="unbounded"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="ObjectType">
        <xsd:choice>
            <xsd:element name="TemporalObject" type="TemporalObjectType"/>
            <xsd:element name="Attributes" type="AttributesType"/>
        </xsd:choice>
        <xsd:attribute name="tvOID" type="BaseTVOIDType" use="required"/>
        <xsd:attribute name="name" type="xsd:string"/>
        <xsd:attribute name="class" type="xsd:string" use="required"/>
        <xsd:attribute name="type" type="xsd:string"/>
    </xsd:complexType>
    <xsd:complexType name="AttributesType">
        <xsd:sequence>
            <xsd:element name="processador" type="processadorType" minOccurs="0" maxOccurs="1"/>
            <xsd:element name="discoRigido" type="discoRigidoType" minOccurs="0" maxOccurs="1"/>
            <xsd:element name="valor" type="valorType" minOccurs="0" maxOccurs="1"/>
            <xsd:element name="duracaoBateria" type="duracaoBateriaType" minOccurs="0" maxOccurs="1"/>
            <xsd:element name="dispositivoApontador" type="dispositivoApontadorType" minOccurs="0" maxOccurs="1"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="processadorType">
        <xsd:sequence>
            <xsd:element name="Value" type="ValueStringType" maxOccurs="unbounded"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="discoRigidoType">
        <xsd:sequence>
            <xsd:element name="Value" type="ValueStringType" maxOccurs="unbounded"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="valorType">
        <xsd:sequence>
            <xsd:element name="Value" type="ValueFloatType" maxOccurs="unbounded"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="duracaoBateriaType">
        <xsd:sequence>
            <xsd:element name="Value" type="ValueStringType" maxOccurs="unbounded"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="dispositivoApontadorType">
        <xsd:sequence>
            <xsd:element name="Value" type="ValueStringType" maxOccurs="unbounded"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="ValueStringType">
        <xsd:simpleContent>
            <xsd:restriction base="xsd:string">
                <xsd:attribute name="tTimei" type="xsd:date"/>
                <xsd:attribute name="tTimef" type="xsd:date"/>
                <xsd:attribute name="vTimei" type="xsd:date"/>
                <xsd:attribute name="vTimef" type="xsd:date"/>
            </xsd:restriction>
        </xsd:simpleContent>
    </xsd:complexType>
```

```
<xsd:complexType name="ValueTVOIDType">
    <xsd:simpleContent>
        <xsd:restriction base="BaseTVOIDType">
            <xsd:attribute name="tTimei" type="xsd:date"/>
            <xsd:attribute name="tTimef" type="xsd:date"/>
            <xsd:attribute name="vTimei" type="xsd:date"/>
            <xsd:attribute name="vTimef" type="xsd:date"/>
        </xsd:restriction>
    </xsd:simpleContent>
</xsd:complexType>
<xsd:complexType name="ValueFloatType">
    <xsd:simpleContent>
        <xsd:restriction base="xsd:float">
            <xsd:attribute name="tTimei" type="xsd:date"/>
            <xsd:attribute name="tTimef" type="xsd:date"/>
            <xsd:attribute name="vTimei" type="xsd:date"/>
            <xsd:attribute name="vTimef" type="xsd:date"/>
        </xsd:restriction>
    </xsd:simpleContent>
</xsd:complexType>
<xsd:complexType name="ValueBooleanType">
    <xsd:simpleContent>
        <xsd:restriction base="xsd:boolean">
            <xsd:attribute name="tTimei" type="xsd:date"/>
            <xsd:attribute name="tTimef" type="xsd:date"/>
            <xsd:attribute name="vTimei" type="xsd:date"/>
            <xsd:attribute name="vTimef" type="xsd:date"/>
        </xsd:restriction>
    </xsd:simpleContent>
</xsd:complexType>
<xsd:complexType name="ValueIntegerType">
    <xsd:simpleContent>
        <xsd:restriction base="xsd:integer">
            <xsd:attribute name="tTimei" type="xsd:date"/>
            <xsd:attribute name="tTimef" type="xsd:date"/>
            <xsd:attribute name="vTimei" type="xsd:date"/>
            <xsd:attribute name="vTimef" type="xsd:date"/>
        </xsd:restriction>
    </xsd:simpleContent>
</xsd:complexType>
<xsd:complexType name="ValueStatusType">
    <xsd:simpleContent>
        <xsd:restriction base="BaseStatusType">
            <xsd:attribute name="tTimei" type="xsd:date"/>
            <xsd:attribute name="tTimef" type="xsd:date"/>
            <xsd:attribute name="vTimei" type="xsd:date"/>
            <xsd:attribute name="vTimef" type="xsd:date"/>
        </xsd:restriction>
    </xsd:simpleContent>
</xsd:complexType>
<xsd:complexType name="TemporalObjectType">
    <xsd:sequence>
        <xsd:element name="alive" type="aliveType"/>
        <xsd:choice>
            <xsd:sequence>
                <xsd:element name="TemporalVersion" type="TemporalVersionType"/>
                <xsd:element name="Attributes" type="AttributesType"/>
            </xsd:sequence>
            <xsd:element name="VersionedObjectControl" type="VersionedObjectControlType"/>
        </xsd:choice>
    </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="aliveType">
    <xsd:sequence>
        <xsd:element name="Value" type="ValueBooleanType" minOccurs="1" maxOccurs="unbounded" default="true"/>
    </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="TemporalVersionType">
    <xsd:sequence>
        <xsd:element name="ascendant" type="ascendantType"/>
        <xsd:element name="descendant" type="descendantType"/>
        <xsd:element name="predecessor" type="predecessorType"/>
        <xsd:element name="successor" type="successorType"/>
        <xsd:element name="configuration" type="configurationType"/>
        <xsd:element name="status" type="statusType"/>
    </xsd:sequence>
    <xsd:attribute name="vocOID" type="BaseTVOIDType"/>
</xsd:complexType>
<xsd:complexType name="ascendantType">
    <xsd:sequence>
        <xsd:element name="Value" type="ValueTVOIDType" maxOccurs="unbounded" default="null"/>
```

```
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="descendantType">
        <xsd:sequence>
            <xsd:element name="Value" type="ValueTVOIDType" maxOccurs="unbounded" default="null"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="predecessorType">
        <xsd:sequence>
            <xsd:element name="Value" type="ValueTVOIDType" default="null"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="successorType">
        <xsd:sequence>
            <xsd:element name="Value" type="ValueTVOIDType" maxOccurs="unbounded" default="null"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="configurationType">
        <xsd:sequence>
            <xsd:element name="Value" type="ValueBooleanType" default="false"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="statusType">
        <xsd:sequence>
            <xsd:element name="Value" type="ValueStatusType" maxOccurs="unbounded" default="Working"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="VersionedObjectControlType">
        <xsd:sequence>
            <xsd:element name="configurationCount" type="configurationCountType"/>
            <xsd:element name="currentVersion" type="currentVersionType"/>
            <xsd:element name="firstVersion" type="firstVersionType"/>
            <xsd:element name="lastVersion" type="lastVersionType"/>
            <xsd:element name="nextVersionNumber" type="nextVersionNumberType"/>
            <xsd:element name="userCurrentFlag" type="userCurrentFlagType"/>
            <xsd:element name="versionCount" type="versionCountType"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="configurationCountType">
        <xsd:sequence>
            <xsd:element name="Value" type="ValueIntegerType" maxOccurs="unbounded"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="currentVersionType">
        <xsd:sequence>
            <xsd:element name="Value" type="ValueTVOIDType" maxOccurs="unbounded"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="firstVersionType">
        <xsd:sequence>
            <xsd:element name="Value" type="ValueTVOIDType" maxOccurs="unbounded"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="lastVersionType">
        <xsd:sequence>
            <xsd:element name="Value" type="ValueTVOIDType" maxOccurs="unbounded"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="nextVersionNumberType">
        <xsd:sequence>
            <xsd:element name="Value" type="ValueIntegerType" default="3"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="userCurrentFlagType">
        <xsd:sequence>
            <xsd:element name="Value" type="ValueBooleanType" maxOccurs="unbounded" default="false"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="versionCountType">
        <xsd:sequence>
            <xsd:element name="Value" type="ValueIntegerType" maxOccurs="unbounded" default="2"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:simpleType name="BaseTVOIDType">
        <xsd:restriction base="xsd:string">
            <xsd:pattern value="([0-9]+,[0-9]+,(([0-9]+)|null))|null"/>
        </xsd:restriction>
    </xsd:simpleType>
    <xsd:simpleType name="BaseStatusType">
        <xsd:restriction base="xsd:string">
            <xsd:enumeration value="Working"/>
```

```
                <xsd:enumeration value="Stable"/>
                <xsd:enumeration value="Consolidated"/>
                <xsd:enumeration value="Deactivated"/>
            </xsd:restriction>
        </xsd:simpleType>
</xsd:schema>
```

Anexo 4 Instâncias do TVM representadas em XML[4]

```
<?xml version="1.0" encoding="UTF-8"?>
<ObjectInstances xsi:noNamespaceSchemaLocation="x-tv-instances.xsd" xmlns:xsi="http://www.w3.org/2000/10/XMLSchema-instance">
        <TemporalObject>
            <alive>
                <Value tTimei="2001-07-06" tTimef="9999-12-31" vTimei="2001-07-06" vTimef="9999-12-31">true</Value>
            </alive>
            <TemporalVersion vocOID="30,3,1">
                <ascendant>
                    <Value tTimei="2001-07-06" tTimef="9999-12-31" vTimei="2001-07-06" vTimef="9999-12-31">null</Value>
                </ascendant>
                <descendant>
                    <Value tTimei="2001-07-06" tTimef="2001-07-10" vTimei="2001-07-06" vTimef="9999-12-31">null</Value>
                    <Value tTimei="2001-07-11" tTimef="9999-12-31" vTimei="2001-07-06" vTimef="2001-07-10">null</Value>
                    <Value tTimei="2001-07-11" tTimef="9999-12-31" vTimei="2001-07-11" vTimef="9999-12-31">10,2,3</Value>
                </descendant>
                <predecessor>
                    <Value>10,1,3</Value>
                </predecessor>
                <successor>
                    <Value tTimei="2001-07-06" tTimef="9999-12-31" vTimei="2001-07-06" vTimef="9999-12-31">null</Value>
                </successor>
                <configuration>
                    <Value>false</Value>
                </configuration>
                <status>
                    <Value tTimei="2001-07-06" tTimef="2001-07-15" vTimei="2001-07-06" vTimef="9999-12-31">Working</Value>
                    <Value tTimei="2001-07-16" tTimef="9999-12-31" vTimei="2001-07-06" vTimef="2001-07-15">Working</Value>
                    <Value tTimei="2001-07-16" tTimef="2001-08-05" vTimei="2001-07-16" vTimef="9999-12-31">Stable</Value>
                    <Value tTimei="2001-08-06" tTimef="9999-12-31" vTimei="2001-07-16" vTimef="2001-08-05">Stable</Value>
                    <Value tTimei="2001-08-06" tTimef="9999-12-31" vTimei="2001-08-06" vTimef="9999-12-31">Consolid.</Value>
                </status>
            </TemporalVersion>
            <Attributes>
                <processador>
                    <Value tTimei="2001-07-06" tTimef="9999-12-31" vTimei="2001-07-06" vTimef="9999-12-31">850</Value>
                </processador>
                <discoRigido>
                    <Value tTimei="2001-07-06" tTimef="2001-07-15" vTimei="2001-07-06" vTimef="9999-12-31">10</Value>
                    <Value tTimei="2001-07-16" tTimef="9999-12-31" vTimei="2001-07-06" vTimef="2001-07-15">10</Value>
                    <Value tTimei="2001-07-16" tTimef="9999-12-31" vTimei="2001-07-16" vTimef="2001-07-20">5</Value>
                    <Value tTimei="2001-07-16" tTimef="9999-12-31" vTimei="2001-07-21" vTimef="9999-12-31">8</Value>
                </discoRigido>
                <valor>
                    <Value tTimei="2001-07-06" tTimef="2001-07-15" vTimei="2001-07-06" vTimef="9999-12-31">4900.00</Value>
                    <Value tTimei="2001-07-16" tTimef="9999-12-31" vTimei="2001-07-06" vTimef="2001-07-15">4900.00</Value>
                    <Value tTimei="2001-07-16" tTimef="9999-12-31" vTimei="2001-07-16" vTimef="9999-12-31">4500.00</Value>
                </valor>
            </Attributes>
        </TemporalObject>
    </Object>
    <Object tvOID="10,1,1" name="PC1" class="Computador" type="Version">
        <TemporalObject>
            <alive>
                <Value tTimei="2001-05-15" tTimef="9999-12-31" vTimei="2001-05-15" vTimef="9999-12-31">true</Value>
            </alive>
            <TemporalVersion vocOID="30,3,1">
                <ascendant>
                    <Value tTimei="2001-05-15" tTimef="9999-12-31" vTimei="2001-05-15" vTimef="9999-12-31">null</Value>
                </ascendant>
                <descendant>
                    <Value tTimei="2001-05-15" tTimef="2001-05-21" vTimei="2001-05-15" vTimef="9999-12-31">10,2,1</Value>
                    <Value tTimei="2001-05-21" tTimef="9999-12-31" vTimei="2001-05-15" vTimef="2001-05-20">10,2,1</Value>
                    <Value tTimei="2001-05-21" tTimef="9999-12-31" vTimei="2001-05-21" vTimef="9999-12-31">null</Value>
                </descendant>
                <predecessor>
                    <Value>null</Value>
                </predecessor>
                <successor>
                    <Value tTimei="2001-05-15" tTimef="2001-06-20" vTimei="2001-05-15" vTimef="9999-12-31">null</Value>
                    <Value tTimei="2001-06-21" tTimef="9999-12-31" vTimei="2001-05-15" vTimef="2001-06-20">null</Value>
                    <Value tTimei="2001-06-21" tTimef="2001-06-24" vTimei="2001-06-21" vTimef="9999-12-31">10,1,2</Value>
                    <Value tTimei="2001-06-25" tTimef="9999-12-31" vTimei="2001-06-21" vTimef="2001-06-24">10,1,2</Value>
                    <Value tTimei="2001-06-25" tTimef="9999-12-31" vTimei="2001-06-25" vTimef="9999-12-31">10,1,2</Value>
```

[4] Em elementos `Value`, descendentes do elemento `status`, houve abreviação do valor `Consolidated` para `Consolid.`. Essa medida foi adotada para apresentar todo o elemento numa mesma linha sem que a fonte fosse diminuída em excesso.

```
                    <Value tTimei="2001-06-25" tTimef="9999-12-31" vTimei="2001-06-25" vTimef="9999-12-31">10,1,3</Value>
                </successor>
                <configuration>
                    <Value>false</Value>
                </configuration>
                <status>
                    <Value tTimei="2001-05-15" tTimef="2001-06-10" vTimei="2001-05-15" vTimef="9999-12-31">Working</Value>
                    <Value tTimei="2001-06-11" tTimef="9999-12-31" vTimei="2001-05-15" vTimef="2001-06-10">Working</Value>
                    <Value tTimei="2001-06-11" tTimef="9999-12-31" vTimei="2001-06-11" vTimef="9999-12-31">Stable</Value>
                </status>
            </TemporalVersion>
            <Attributes>
                <processador>
                    <Value tTimei="2001-05-15" tTimef="2001-05-24" vTimei="2001-05-15" vTimef="9999-12-31">700</Value>
                    <Value tTimei="2001-05-25" tTimef="9999-12-31" vTimei="2001-05-15" vTimef="2001-05-24">700</Value>
                    <Value tTimei="2001-06-25" tTimef="2001-06-08" vTimei="2001-05-25" vTimef="9999-12-31">750</Value>
                    <Value tTimei="2001-06-09" tTimef="9999-12-31" vTimei="2001-05-25" vTimef="2001-06-08">750</Value>
                    <Value tTimei="2001-06-09" tTimef="9999-12-31" vTimei="2001-06-09" vTimef="9999-12-31">700</Value>
                </processador>
                <discoRigido>
                    <Value tTimei="2001-05-15" tTimef="2001-06-22" vTimei="2001-05-15" vTimef="9999-12-31">5</Value>
                    <Value tTimei="2001-06-23" tTimef="9999-12-31" vTimei="2001-05-15" vTimef="2001-06-22">5</Value>
                    <Value tTimei="2001-06-23" tTimef="9999-12-31" vTimei="2001-06-23" vTimef="9999-12-31">10</Value>
                </discoRigido>
                <valor>
                    <Value tTimei="2001-05-15" tTimef="9999-12-31" vTimei="2001-05-15" vTimef="9999-12-31">4000.00</Value>
                </valor>
            </Attributes>
        </TemporalObject>
    </Object>
    <Object tvOID="10,1,2" name="PC2" class="Computador" type="Version">
        <TemporalObject>
            <alive>
                <Value tTimei="2001-06-21" tTimef="9999-12-31" vTimei="2001-06-21" vTimef="9999-12-31">true</Value>
            </alive>
            <TemporalVersion vocOID="30,3,1">
                <ascendant>
                    <Value tTimei="2001-06-21" tTimef="9999-12-31" vTimei="2001-06-21" vTimef="9999-12-31">null</Value>
                </ascendant>
                <descendant>
                    <Value tTimei="2001-06-21" tTimef="2001-06-27" vTimei="2001-06-21" vTimef="9999-12-31">10,2,1</Value>
                    <Value tTimei="2001-06-28" tTimef="9999-12-31" vTimei="2001-06-21" vTimef="2001-06-27">10,2,1</Value>
                    <Value tTimei="2001-06-28" tTimef="2001-07-04" vTimei="2001-06-28" vTimef="9999-12-31">10,2,2</Value>
                    <Value tTimei="2001-07-05" tTimef="9999-12-31" vTimei="2001-06-28" vTimef="2001-07-04">10,2,2</Value>
                    <Value tTimei="2001-07-05" tTimef="9999-12-31" vTimei="2001-07-05" vTimef="9999-12-31">null</Value>
                </descendant>
                <predecessor>
                    <Value>10,1,1</Value>
                </predecessor>
                <successor>
                    <Value tTimei="2001-06-21" tTimef="9999-12-31" vTimei="2001-06-21" vTimef="9999-12-31">null</Value>
                </successor>
                <configuration>
                    <Value>false</Value>
                </configuration>
                <status>
                    <Value tTimei="2001-06-21" tTimef="2001-06-24" vTimei="2001-06-21" vTimef="9999-12-31">Working</Value>
                    <Value tTimei="2001-06-25" tTimef="9999-12-31" vTimei="2001-06-21" vTimef="2001-06-24">Working</Value>
                    <Value tTimei="2001-06-25" tTimef="9999-12-31" vTimei="2001-06-25" vTimef="2001-06-26">Stable</Value>
                    <Value tTimei="2001-06-25" tTimef="9999-12-31" vTimei="2001-06-27" vTimef="9999-12-31">Consolid.</Value>
                </status>
            </TemporalVersion>
            <Attributes>
                <processador>
                    <Value tTimei="2001-06-21" tTimef="9999-12-31" vTimei="2001-06-21" vTimef="2001-06-22">700</Value>
                    <Value tTimei="2001-06-21" tTimef="9999-12-31" vTimei="2001-06-23" vTimef="9999-12-31">900</Value>
                </processador>
                <discoRigido>
                    <Value tTimei="2001-06-21" tTimef="9999-12-31" vTimei="2001-06-21" vTimef="2001-07-03">5</Value>
                    <Value tTimei="2001-06-21" tTimef="9999-12-31" vTimei="2001-07-04" vTimef="9999-12-31">10</Value>
                </discoRigido>
                <valor>
                    <Value tTimei="2001-06-21" tTimef="9999-12-31" vTimei="2001-06-21" vTimef="2001-06-23">4000.00</Value>
                    <Value tTimei="2001-06-21" tTimef="9999-12-31" vTimei="2001-06-24" vTimef="9999-12-31">4200.00</Value>
                </valor>
            </Attributes>
        </TemporalObject>
    </Object>
    <Object tvOID="10,1,3" name="PC3" class="Computador" type="Version">
        <TemporalObject>
            <alive>
                <Value tTimei="2001-06-25" tTimef="9999-12-31" vTimei="2001-06-25" vTimef="9999-12-31">true</Value>
            </alive>
            <TemporalVersion vocOID="30,3,1">
                <ascendant>
                    <Value tTimei="2001-06-25" tTimef="9999-12-31" vTimei="2001-06-25" vTimef="9999-12-31">null</Value>
                </ascendant>
                <descendant>
                    <Value tTimei="2001-06-25" tTimef="2001-07-04" vTimei="2001-06-25" vTimef="9999-12-31">null</Value>
                    <Value tTimei="2001-07-05" tTimef="9999-12-31" vTimei="2001-06-25" vTimef="2001-07-04">null</Value>
                    <Value tTimei="2001-07-05" tTimef="2001-07-06" vTimei="2001-07-05" vTimef="9999-12-31">10,2,2</Value>
                    <Value tTimei="2001-07-07" tTimef="9999-12-31" vTimei="2001-07-05" vTimef="2001-07-06">10,2,2</Value>
                    <Value tTimei="2001-07-07" tTimef="2001-07-10" vTimei="2001-07-07" vTimef="9999-12-31">10,2,3</Value>
```

```
                    <Value tTimei="2001-07-11" tTimef="9999-12-31" vTimei="2001-07-07" vTimef="2001-07-10">10,2,3</Value>
                    <Value tTimei="2001-07-11" tTimef="9999-12-31" vTimei="2001-07-11" vTimef="9999-12-31">null</Value>
                </descendant>
                <predecessor>
                    <Value>10,1,1</Value>
                </predecessor>
                <successor>
                    <Value tTimei="2001-06-25" tTimef="2001-07-05" vTimei="2001-06-25" vTimef="9999-12-31">null</Value>
                    <Value tTimei="2001-07-06" tTimef="9999-12-31" vTimei="2001-06-25" vTimef="2001-07-05">null</Value>
                    <Value tTimei="2001-07-06" tTimef="9999-12-31" vTimei="2001-07-06" vTimef="9999-12-31">10,1,4</Value>
                </successor>
                <configuration>
                    <Value>false</Value>
                </configuration>
                <status>
                    <Value tTimei="2001-06-25" tTimef="2001-07-01" vTimei="2001-06-25" vTimef="9999-12-31">Working</Value>
                    <Value tTimei="2001-07-02" tTimef="9999-12-31" vTimei="2001-06-25" vTimef="2001-07-01">Working</Value>
                    <Value tTimei="2001-07-02" tTimef="9999-12-31" vTimei="2001-07-02" vTimef="9999-12-31">Stable</Value>
                </status>
            </TemporalVersion>
            <Attributes>
                <processador>
                    <Value tTimei="2001-06-25" tTimef="2001-06-28" vTimei="2001-06-25" vTimef="9999-12-31">700</Value>
                    <Value tTimei="2001-06-29" tTimef="9999-12-31" vTimei="2001-06-25" vTimef="2001-06-28">700</Value>
                    <Value tTimei="2001-06-29" tTimef="9999-12-31" vTimei="2001-06-29" vTimef="9999-12-31">850</Value>
                </processador>
                <discoRigido>
                    <Value tTimei="2001-06-25" tTimef="9999-12-31" vTimei="2001-06-25" vTimef="2001-06-28">10</Value>
                    <Value tTimei="2001-06-25" tTimef="9999-12-31" vTimei="2001-06-29" vTimef="2001-07-04">8</Value>
                    <Value tTimei="2001-06-25" tTimef="9999-12-31" vTimei="2001-07-05" vTimef="9999-12-31">10</Value>
                </discoRigido>
                <valor>
                    <Value tTimei="2001-06-25" tTimef="2001-07-04" vTimei="2001-06-25" vTimef="9999-12-31">4000.00</Value>
                    <Value tTimei="2001-07-05" tTimef="9999-12-31" vTimei="2001-06-25" vTimef="2001-07-04">4000.00</Value>
                    <Value tTimei="2001-07-05" tTimef="9999-12-31" vTimei="2001-07-05" vTimef="9999-12-31">4900.00</Value>
                </valor>
            </Attributes>
        </TemporalObject>
    </Object>
    <Object tvOID="10,1,4" name="PC4" class="Computador" type="Version">
        <TemporalObject>
            <alive>
                <Value tTimei="2001-07-06" tTimef="9999-12-31" vTimei="2001-07-06" vTimef="9999-12-31">true</Value>
            </alive>
            <TemporalVersion vocOID="30,3,1">
                <ascendant>
                    <Value tTimei="2001-07-06" tTimef="9999-12-31" vTimei="2001-07-06" vTimef="9999-12-31">null</Value>
                </ascendant>
                <descendant>
                    <Value tTimei="2001-07-06" tTimef="2001-07-10" vTimei="2001-07-06" vTimef="9999-12-31">null</Value>
                    <Value tTimei="2001-07-11" tTimef="9999-12-31" vTimei="2001-07-06" vTimef="2001-07-10">null</Value>
                    <Value tTimei="2001-07-11" tTimef="9999-12-31" vTimei="2001-07-11" vTimef="9999-12-31">10,2,3</Value>
                </descendant>
                <predecessor>
                    <Value>10,1,3</Value>
                </predecessor>
                <successor>
                    <Value tTimei="2001-07-06" tTimef="9999-12-31" vTimei="2001-07-06" vTimef="9999-12-31">null</Value>
                </successor>
                <configuration>
                    <Value>false</Value>
                </configuration>
                <status>
                    <Value tTimei="2001-07-06" tTimef="2001-07-15" vTimei="2001-07-06" vTimef="9999-12-31">Working</Value>
                    <Value tTimei="2001-07-16" tTimef="9999-12-31" vTimei="2001-07-06" vTimef="2001-07-15">Working</Value>
                    <Value tTimei="2001-07-16" tTimef="2001-08-05" vTimei="2001-07-16" vTimef="9999-12-31">Stable</Value>
                    <Value tTimei="2001-08-06" tTimef="9999-12-31" vTimei="2001-07-16" vTimef="2001-08-05">Stable</Value>
                    <Value tTimei="2001-08-06" tTimef="9999-12-31" vTimei="2001-08-06" vTimef="9999-12-31">Consolid.</Value>
                </status>
            </TemporalVersion>
            <Attributes>
                <processador>
                    <Value tTimei="2001-07-06" tTimef="9999-12-31" vTimei="2001-07-06" vTimef="9999-12-31">850</Value>
                </processador>
                <discoRigido>
                    <Value tTimei="2001-07-06" tTimef="2001-07-15" vTimei="2001-07-06" vTimef="9999-12-31">10</Value>
                    <Value tTimei="2001-07-16" tTimef="9999-12-31" vTimei="2001-07-06" vTimef="2001-07-15">10</Value>
                    <Value tTimei="2001-07-16" tTimef="9999-12-31" vTimei="2001-07-16" vTimef="2001-07-20">5</Value>
                    <Value tTimei="2001-07-16" tTimef="9999-12-31" vTimei="2001-07-21" vTimef="9999-12-31">8</Value>
                </discoRigido>
                <valor>
                    <Value tTimei="2001-07-06" tTimef="2001-07-15" vTimei="2001-07-06" vTimef="9999-12-31">4900.00</Value>
                    <Value tTimei="2001-07-16" tTimef="9999-12-31" vTimei="2001-07-06" vTimef="2001-07-15">4900.00</Value>
                    <Value tTimei="2001-07-16" tTimef="9999-12-31" vTimei="2001-07-16" vTimef="9999-12-31">4500.00</Value>
                </valor>
            </Attributes>
        </TemporalObject>
    </Object>
    <Object tvOID="11,1,0" name="PCX" class="Computador" type="VersionedObject">
        <TemporalObject>
            <alive>
                <Value tTimei="2001-10-01" tTimef="9999-12-31" vTimei="2001-10-01" vTimef="9999-12-31">true</Value>
            </alive>
```

```
            <TemporalVersion vocOID="35,3,1">
                <ascendant>
                    <Value tTimei="2001-10-01" tTimef="9999-12-31" vTimei="2001-10-01" vTimef="9999-12-31">null</Value>
                </ascendant>
                <descendant>
                    <Value tTimei="2001-10-01" tTimef="9999-12-31" vTimei="2001-10-01" vTimef="9999-12-31">null</Value>
                </descendant>
                <predecessor>
                    <Value>11,1,1</Value>
                </predecessor>
                <successor>
                    <Value tTimei="2001-10-01" tTimef="9999-12-31" vTimei="2001-10-01" vTimef="9999-12-31">null</Value>
                </successor>
                <configuration>
                    <Value>false</Value>
                </configuration>
                <status>
                    <Value tTimei="2001-10-01" tTimef="9999-12-31" vTimei="2001-10-01" vTimef="9999-12-31">Working</Value>
                </status>
            </TemporalVersion>
            <Attributes>
                <processador>
                    <Value tTimei="2001-10-01" tTimef="9999-12-31" vTimei="2001-10-01" vTimef="9999-12-31">1000</Value>
                </processador>
                <discoRigido>
                    <Value tTimei="2001-10-01" tTimef="9999-12-31" vTimei="2001-10-01" vTimef="9999-12-31">10</Value>
                </discoRigido>
                <valor>
                    <Value tTimei="2001-10-01" tTimef="9999-12-31" vTimei="2001-10-01" vTimef="9999-12-31">5800.00</Value>
                </valor>
            </Attributes>
        </TemporalObject>
    </Object>
    <Object tvOID="11,1,1" name="PCX1" class="Computador" type="Version">
        <TemporalObject>
            <alive>
                <Value tTimei="2001-08-25" tTimef="9999-12-31" vTimei="2001-08-25" vTimef="9999-12-31">true</Value>
            </alive>
            <TemporalVersion vocOID="35,3,1">
                <ascendant>
                    <Value tTimei="2001-08-25" tTimef="9999-12-31" vTimei="2001-08-25" vTimef="9999-12-31">null</Value>
                </ascendant>
                <descendant>
                    <Value tTimei="2001-08-25" tTimef="2001-09-14" vTimei="2001-08-25" vTimef="9999-12-31">11,2,1</Value>
                    <Value tTimei="2001-09-15" tTimef="9999-12-31" vTimei="2001-08-25" vTimef="2001-09-14">11,2,1</Value>
                    <Value tTimei="2001-09-15" tTimef="2001-09-30" vTimei="2001-09-15" vTimef="9999-12-31">11,2,2</Value>
                    <Value tTimei="2001-10-01" tTimef="9999-12-31" vTimei="2001-09-15" vTimef="2001-09-30">11,2,2</Value>
                    <Value tTimei="2001-10-01" tTimef="9999-12-31" vTimei="2001-10-01" vTimef="9999-12-31">11,2,3</Value>
                </descendant>
                <predecessor>
                    <Value>null</Value>
                </predecessor>
                <successor>
                    <Value tTimei="2001-08-25" tTimef="9999-12-31" vTimei="2001-08-25" vTimef="9999-12-31">null</Value>
                    <Value tTimei="2001-10-01" tTimef="9999-12-31" vTimei="2001-08-25" vTimef="2001-09-30">null</Value>
                    <Value tTimei="2001-10-01" tTimef="9999-12-31" vTimei="2001-10-01" vTimef="9999-12-31">11,1,1</Value>
                </successor>
                <configuration>
                    <Value>false</Value>
                </configuration>
                <status>
                    <Value tTimei="2001-08-25" tTimef="2001-09-05" vTimei="2001-08-25" vTimef="9999-12-31">Working</Value>
                    <Value tTimei="2001-09-06" tTimef="9999-12-31" vTimei="2001-08-25" vTimef="2001-09-05">Working</Value>
                    <Value tTimei="2001-09-06" tTimef="9999-12-31" vTimei="2001-09-06" vTimef="9999-12-31">Stable</Value>
                </status>
            </TemporalVersion>
            <Attributes>
                <processador>
                    <Value tTimei="2001-08-25" tTimef="9999-12-31" vTimei="2001-08-25" vTimef="2001-08-29">1000</Value>
                    <Value tTimei="2001-08-25" tTimef="2001-09-02" vTimei="2001-08-30" vTimef="9999-12-31">850</Value>
                    <Value tTimei="2001-09-03" tTimef="9999-12-31" vTimei="2001-08-30" vTimef="2001-09-02">850</Value>
                    <Value tTimei="2001-09-03" tTimef="2001-09-04" vTimei="2001-09-03" vTimef="9999-12-31">900</Value>
                    <Value tTimei="2001-09-05" tTimef="9999-12-31" vTimei="2001-09-03" vTimef="2001-09-04">900</Value>
                    <Value tTimei="2001-09-05" tTimef="9999-12-31" vTimei="2001-09-05" vTimef="9999-12-31">1000</Value>
                </processador>
                <discoRigido>
                    <Value tTimei="2001-08-25" tTimef="9999-12-31" vTimei="2001-08-25" vTimef="2001-08-29">15</Value>
                    <Value tTimei="2001-08-25" tTimef="2001-09-02" vTimei="2001-08-30" vTimef="9999-12-31">5</Value>
                    <Value tTimei="2001-09-03" tTimef="9999-12-31" vTimei="2001-08-30" vTimef="2001-09-02">5</Value>
                    <Value tTimei="2001-09-03" tTimef="9999-12-31" vTimei="2001-09-03" vTimef="9999-12-31">10</Value>
                </discoRigido>
                <valor>
                    <Value tTimei="2001-08-25" tTimef="2001-09-05" vTimei="2001-08-25" vTimef="9999-12-31">5400.00</Value>
                    <Value tTimei="2001-09-06" tTimef="9999-12-31" vTimei="2001-08-25" vTimef="2001-09-05">5400.00</Value>
                    <Value tTimei="2001-09-06" tTimef="9999-12-31" vTimei="2001-09-06" vTimef="9999-12-31">5800.00</Value>
                </valor>
            </Attributes>
        </TemporalObject>
    </Object>
    <Object tvOID="11,1,2" name="PCX2" class="Computador" type="Version">
        <TemporalObject>
            <alive>
                <Value tTimei="2001-10-01" tTimef="9999-12-31" vTimei="2001-10-01" vTimef="9999-12-31">true</Value>
```

```
            </alive>
            <TemporalVersion vocOID="35,3,1">
                <ascendant>
                    <Value tTimei="2001-10-01" tTimef="9999-12-31" vTimei="2001-10-01" vTimef="9999-12-31">null</Value>
                </ascendant>
                <descendant>
                    <Value tTimei="2001-10-01" tTimef="9999-12-31" vTimei="2001-10-01" vTimef="9999-12-31">null</Value>
                </descendant>
                <predecessor>
                    <Value>11,1,1</Value>
                </predecessor>
                <successor>
                    <Value tTimei="2001-10-01" tTimef="9999-12-31" vTimei="2001-10-01" vTimef="9999-12-31">null</Value>
                </successor>
                <configuration>
                    <Value>false</Value>
                </configuration>
                <status>
                    <Value tTimei="2001-10-01" tTimef="9999-12-31" vTimei="2001-10-01" vTimef="9999-12-31">Working</Value>
                </status>
            </TemporalVersion>
            <Attributes>
                <processador>
                    <Value tTimei="2001-10-01" tTimef="9999-12-31" vTimei="2001-10-01" vTimef="9999-12-31">1000</Value>
                </processador>
                <discoRigido>
                    <Value tTimei="2001-10-01" tTimef="9999-12-31" vTimei="2001-10-01" vTimef="9999-12-31">10</Value>
                </discoRigido>
                <valor>
                    <Value tTimei="2001-10-01" tTimef="9999-12-31" vTimei="2001-10-01" vTimef="9999-12-31">5800.00</Value>
                </valor>
            </Attributes>
        </TemporalObject>
    </Object>
    <Object tvOID="10,2,0" name="E500" class="Notebook" type="VersionedObject">
        <TemporalObject>
            <alive>
                <Value tTimei="2001-07-02" tTimef="9999-12-31" vTimei="2001-07-02" vTimef="9999-12-31">true</Value>
            </alive>
            <TemporalVersion vocOID="40,3,1">
                <ascendant>
                    <Value tTimei="2001-07-02" tTimef="2001-07-06" vTimei="2001-07-02" vTimef="9999-12-31">null</Value>
                    <Value tTimei="2001-07-07" tTimef="9999-12-31" vTimei="2001-07-02" vTimef="2001-07-06">null</Value>
                    <Value tTimei="2001-07-07" tTimef="2001-07-10" vTimei="2001-07-07" vTimef="9999-12-31">10,1,3</Value>
                    <Value tTimei="2001-07-11" tTimef="9999-12-31" vTimei="2001-07-07" vTimef="2001-07-10">10,1,3</Value>
                    <Value tTimei="2001-07-11" tTimef="9999-12-31" vTimei="2001-07-11" vTimef="9999-12-31">10,1,4</Value>
                </ascendant>
                <descendant>
                    <Value tTimei="2001-07-02" tTimef="9999-12-31" vTimei="2001-07-02" vTimef="9999-12-31">null</Value>
                </descendant>
                <predecessor>
                    <Value>10,2,2</Value>
                </predecessor>
                <successor>
                    <Value tTimei="2001-07-02" tTimef="9999-12-31" vTimei="2001-07-02" vTimef="9999-12-31">null</Value>
                </successor>
                <configuration>
                    <Value>false</Value>
                </configuration>
                <status>
                    <Value tTimei="2001-07-02" tTimef="2001-07-15" vTimei="2001-07-02" vTimef="9999-12-31">Working</Value>
                    <Value tTimei="2001-07-16" tTimef="9999-12-31" vTimei="2001-07-02" vTimef="2001-07-15">Working</Value>
                    <Value tTimei="2001-07-16" tTimef="2001-07-20" vTimei="2001-07-16" vTimef="9999-12-31">Stable</Value>
                    <Value tTimei="2001-07-21" tTimef="9999-12-31" vTimei="2001-07-16" vTimef="2001-07-20">Stable</Value>
                    <Value tTimei="2001-07-21" tTimef="9999-12-31" vTimei="2001-07-21" vTimef="9999-12-31">Consolid.</Value>
                </status>
            </TemporalVersion>
            <Attributes>
                <duracaoBateria>
                    <Value tTimei="2001-07-02" tTimef="2001-07-09" vTimei="2001-07-02" vTimef="9999-12-31">200</Value>
                    <Value tTimei="2001-07-10" tTimef="9999-12-31" vTimei="2001-07-02" vTimef="2001-07-09">200</Value>
                    <Value tTimei="2001-07-10" tTimef="9999-12-31" vTimei="2001-07-10" vTimef="9999-12-31">220</Value>
                </duracaoBateria>
                <dispositivoApontador>
                    <Value tTimei="2001-07-02" tTimef="9999-12-31" vTimei="2001-07-02" vTimef="9999-12-31">Accupoint</Value>
                </dispositivoApontador>
            </Attributes>
        </TemporalObject>
    </Object>
    <Object tvOID="10,2,1" name="E500A" class="Notebook" type="Version">
        <TemporalObject>
            <alive>
                <Value tTimei="2001-05-05" tTimef="9999-12-31" vTimei="2001-05-05" vTimef="9999-12-31">true</Value>
            </alive>
            <TemporalVersion vocOID="40,3,1">
                <ascendant>
                    <Value tTimei="2001-05-05" tTimef="2001-05-14" vTimei="2001-05-05" vTimef="9999-12-31">null</Value>
                    <Value tTimei="2001-05-15" tTimef="9999-12-31" vTimei="2001-05-05" vTimef="2001-05-14">null</Value>
                    <Value tTimei="2001-05-15" tTimef="2001-06-20" vTimei="2001-05-15" vTimef="9999-12-31">10,1,1</Value>
                    <Value tTimei="2001-06-21" tTimef="9999-12-31" vTimei="2001-05-15" vTimef="2001-06-20">10,1,1</Value>
                    <Value tTimei="2001-06-21" tTimef="2001-06-27" vTimei="2001-06-21" vTimef="9999-12-31">10,1,2</Value>
                    <Value tTimei="2001-06-28" tTimef="9999-12-31" vTimei="2001-06-21" vTimef="2001-06-27">10,1,2</Value>
                    <Value tTimei="2001-06-28" tTimef="9999-12-31" vTimei="2001-06-28" vTimef="9999-12-31">null</Value>
```

```
                    </ascendant>
                    <descendant>
                        <Value tTimei="2001-05-05" tTimef="9999-12-31" vTimei="2001-05-05" vTimef="9999-12-31">null</Value>
                    </descendant>
                    <predecessor>
                        <Value>null</Value>
                    </predecessor>
                    <successor>
                        <Value tTimei="2001-05-05" tTimef="2001-06-27" vTimei="2001-05-05" vTimef="9999-12-31">null</Value>
                        <Value tTimei="2001-06-28" tTimef="9999-12-31" vTimei="2001-05-05" vTimef="2001-06-27">null</Value>
                        <Value tTimei="2001-06-28" tTimef="9999-12-31" vTimei="2001-06-28" vTimef="9999-12-31">10,2,2</Value>
                    </successor>
                    <configuration>
                        <Value>false</Value>
                    </configuration>
                    <status>
                        <Value tTimei="2001-05-05" tTimef="2001-05-09" vTimei="2001-05-05" vTimef="9999-12-31">Working</Value>
                        <Value tTimei="2001-05-10" tTimef="9999-12-31" vTimei="2001-05-05" vTimef="2001-06-20">Working</Value>
                        <Value tTimei="2001-06-21" tTimef="9999-12-31" vTimei="2001-06-21" vTimef="9999-12-31">Stable</Value>
                    </status>
                </TemporalVersion>
                <Attributes>
                    <duracaoBateria>
                        <Value tTimei="2001-05-05" tTimef="2001-06-04" vTimei="2001-05-05" vTimef="9999-12-31">180</Value>
                        <Value tTimei="2001-06-05" tTimef="9999-12-31" vTimei="2001-05-05" vTimef="2001-06-04">180</Value>
                        <Value tTimei="2001-05-05" tTimef="2001-06-13" vTimei="2001-06-05" vTimef="9999-12-31">220</Value>
                        <Value tTimei="2001-06-14" tTimef="9999-12-31" vTimei="2001-06-05" vTimef="2001-06-13">220</Value>
                        <Value tTimei="2001-06-14" tTimef="9999-12-31" vTimei="2001-06-14" vTimef="9999-12-31">180</Value>
                    </duracaoBateria>
                    <dispositivoApontador>
                        <Value tTimei="2001-05-05" tTimef="2001-06-07" vTimei="2001-05-05" vTimef="9999-12-31">Touchpad</Value>
                        <Value tTimei="2001-06-08" tTimef="9999-12-31" vTimei="2001-05-05" vTimef="2001-06-07">Touchpad</Value>
                        <Value tTimei="2001-06-08" tTimef="9999-12-31" vTimei="2001-06-08" vTimef="9999-12-31">Accupoint</Value>
                    </dispositivoApontador>
                </Attributes>
            </TemporalObject>
        </Object>
        <Object tvOID="10,2,2" name="E500B" class="Notebook" type="Version">
            <TemporalObject>
                <alive>
                    <Value tTimei="2001-06-28" tTimef="9999-12-31" vTimei="2001-06-28" vTimef="9999-12-31">true</Value>
                </alive>
                <TemporalVersion vocOID="40,3,1">
                    <ascendant>
                        <Value tTimei="2001-06-28" tTimef="2001-07-04" vTimei="2001-06-28" vTimef="9999-12-31">10,1,2</Value>
                        <Value tTimei="2001-07-05" tTimef="9999-12-31" vTimei="2001-06-28" vTimef="2001-07-04">10,1,2</Value>
                        <Value tTimei="2001-07-05" tTimef="2001-07-06" vTimei="2001-07-05" vTimef="9999-12-31">10,1,3</Value>
                        <Value tTimei="2001-07-07" tTimef="9999-12-31" vTimei="2001-07-05" vTimef="2001-07-06">10,1,3</Value>
                        <Value tTimei="2001-07-07" tTimef="9999-12-31" vTimei="2001-07-07" vTimef="9999-12-31">null</Value>
                    </ascendant>
                    <descendant>
                        <Value tTimei="2001-06-28" tTimef="9999-12-31" vTimei="2001-06-28" vTimef="9999-12-31">null</Value>
                    </descendant>
                    <predecessor>
                        <Value>10,2,1</Value>
                    </predecessor>
                    <successor>
                        <Value tTimei="2001-06-28" tTimef="2001-07-01" vTimei="2001-06-28" vTimef="9999-12-31">null</Value>
                        <Value tTimei="2001-07-02" tTimef="9999-12-31" vTimei="2001-06-28" vTimef="2001-07-01">null</Value>
                        <Value tTimei="2001-07-02" tTimef="9999-12-31" vTimei="2001-07-02" vTimef="9999-12-31">10,2,3</Value>
                    </successor>
                    <configuration>
                        <Value>false</Value>
                    </configuration>
                    <status>
                        <Value tTimei="2001-06-28" tTimef="2001-06-29" vTimei="2001-06-28" vTimef="9999-12-31">Working</Value>
                        <Value tTimei="2001-06-30" tTimef="9999-12-31" vTimei="2001-06-28" vTimef="2001-06-29">Working</Value>
                        <Value tTimei="2001-06-30" tTimef="9999-12-31" vTimei="2001-06-30" vTimef="9999-12-31">Stable</Value>
                    </status>
                </TemporalVersion>
                <Attributes>
                    <duracaoBateria>
                        <Value tTimei="2001-06-28" tTimef="2001-06-28" vTimei="2001-06-28" vTimef="9999-12-31">180</Value>
                        <Value tTimei="2001-06-29" tTimef="9999-12-31" vTimei="2001-06-28" vTimef="2001-06-28">180</Value>
                        <Value tTimei="2001-06-29" tTimef="9999-12-31" vTimei="2001-06-29" vTimef="9999-12-31">200</Value>
                    </duracaoBateria>
                    <dispositivoApontador>
                        <Value tTimei="2001-06-28" tTimef="9999-12-31" vTimei="2001-06-28" vTimef="9999-12-31">Accupoint</Value>
                    </dispositivoApontador>
                </Attributes>
            </TemporalObject>
        </Object>
        <Object tvOID="10,2,3" name="E500C" class="Notebook" type="Version">
            <TemporalObject>
                <alive>
                    <Value tTimei="2001-07-02" tTimef="9999-12-31" vTimei="2001-07-02" vTimef="9999-12-31">true</Value>
                </alive>
                <TemporalVersion vocOID="40,3,1">
                    <ascendant>
                        <Value tTimei="2001-07-02" tTimef="2001-07-06" vTimei="2001-07-02" vTimef="9999-12-31">null</Value>
                        <Value tTimei="2001-07-07" tTimef="9999-12-31" vTimei="2001-07-02" vTimef="2001-07-06">null</Value>
                        <Value tTimei="2001-07-07" tTimef="2001-07-10" vTimei="2001-07-07" vTimef="9999-12-31">10,1,3</Value>
                        <Value tTimei="2001-07-11" tTimef="9999-12-31" vTimei="2001-07-07" vTimef="2001-07-10">10,1,3</Value>
```

```
                    <Value tTimei="2001-07-11" tTimef="9999-12-31" vTimei="2001-07-11" vTimef="9999-12-31">10,1,4</Value>
                </ascendant>
                <descendant>
                    <Value tTimei="2001-07-02" tTimef="9999-12-31" vTimei="2001-07-02" vTimef="9999-12-31">null</Value>
                </descendant>
                <predecessor>
                    <Value>10,2,2</Value>
                </predecessor>
                <successor>
                    <Value tTimei="2001-07-02" tTimef="9999-12-31" vTimei="2001-07-02" vTimef="9999-12-31">null</Value>
                </successor>
                <configuration>
                    <Value>false</Value>
                </configuration>
                <status>
                    <Value tTimei="2001-07-02" tTimef="2001-07-15" vTimei="2001-07-02" vTimef="9999-12-31">Working</Value>
                    <Value tTimei="2001-07-16" tTimef="9999-12-31" vTimei="2001-07-02" vTimef="2001-07-15">Working</Value>
                    <Value tTimei="2001-07-16" tTimef="2001-07-20" vTimei="2001-07-16" vTimef="9999-12-31">Stable</Value>
                    <Value tTimei="2001-07-21" tTimef="9999-12-31" vTimei="2001-07-16" vTimef="2001-07-20">Stable</Value>
                    <Value tTimei="2001-07-21" tTimef="9999-12-31" vTimei="2001-07-21" vTimef="9999-12-31">Consolid.</Value>
                </status>
            </TemporalVersion>
            <Attributes>
                <duracaoBateria>
                    <Value tTimei="2001-07-02" tTimef="2001-07-09" vTimei="2001-07-02" vTimef="9999-12-31">200</Value>
                    <Value tTimei="2001-07-10" tTimef="9999-12-31" vTimei="2001-07-02" vTimef="2001-07-09">200</Value>
                    <Value tTimei="2001-07-10" tTimef="9999-12-31" vTimei="2001-07-10" vTimef="9999-12-31">220</Value>
                </duracaoBateria>
                <dispositivoApontador>
                    <Value tTimei="2001-07-02" tTimef="9999-12-31" vTimei="2001-07-02" vTimef="9999-12-31">Accupoint</Value>
                </dispositivoApontador>
            </Attributes>
        </TemporalObject>
    </Object>
    <Object tvOID="11,2,0" name="E700" class="Notebook" type="VersionedObject">
        <TemporalObject>
            <alive>
                <Value tTimei="2001-10-01" tTimef="9999-12-31" vTimei="2001-10-01" vTimef="9999-12-31">true</Value>
            </alive>
            <TemporalVersion vocOID="45,3,1">
                <ascendant>
                    <Value tTimei="2001-10-01" tTimef="9999-12-31" vTimei="2001-10-01" vTimef="9999-12-31">11,1,1</Value>
                </ascendant>
                <descendant>
                    <Value tTimei="2001-10-01" tTimef="9999-12-31" vTimei="2001-10-01" vTimef="9999-12-31">null</Value>
                </descendant>
                <predecessor>
                    <Value>11,2,1</Value>
                </predecessor>
                <successor>
                    <Value tTimei="2001-10-01" tTimef="9999-12-31" vTimei="2001-10-01" vTimef="9999-12-31">null</Value>
                </successor>
                <configuration>
                    <Value>false</Value>
                </configuration>
                <status>
                    <Value tTimei="2001-10-01" tTimef="9999-12-31" vTimei="2001-10-01" vTimef="9999-12-31">Working</Value>
                </status>
            </TemporalVersion>
            <Attributes>
                <duracaoBateria>
                    <Value tTimei="2001-10-01" tTimef="9999-12-31" vTimei="2001-10-01" vTimef="9999-12-31">220</Value>
                </duracaoBateria>
                <dispositivoApontador>
                    <Value tTimei="2001-10-01" tTimef="9999-12-31" vTimei="2001-10-01" vTimef="9999-12-31">Touchpad</Value>
                </dispositivoApontador>
            </Attributes>
        </TemporalObject>
    </Object>
    <Object tvOID="11,2,1" name="E700X" class="Notebook" type="Version">
        <TemporalObject>
            <alive>
                <Value tTimei="2001-05-05" tTimef="9999-12-31" vTimei="2001-05-05" vTimef="9999-12-31">true</Value>
            </alive>
            <TemporalVersion vocOID="45,3,1">
                <ascendant>
                    <Value tTimei="2001-05-05" tTimef="2001-08-24" vTimei="2001-05-05" vTimef="9999-12-31">null</Value>
                    <Value tTimei="2001-08-25" tTimef="9999-12-31" vTimei="2001-05-05" vTimef="2001-08-24">null</Value>
                    <Value tTimei="2001-25-05" tTimef="2001-09-14" vTimei="2001-08-25" vTimef="9999-12-31">11,1,1</Value>
                    <Value tTimei="2001-09-15" tTimef="9999-12-31" vTimei="2001-08-25" vTimef="2001-09-14">11,1,1</Value>
                    <Value tTimei="2001-09-15" tTimef="9999-12-31" vTimei="2001-09-15" vTimef="9999-12-31">null</Value>
                </ascendant>
                <descendant>
                    <Value tTimei="2001-05-05" tTimef="9999-12-31" vTimei="2001-05-05" vTimef="9999-12-31">null</Value>
                </descendant>
                <predecessor>
                    <Value>null</Value>
                </predecessor>
                <successor>
                    <Value tTimei="2001-05-05" tTimef="2001-08-31" vTimei="2001-05-05" vTimef="9999-12-31">null</Value>
                    <Value tTimei="2001-09-01" tTimef="9999-12-31" vTimei="2001-05-05" vTimef="2001-08-31">null</Value>
                    <Value tTimei="2001-09-01" tTimef="2001-09-30" vTimei="2001-09-01" vTimef="9999-12-31">11,2,2</Value>
                    <Value tTimei="2001-10-01" tTimef="9999-12-31" vTimei="2001-09-01" vTimef="2001-09-30">11,2,2</Value>
```

```
                    <Value tTimei="2001-10-01" tTimef="9999-12-31" vTimei="2001-10-01" vTimef="9999-12-31">11,2,2</Value>
                    <Value tTimei="2001-10-01" tTimef="9999-12-31" vTimei="2001-10-01" vTimef="9999-12-31">11,2,3</Value>
                </successor>
                <configuration>
                    <Value>false</Value>
                </configuration>
                <status>
                    <Value tTimei="2001-05-05" tTimef="2001-06-25" vTimei="2001-05-05" vTimef="9999-12-31">Working</Value>
                    <Value tTimei="2001-06-26" tTimef="9999-12-31" vTimei="2001-05-05" vTimef="2001-06-25">Working</Value>
                    <Value tTimei="2001-06-26" tTimef="9999-12-31" vTimei="2001-06-26" vTimef="2001-07-20">Stable</Value>
                    <Value tTimei="2001-07-21" tTimef="9999-12-31" vTimei="2001-07-21" vTimef="9999-12-31">Consolid.</Value>
                </status>
            </TemporalVersion>
            <Attributes>
                <duracaoBateria>
                    <Value tTimei="2001-05-05" tTimef="2001-06-07" vTimei="2001-05-05" vTimef="9999-12-31">250</Value>
                    <Value tTimei="2001-06-08" tTimef="9999-12-31" vTimei="2001-05-05" vTimef="2001-06-07">250</Value>
                    <Value tTimei="2001-06-08" tTimef="9999-12-31" vTimei="2001-06-08" vTimef="9999-12-31">220</Value>
                </duracaoBateria>
                <dispositivoApontador>
                    <Value tTimei="2001-05-05" tTimef="9999-12-31" vTimei="2001-05-05" vTimef="9999-12-31">Touchpad</Value>
                </dispositivoApontador>
            </Attributes>
        </TemporalObject>
    </Object>
    <Object tvOID="11,2,2" name="E700Y" class="Notebook" type="Version">
        <TemporalObject>
            <alive>
                <Value tTimei="2001-09-01" tTimef="9999-12-31" vTimei="2001-09-01" vTimef="9999-12-31">true</Value>
            </alive>
            <TemporalVersion vocOID="45,3,1">
                <ascendant>
                    <Value tTimei="2001-09-01" tTimef="2001-09-14" vTimei="2001-09-01" vTimef="9999-12-31">null</Value>
                    <Value tTimei="2001-09-15" tTimef="9999-12-31" vTimei="2001-09-01" vTimef="2001-09-14">null</Value>
                    <Value tTimei="2001-09-15" tTimef="2001-09-30" vTimei="2001-09-15" vTimef="9999-12-31">11,1,1</Value>
                    <Value tTimei="2001-10-01" tTimef="9999-12-31" vTimei="2001-09-15" vTimef="2001-09-30">11,1,1</Value>
                    <Value tTimei="2001-10-01" tTimef="9999-12-31" vTimei="2001-10-01" vTimef="9999-12-31">null</Value>
                </ascendant>
                <descendant>
                    <Value tTimei="2001-09-01" tTimef="9999-12-31" vTimei="2001-09-01" vTimef="9999-12-31">null</Value>
                </descendant>
                <predecessor>
                    <Value>11,2,1</Value>
                </predecessor>
                <successor>
                    <Value tTimei="2001-09-01" tTimef="9999-12-31" vTimei="2001-09-01" vTimef="9999-12-31">null</Value>
                </successor>
                <configuration>
                    <Value>false</Value>
                </configuration>
                <status>
                    <Value tTimei="2001-09-01" tTimef="2001-09-10" vTimei="2001-09-01" vTimef="9999-12-31">Working</Value>
                    <Value tTimei="2001-09-11" tTimef="9999-12-31" vTimei="2001-09-01" vTimef="2001-09-10">Working</Value>
                    <Value tTimei="2001-09-11" tTimef="9999-12-31" vTimei="2001-09-11" vTimef="9999-12-31">Stable</Value>
                </status>
            </TemporalVersion>
            <Attributes>
                <duracaoBateria>
                    <Value tTimei="2001-09-01" tTimef="2001-09-08" vTimei="2001-09-01" vTimef="9999-12-31">220</Value>
                    <Value tTimei="2001-09-09" tTimef="9999-12-31" vTimei="2001-09-01" vTimef="2001-09-08">220</Value>
                    <Value tTimei="2001-09-09" tTimef="9999-12-31" vTimei="2001-09-09" vTimef="9999-12-31">180</Value>
                </duracaoBateria>
                <dispositivoApontador>
                    <Value tTimei="2001-09-01" tTimef="2001-09-05" vTimei="2001-09-01" vTimef="9999-12-31">Touchpad</Value>
                    <Value tTimei="2001-09-06" tTimef="9999-12-31" vTimei="2001-09-01" vTimef="2001-09-05">Touchpad</Value>
                    <Value tTimei="2001-09-06" tTimef="9999-12-31" vTimei="2001-09-06" vTimef="9999-12-31">Accupoint</Value>
                </dispositivoApontador>
            </Attributes>
        </TemporalObject>
    </Object>
    <Object tvOID="11,2,3" name="E700Z" class="Notebook" type="Version">
        <TemporalObject>
            <alive>
                <Value tTimei="2001-10-01" tTimef="9999-12-31" vTimei="2001-10-01" vTimef="9999-12-31">true</Value>
            </alive>
            <TemporalVersion vocOID="45,3,1">
                <ascendant>
                    <Value tTimei="2001-10-01" tTimef="9999-12-31" vTimei="2001-10-01" vTimef="9999-12-31">11,1,1</Value>
                </ascendant>
                <descendant>
                    <Value tTimei="2001-10-01" tTimef="9999-12-31" vTimei="2001-10-01" vTimef="9999-12-31">null</Value>
                </descendant>
                <predecessor>
                    <Value>11,2,1</Value>
                </predecessor>
                <successor>
                    <Value tTimei="2001-10-01" tTimef="9999-12-31" vTimei="2001-10-01" vTimef="9999-12-31">null</Value>
                </successor>
                <configuration>
                    <Value>false</Value>
                </configuration>
                <status>
                    <Value tTimei="2001-10-01" tTimef="9999-12-31" vTimei="2001-10-01" vTimef="9999-12-31">Working</Value>
```

```
                </status>
            </TemporalVersion>
            <Attributes>
                <duracaoBateria>
                    <Value tTimei="2001-10-01" tTimef="9999-12-31" vTimei="2001-10-01" vTimef="9999-12-31">220</Value>
                </duracaoBateria>
                <dispositivoApontador>
                    <Value tTimei="2001-10-01" tTimef="9999-12-31" vTimei="2001-10-01" vTimef="9999-12-31">Touchpad</Value>
                </dispositivoApontador>
            </Attributes>
        </TemporalObject>
    </Object>
    <Object tvOID="30,3,1" class="VersionedObjectControl">
        <TemporalObject>
            <alive>
                <Value tTimei="2001-06-21" tTimef="9999-12-31" vTimei="2001-06-21" vTimef="9999-12-31">true</Value>
            </alive>
            <VersionedObjectControl>
                <configurationCount>
                    <Value tTimei="2001-06-21" tTimef="9999-12-31" vTimei="2001-06-21" vTimef="9999-12-31">0</Value>
                </configurationCount>
                <currentVersion>
                    <Value tTimei="2001-06-21" tTimef="2001-06-24" vTimei="2001-06-21" vTimef="9999-12-31">10,1,2</Value>
                    <Value tTimei="2001-06-25" tTimef="9999-12-31" vTimei="2001-06-21" vTimef="2001-06-24">10,1,2</Value>
                    <Value tTimei="2001-06-25" tTimef="2001-07-05" vTimei="2001-06-25" vTimef="9999-12-31">10,1,3</Value>
                    <Value tTimei="2001-07-06" tTimef="9999-12-31" vTimei="2001-06-25" vTimef="2001-07-05">10,1,3</Value>
                    <Value tTimei="2001-07-06" tTimef="9999-12-31" vTimei="2001-07-06" vTimef="9999-12-31">10,1,4</Value>
                </currentVersion>
                <firstVersion>
                    <Value tTimei="2001-06-21" tTimef="9999-12-31" vTimei="2001-06-21" vTimef="9999-12-31">10,1,1</Value>
                </firstVersion>
                <lastVersion>
                    <Value tTimei="2001-06-21" tTimef="2001-06-24" vTimei="2001-06-21" vTimef="9999-12-31">10,1,2</Value>
                    <Value tTimei="2001-06-25" tTimef="9999-12-31" vTimei="2001-06-21" vTimef="2001-06-24">10,1,2</Value>
                    <Value tTimei="2001-06-25" tTimef="2001-07-05" vTimei="2001-06-25" vTimef="9999-12-31">10,1,3</Value>
                    <Value tTimei="2001-07-06" tTimef="9999-12-31" vTimei="2001-06-25" vTimef="2001-07-05">10,1,3</Value>
                    <Value tTimei="2001-07-06" tTimef="9999-12-31" vTimei="2001-07-06" vTimef="9999-12-31">10,1,4</Value>
                </lastVersion>
                <nextVersionNumber>
                    <Value>220</Value>
                </nextVersionNumber>
                <userCurrentFlag>
                    <Value tTimei="2001-06-21" tTimef="9999-12-31" vTimei="2001-06-21" vTimef="9999-12-31">false</Value>
                </userCurrentFlag>
                <versionCount>
                    <Value tTimei="2001-06-21" tTimef="2001-06-24" vTimei="2001-06-21" vTimef="9999-12-31">2</Value>
                    <Value tTimei="2001-06-25" tTimef="9999-12-31" vTimei="2001-06-21" vTimef="2001-06-24">2</Value>
                    <Value tTimei="2001-06-25" tTimef="2001-07-05" vTimei="2001-06-25" vTimef="9999-12-31">3</Value>
                    <Value tTimei="2001-07-06" tTimef="9999-12-31" vTimei="2001-06-25" vTimef="2001-07-05">3</Value>
                    <Value tTimei="2001-07-06" tTimef="9999-12-31" vTimei="2001-07-06" vTimef="9999-12-31">4</Value>
                </versionCount>
            </VersionedObjectControl>
        </TemporalObject>
    </Object>
    <Object tvOID="35,3,1" class="VersionedObjectControl">
        <TemporalObject>
            <alive>
                <Value tTimei="2001-10-01" tTimef="9999-12-31" vTimei="2001-10-01" vTimef="9999-12-31">true</Value>
            </alive>
            <VersionedObjectControl>
                <configurationCount>
                    <Value tTimei="2001-06-21" tTimef="9999-12-31" vTimei="2001-06-21" vTimef="9999-12-31">0</Value>
                </configurationCount>
                <currentVersion>
                    <Value tTimei="2001-10-01" tTimef="9999-12-31" vTimei="2001-10-01" vTimef="9999-12-31">11,1,2</Value>
                </currentVersion>
                <firstVersion>
                    <Value tTimei="2001-10-01" tTimef="9999-12-31" vTimei="2001-10-01" vTimef="9999-12-31">11,1,1</Value>
                </firstVersion>
                <lastVersion>
                    <Value tTimei="2001-10-01" tTimef="9999-12-31" vTimei="2001-10-01" vTimef="9999-12-31">11,1,2</Value>
                </lastVersion>
                <nextVersionNumber>
                    <Value>3</Value>
                </nextVersionNumber>
                <userCurrentFlag>
                    <Value tTimei="2001-10-01" tTimef="9999-12-31" vTimei="2001-10-01" vTimef="9999-12-31">false</Value>
                </userCurrentFlag>
                <versionCount>
                    <Value tTimei="2001-10-01" tTimef="9999-12-31" vTimei="2001-10-01" vTimef="9999-12-31">2</Value>
                </versionCount>
            </VersionedObjectControl>
        </TemporalObject>
    </Object>
    <Object tvOID="40,3,1" class="VersionedObjectControl">
        <TemporalObject>
            <alive>
                <Value tTimei="2001-06-28" tTimef="9999-12-31" vTimei="2001-06-28" vTimef="9999-12-31">true</Value>
            </alive>
            <VersionedObjectControl>
                <configurationCount>
                    <Value tTimei="2001-06-28" tTimef="9999-12-31" vTimei="2001-06-28" vTimef="9999-12-31">0</Value>
                </configurationCount>
```

```
                        <currentVersion>
                            <Value tTimei="2001-06-28" tTimef="2001-07-01" vTimei="2001-06-28" vTimef="9999-12-31">10,2,2</Value>
                            <Value tTimei="2001-07-02" tTimef="9999-12-31" vTimei="2001-06-28" vTimef="2001-07-01">10,2,2</Value>
                            <Value tTimei="2001-07-02" tTimef="9999-12-31" vTimei="2001-07-02" vTimef="9999-12-31">10,2,3</Value>
                        </currentVersion>
                        <firstVersion>
                            <Value tTimei="2001-06-28" tTimef="9999-12-31" vTimei="2001-06-28" vTimef="9999-12-31">10,2,1</Value>
                        </firstVersion>
                        <lastVersion>
                            <Value tTimei="2001-06-28" tTimef="2001-07-01" vTimei="2001-06-28" vTimef="9999-12-31">10,2,2</Value>
                            <Value tTimei="2001-07-02" tTimef="9999-12-31" vTimei="2001-06-28" vTimef="2001-07-01">10,2,2</Value>
                            <Value tTimei="2001-07-02" tTimef="9999-12-31" vTimei="2001-07-02" vTimef="9999-12-31">10,2,3</Value>
                        </lastVersion>
                        <nextVersionNumber>
                            <Value>250</Value>
                        </nextVersionNumber>
                        <userCurrentFlag>
                            <Value tTimei="2001-06-28" tTimef="9999-12-31" vTimei="2001-06-28" vTimef="9999-12-31">false</Value>
                        </userCurrentFlag>
                        <versionCount>
                            <Value tTimei="2001-06-28" tTimef="2001-07-01" vTimei="2001-06-28" vTimef="9999-12-31">2</Value>
                            <Value tTimei="2001-07-02" tTimef="9999-12-31" vTimei="2001-06-28" vTimef="2001-07-01">2</Value>
                            <Value tTimei="2001-07-02" tTimef="9999-12-31" vTimei="2001-07-02" vTimef="9999-12-31">3</Value>
                        </versionCount>
                    </VersionedObjectControl>
                </TemporalObject>
            </Object>
            <Object tvOID="45,3,1" class="VersionedObjectControl">
                <TemporalObject>
                    <alive>
                        <Value tTimei="2001-09-01" tTimef="9999-12-31" vTimei="2001-09-01" vTimef="9999-12-31">true</Value>
                    </alive>
                    <VersionedObjectControl>
                        <configurationCount>
                            <Value tTimei="2001-09-01" tTimef="9999-12-31" vTimei="2001-09-01" vTimef="9999-12-31">0</Value>
                        </configurationCount>
                        <currentVersion>
                            <Value tTimei="2001-09-01" tTimef="2001-09-30" vTimei="2001-09-01" vTimef="9999-12-31">11,2,2</Value>
                            <Value tTimei="2001-10-01" tTimef="9999-12-31" vTimei="2001-09-01" vTimef="2001-09-30">11,2,2</Value>
                            <Value tTimei="2001-10-01" tTimef="9999-12-31" vTimei="2001-10-01" vTimef="9999-12-31">11,2,3</Value>
                        </currentVersion>
                        <firstVersion>
                            <Value tTimei="2001-09-01" tTimef="9999-12-31" vTimei="2001-09-01" vTimef="9999-12-31">11,2,1</Value>
                        </firstVersion>
                        <lastVersion>
                            <Value tTimei="2001-09-01" tTimef="2001-09-30" vTimei="2001-09-01" vTimef="9999-12-31">11,2,2</Value>
                            <Value tTimei="2001-10-01" tTimef="9999-12-31" vTimei="2001-09-01" vTimef="2001-09-30">11,2,2</Value>
                            <Value tTimei="2001-10-01" tTimef="9999-12-31" vTimei="2001-10-01" vTimef="9999-12-31">11,2,3</Value>
                        </lastVersion>
                        <nextVersionNumber>
                            <Value>250</Value>
                        </nextVersionNumber>
                        <userCurrentFlag>
                            <Value tTimei="2001-09-01" tTimef="9999-12-31" vTimei="2001-09-01" vTimef="9999-12-31">false</Value>
                        </userCurrentFlag>
                        <versionCount>
                            <Value tTimei="2001-09-01" tTimef="2001-09-30" vTimei="2001-09-01" vTimef="9999-12-31">2</Value>
                            <Value tTimei="2001-10-01" tTimef="9999-12-31" vTimei="2001-09-01" vTimef="2001-09-30">2</Value>
                            <Value tTimei="2001-10-01" tTimef="9999-12-31" vTimei="2001-10-01" vTimef="9999-12-31">3</Value>
                        </versionCount>
                    </VersionedObjectControl>
                </TemporalObject>
            </Object>
        </TemporalObjects>
    </ObjectInstances>
```

Referências

[ABI 97a] ABITEBOUL, S. Querying Semistructured Data. In: INTERNATIONAL CONFERENCE ON DATABASE THEORY, ICDT, 6., 1997, Delphi. **Database Theory:** proceedings. Berlin: Springer-Verlag, 1997. p.1-18.

[ABI 97b] ABITEBOUL, S.; QUASS, D.; MCHUGH, J.; WIDOM, J.; WIENER, J.L. The Lorel Query Language for Semistructured Data. **International Journal on Digital Libraries**, [S.l.], v.1, n.1, p.68-88, 1997.

[ABI 98] ABITEBOUL, S.; WIDOM, J.; LAHIRI, T. A Unified Approach for Querying Structured Data and XML. In: THE QUERY LANGUAGES WORKSHOP, QL, 1998, Boston. **Proceedings**… Boston, Massachussets, USA: [s.n.], 1998. Disponível em: <http://www.w3.org/ TandS/QL/ QL98/pp/serge.html. Acesso em: dez.2001>.

[AGR 91] AGRAWAL, R.; BUROFF, S.; GEHANI, N.; SHASHA, D. Object versioning in Ode. In: INTERNATIONAL CONFERENCE ON DATA ENGINEERING, 7., 1991, Tokyo. **Proceedings**… Los Alamitos: IEEE Computer Society Press, 1999. p. 446-455.

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

[ANT 97] ANTUNES, D.; HEUSER, C.; EDELWEISS, N. TempER: uma abordagem para modelagem temporal de banco de dados. **Revista de Informática Teórica e Aplicada**, Porto Alegre, v. 4, n. 1, p. 49-85, 1997.

[ATZ 98] ATZENI, P.; MECCA, G.; MERIALDO, P. Do we really need a new query language for XML? In: THE QUERY LANGUAGES WORKSHOP, QL, 1998, Boston. **Proceedings**… Boston, Massachussets, USA: [s.n.], 1998. Disponível em: <http://w3.org/ TandS/QL/QL98/pp/MeMAql98.html>. Acesso em: dez.2001.

[BON 2000] BONIFATI, A.; CERI, S. Comparative Analysis of Five XML Query Languages. **ACM SIGMOD Record**, New York, v. 29, n. 1, p. 68-79, 2000. Disponível em: <http://www.toriisoft.com/tech-papers/ xmlsurvey.pdf>. Acesso em: nov.2001.

[BOU 2001] BOURRET, R. **Mapping W3C Schemas to Object Schemas to Relational Schemas**. Disponível em: <http://www.rpbourret.com/xml/ schemamap.htm>. Acesso em: dez.2001.

[BRA 2000] BRAY, T.; PAOLI, J.; MALER. E.; SPERBERG-MCQUEEN, C. M. **Extensible Markup Language (XML) 1.0 (Second Edition):** W3C Recommendation. 2000. Disponível em: <http://w3.org/TR/2000/REC-xml-20001006>. Acesso em: dez.2001.

[BUN 97] BUNEMAN, P. Semistructured Data. In: SIGMOD INTERNATIONAL SYMPOSIUM ON PRINCIPLES OF DATABASE SYSTEMS, PODS, 16., 1997, Tucson. **Proceedings**… Tucson, Arizona, USA: [s.n.], 1997. p.117-121.

[BUN 98] BUNEMAN, P.; DEUTSCH, A.; FAN, W.; LIEFKE, H.; SAHUGUET, A.; TAN, W.C. Beyond XML query languages In: THE QUERY LANGUAGES WORKSHOP, QL, 1998, Boston. **Proceedings**… Boston, Massachussets, USA: [s.n.], 1998. Disponível em: <http://www.cis.upenn.edu/%7Eliefke/papers/ql98.ps>. Acesso em: dez.2001.

[BRA 2000] BRAY, T.; PAOLI, J.; MALER. E.; SPERBERG-MCQUEEN, C. M. **Extensible Markup Language (XML) 1.0 (Second Edition).** W3C Recommendation. 2000. Disponível em: <http://w3.org/TR/2000/REC-xml-20001006>. Acesso em: nov.2001.

[CHA 2000] CHAMBERLIN, D.; ROBIE, J.; FLORESCU, D. Quilt: An XML Query Language for Heterogeneous Data Sources. In: INTERNATIONAL WORKSHOP ON THE WEB AND DATABASES, WebDB, 2000, Dallas. **The Web and Databases:** proceedings. Berlin: Springer-Verlag, 2000.

[CHA 2001] CHAMBERLIN, D.; CLARK, J.; FLORESCU, D.; ROBIE, J.; SIMÉON, J.; STEFANESCU, M. **XQuery 1.0:** An Query Language for XML: W3C Working Draft. 2001. Disponível em: <http://www.w3.org/TR/xquery>. Acesso em: nov.2001.

[CHA 2001a] CHAMBERLIN, D.; FANKHAUSER, P.; MARCHIORI, M.; ROBIE, J. **XML Query Requirements**. W3C Working Draft. 2001. Disponível em: <http://www.w3.org/TR/xmlquery-req>. Acesso em: nov.2001.

[CHI 2000] CHIEN, S. Y.; TSOTRAS, V. J.; ZANIOLO, C. **A Comparative Study of Version Management Schemes for XML Documents.** TimeCenter Technical Report TR-51. Disponível em: <http://www.cs.auc.dk/research/DP/tdb/TimeCenter/TimeCenterPublications/TR-51.pdf>. Acesso em: dez.2001.

[CHI 2001] CHIEN, S. Y.; TSOTRAS, V. J.; ZANIOLO, C; ZHANG, D. Storing and Querying MultiIversion XML Documents using Durable Node Numbers. In: INTERNATIONAL CONFERENCE ON WEB INFORMATION SYSTEMS ENGINEERING, WISE, 2., 2001, Kyoto. **Proceedings**… Kyoto, Japan: [s.n.], 2001. Disponível em: <http://www.cs.ucr.edu/~donghui/publications/xml.pdf.> Acesso em: dez.2001.

[CAT 97] CATTELL, R.; BARRY, D. **The Object Database Standard:** ODMG 2.0. San Francisco: Morgan Kaufmann, 1997. 270 p.

[CLA 99] CLARK, J.; DEROSE, S. **XML Path Language 1.0.** W3C Recommendation. 1999. Disponível em: <http://w3.org/TR/xpath.htm>. Acesso em: nov.2001.

[DEU 98] DEUTSCH, A.; FERNANDEZ, M.; FLORESCU, D.; LEVY, A.; SUCIU, D. **XML-QL:** A Query Language for XML. Submission to the World Wide Web Consortium. 1998. Disponível em: <http://w3.org/TR/1998/NOTE-xml-ql-19980819>. Acesso em: nov.2001.

[DEU 99] DEUTSCH, A.; FERNANDEZ, M.; FLORESCU, D.; LEVY, A.; SUCIU, D. Querying {XML} Data. **IEEE Data Engineering Bulletin**, [S.l.], v. 22, n. 3, p.10-18, 1999.

[DIV 92] DIVERIO, T. A. et al. **Introdução a Teoria dos Intervalos**. Porto Alegre: PPGC da UFRGS, 1992. 39 p. (RP - 172).

[EDE 93] EDELWEISS, N.; OLIVEIRA, J. P. M. de; PERNICI, B. An Object-Oriented Temporal Model. In: INTERNATIONAL. CONFERENCE ON ADVANCED INFORMATION SYSTEMS ENGINEERING, CAISE, 5., 1993, Paris. **Advanced Information Systems Engineering**. Berlin: Springer-Verlag, 1993. p.397-415.

[FER 99] FERNANDEZ, M.; SIMEON, J.; WADLER, P. **XML Query Languages:** Experiences and Exemplars. 1999. Disponível em: <http://www-db.research.bell-labs.com/user/simeon/xquery.os>. Acesso em: nov.2001.

[FER 2001] FERNANDEZ, M.; MARSH, J. XQuery 1.0 and XPath 2.0 Data Model: **W3C Working Draft**. 2001. Disponível em: <http://www.w3.org/TR/ query-datamodel>. Acesso em: dez.2001.

[GAD 88] GADIA, S. K.; YEUNG, C. S. A Generalized model for a relational temporal database. In: ACM SIGMOD INTERNATIONAL CONFERENCE ON MANAGEMENT OF DATA, SIGMOD, 1988, Chicago. **Proceedings**… New York: ACM, 1988. p. 251-259.

[GOD 99] GOLDMAN, R.; MCHUGH, J.; WIDOM, J. From Semistructured Data to XML: Migrating the Lore Data Model and Query Language. In: INTERNATIONAL WORKSHOP ON THE WEB AND DATABASES WebDB, 2., 1999, Philadelphia. **WebDB'99 [arquivo de computador]**.

[GOL 95] GOLENDZINER, L.; SANTOS, C. S. dos. Versions and configurations in object-oriented database systems: a uniform treatment. In: INTERNATIONAL CONFERENCE ON MANAGEMENT OF DATA. 7., 1995, Pune. **Proceedings**… Pune, India: [s.n.], 1995. p.18-37.

[HÜB 2000] HÜBLER, P. N. **Definição de um Gerenciador para o Modelo de Dados Temporal TF-ORM**. 2000. 108 p. Dissertação (Mestrado em Ciência da Computação) – Instituto de Informática, Universidade Federal do Rio Grande do Sul, Porto Alegre.

[IBM 2000] IBM CORPORATION. **XML Extender Administration and Programming**. United States, 2000. 228 p.

[KAF 92] KÄFER, W.; SCHÖNING, H. Realizing a temporal complex-object data model. ACM SIGMOD INTERNATIONAL CONFERENCE ON MANAGEMENT OF DATA, 1992, San Diego. **Proceedings**… San Diego, USA: [s.n.], 1992. p.266-275.

[LIG 99] LIGHT, R. **Iniciando em XML.** São Paulo: Makron Books, 1999. 404 p.

[LOU 91] LOUCOPOULOS, P.; THEODOULIDIS, C.; WANGLER, B. The entity relationship time model and conceptual rule language. In: INTERNATIONAL CONFERENCE ON THE ENTITY RELATIONSHIP APPROACH, 10., 1991, San Mateo. **Brindging the Gap.** San Mateo: The E/R Institute, 1991.

[MAL 2001] MALHOTRA, A.; MARSH, J.; MELTON, J.; ROBIE, J. **XQuery 1.0 and XPath 2.0 Functions and Operators:** W3C Working Draft. 2001. Disponível em: <http://www.w3.org/TR/xquery-operators>. Acesso em: nov.2001.

[MAN 2001] MANUKYAN, M. G.; KALINICHENKO, L. A. Temporal XML. In: ADVANCES IN DATABASES AND INFORMATION SYSTEMS, ADBIS, 5., 2001, Vilnius. **Proceedings**… Vilnius, Lithuania: [s.n], 2001. Disponivel em: <http://www.science.mii.lt/ADBIS/local1/manuk.pdf>. Acesso em: dez.2001.

[MAR 2000] MARCHAL, B. **XML by Example.** Indianápolis: QUE, 2000. 505 p.

[MOR 2001] MORO, M. M. **Modelo Temporal de Versões**. 2001. 120 p. Dissertação (Mestrado em Ciência da Computação) – Instituto de Informática, Universidade Federal do Rio Grande do Sul, Porto Alegre.

[MOR 2001a] MORO, M. M.; GELATTI, P. C.; GOMES, C. H. P.; ROSSETTI, L. L. F.; ZAUPA, A. P.; EDELWEISS, N.; SANTOS, C. S. **Linguagem de Consultas para o Modelo Temporal de Versões**. Porto Alegre: PPGC da UFRGS, 2001. 92 p. (RP - 308).

[NAV 89] NAVATHE, S. B.; AHMED, R. A. Temporal Relational Model and a Query Language. **Information Sciences**, [S.l.], v. 49 n. 1-3, p. 147-175, Mar. 1989.

[OLK 98] OLKEN, F.; MCCARTHY, J. Requirements and Desiderata for an XML Query Language. In: THE QUERY LANGUAGES WORKSHOP, QL, 1998, Boston. **Proceedings**… Boston, Massachussets, USA: [s.n.], 1998. Disponível em: <http://w3.org/TandS/QL/QL98/pp/xml.ql. reqs.html>. Acesso em: dez.2001.

[QUA 98] QUASS, D. Ten Features Necessary for an XML Query Language. In: THE QUERY LANGUAGES WORKSHOP, QL, 1998, Boston. **Proceedings**… Boston, Massachussets, USA**:** [s.n.], 1998. Disponível em: <http://w3.org/ TandS/QL/QL98/pp/quass.html>. Acesso em: dez.2001.

[ROB 98] ROBIE, J.; LAPP, J.; SCHACH, D. XML Query Language (XQL). In: THE QUERY LANGUAGES WORKSHOP, QL, 1998, Boston. **Proceedings**… Boston, Massachussets, USA: [s.n.], 1998. Disponível em: <http://w3.org/TandS/QL/QL98/pp/xql.html>. Acesso em: nov.2001.

[ROB 99] ROBIE, J. **XQL (XML Query Language):** W3C Recommendation. 1999. Disponível em: <http://www.ibiblio.org/xql/xql-proposal.html>. Acesso em: nov.2001.

[ROB 2000] ROBIE, J.; CHAMBERLIN, D.; FLORESCU, D. Quilt: an XML Query Language. In: XML EUROPE 2000, Paris. **Proceedings**… Paris, France: [s.n.], 2000. Disponível em: <http://www.almaden.ibm.com/cs/ people/ chamberlin/quilt_euro.html>. Acesso em: nov.2001.

[SUC 98a] SUCIU D. An Overview of Semistructured Data. **SIGACT News**, [S.l.], v. 29, n. 4, p.28-38, 1998.

[SUC 98b] SUCIU, D. Semistructured Data and XML. In: INTERNATIONAL CONFERENCE ON FOUNDATIONS OF DATA ORGANIZATION. **Proceedings**… [S.l.:s.n.], 1998. Disponível em: <http://www.research.att.com/~suciu/strudel/external/files/_F593433959.ps>. Acesso em: dez.2001.

[TAN 86] TANSEL, A. U. Adding time dimension to relational model and extending relational algebra. **Information Systems**, [S.l.], v. 11, n. 4, p. 343-355, 1986.

[W3C 2000] WORLD WIDE WEB CONSORTIUM. **XML Query Requirements:** W3C Working Draft. 31 January 2000. Disponível em: <http://www.w3.org/TR/xmlquery-req>. Acesso em: dez.2001.