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

Um meta esquema para especificação do Modelo Temporal de Versões em XML

por

LIALDA LÚCIA FERNANDES ROSSETTI

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, março de 2002.

CIP — CATALOGAÇÃO NA PUBLICAÇÃO

Rossetti, Lialda Lúcia Fernandes

Um meta esquema para especificação do Modelo Temporal de Versões em XML / por Lialda Lúcia Fernandes Rossetti. — Porto Alegre: PPGC da UFRGS, 2002.

182 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.

1. Versão, Modelo Temporal de Versão, objeto-relacional, XML, XSLT, DB2. I. Santos, Clesio Saraiva dos. II. Título.

Agradecimentos

É difícil expressar em palavras a minha gratidão por todos aqueles que, de alguma forma, contribuíram para o desenvolvimento desse trabalho. Esperando ser justa com todos, deixo registrado aqui os meus agradecimentos.

Ao CNPQ pela bolsa fornecida durante esses dois anos.

Ao meu orientador, prof. Clésio, pelo apoio, orientação e incentivo, os quais foram fundamentais para a conclusão do trabalho.

A minha co-orientadora, profa. Nina, pela sua minuciosa leitura na minha dissertação e pelas suas importantes dicas.

Ao grupo de pesquisa ao qual faço parte, por todas as reuniões e discussões que geraram questionamentos, dúvidas e correções nas especificações definidas nesse trabalho. Em especial, gostaria de agradecer ao colega Cláudio Gomes, pelas inúmeras sugestões e contribuições de suma relevância.

A Júlio Bohel da IBM, pela atenção dispendida e por todas as dúvidas esclarecidas em relação ao DB2.

Ao meu amigo Glauber e as minhas amigas Dani e Ma. Dani, que mesmo a distância, estiveram presente em todos os momentos. Obrigada pelo apoio, compreensão e carinho.

A minha amiga Vanessa, por todas as idéias dadas para melhorar esse trabalho, pelas injeções de ânimo dadas nos momentos de dúvidas, por todo apoio dado em situações difíceis, pelas alegrias e tristezas compartilhadas, pela acolhida na sua casa, entre tantas outras coisas. Enfim, obrigada pela sua amizade.

As minhas amigas, Ivana e Letícia, por esses dois anos compartilhando o mesmo lar, as alegrias, as tristezas, as incertezas do futuro, as certezas do dia-a-dia. Pelo ombro amigo que sempre me amparou nos momentos mais difíceis, por todas as críticas, conselhos e sugestões que sempre me fizeram refletir. Pelos nossos inúmeros programas ”lights”de fim de semana, enfim, por tudo que vivemos e que me permitiu amadurecer e a compreender melhor as pessoas. Obrigada por tudo.

Ao meus irmãos, Adroaldo Jr. e Rosaldo, pela presença, mesmo distantes, pelo apoio, pelo carinho e pela torcida. Quero agradecer principalmente ao Rosaldo, pelas ajudas nas pesquisas e traduções, e pelas inúmeras dicas para melhorar o trabalho, solucionar os problemas e lidar com os imprevistos.

Agradeço principalmente aos meus pais, Adroaldo e Rosália, pelo apoio incondicional, pela confiança a mim depositada, pela paciência e carinho dados nos momentos em que mais precisei. Pela presença constante, pelo incentivo,

pela torcida, pelas críticas construtivas e por todas as sugestões que sempre me ajudaram muito. Tenho certeza que nenhuma palavra poderá descrever tudo o que vocês fizeram por mim. A vocês a minha gratidão e o meu amor. Dedico esse trabalho a vocês, meus melhores amigos.

Sumário

Lista de Abreviaturas . . . 7

Lista de Figuras . . . 8

Lista de Tabelas . . . 9

Resumo . . . 10

Abstract . . . 11

1 Introdução . . . 12

2 Estado da Arte . . . 14
2.1 Linguagem EXPRESS . . . 14
2.2 Modelo de Dados PCO . . . 16
2.3 XPERANTO . . . 18
2.4 Conclusão . . . 19

3 Modelo Temporal de Versões . . . 21
3.1 Características do Modelo . . . 21
3.1.1 Características das versões . . . 23
3.1.2 Características temporais . . . 25
3.1.3 Exemplo . . . 26
3.2 Propriedades e Operações das Classes . . . 29
3.3 Linguagem de Definição de Dados Estendida . . . 30
3.4 Conclusão . . . 31

4 Linguagem de Consulta do TVM . . . 34
4.1 Características temporais . . . 34
4.2 Características de versões . . . 38
4.3 Metadados . . . 39
4.4 Exemplo . . . 41
4.5 Conclusão . . . 43

5 *Extensible Stylesheet Language Transformation* - XSLT . . . 44
5.1 Características Gerais . . . 44
5.2 Características Adicionais . . . 48
5.3 Conclusão . . . 49

6 Especificação em XML . . . 50
6.1 Modelo Temporal de Versões . . . 51
6.1.1 Documento XML . . . 52
6.1.2 Sintaxe DB2 . . . 55
6.1.3 Regras de mapeamento . . . 61
6.1.4 Arquivo XSLT . . . 64
6.2 Linguagem de Consulta do TVM . . . 67
6.2.1 Estrutura . . . 67
6.2.2 Exemplos . . . 68

6.3 Conclusão 72
7 Conclusão 74
Anexo A Gramática da XTVQL 76
Anexo B Gramática e Documento do TVM 77
Anexo C XSLT (TVM e DDL) e Resultados da transformação 114
Anexo D Métodos implementados 173
Bibliografia 178

Lista de Abreviaturas

DDL *Data Definition Language*

ODMG *Object Data Management Group*

SQL *Structured Query Language*

TTI Tempo de Transação Inicial

TTF Tempo de Transação Final

TVI Tempo de Validade Inicial

TVF Tempo de Validade Final

TVM *Temporal Versions Model*

TVQL *Temporal Version Query Language*

UML *Unified Modeling Language*

XML *Extensible Markup Language*

XSLT *Extensible Style sheet Language Transformation*

XTVQL *XML Temporal Version Query Language*

W3C *XML Working Group*

Lista de Figuras

FIGURA 2.1 – Herança semântica 17

FIGURA 3.1 – Hierarquia de Classes do TVM 22
FIGURA 3.2 – Diagrama de estados de uma versão 23
FIGURA 3.3 – Tipos de correspondências entre versões 24
FIGURA 3.4 – Classes que modelam os rótulos temporais para propriedades e relacionamentos 26
FIGURA 3.5 – Métodos da classe *TemporalLabel* 26
FIGURA 3.6 – Métodos da classe *InstantAttribute* e *InstantRelationship* . . . 27
FIGURA 3.7 – Métodos da classe *TemporalAttribute* e *TemporalRelationship* . 27
FIGURA 3.8 – Aplicação exemplo 28
FIGURA 3.9 – Especificação das classes *Object*, *OIDt* e *Name* 31
FIGURA 3.10 – Especificação das classes *TemporalObject* e *VersionedObject-Control* 32
FIGURA 3.11 – Especificação da classe *TemporalVersion* 33
FIGURA 3.12 – Linguagem de definição de dados estendida 33

FIGURA 4.1 – Operadores de comparação para intervalos 36
FIGURA 4.2 – Operadores de comparação para instantes e intervalos . . . 37
FIGURA 4.3 – Metadados do Modelo Temporal de Versões 40
FIGURA 4.4 – Histórico da propriedade `valor` da versão `c1` 41

FIGURA 5.1 – Processador XSLT 45
FIGURA 5.2 – Documento XML com tabelas de uma aplicação 46

FIGURA 6.1 – Aplicação 51
FIGURA 6.2 – Exemplo modelo de dados para DB2 55
FIGURA 6.3 – Implementação das classes do usuário 66

Lista de Tabelas

TABELA 2.1 – Resumo mapeamento esquema do BD para XML Schema 19

TABELA 3.1 – Exemplo de operações sobre propriedades temporais . . 29

TABELA 4.1 – Condições temporais para instantes 36
TABELA 4.2 – Condições temporais para intervalos 36
TABELA 4.3 – Condições temporais para instantes e intervalos 37

TABELA 6.1 – Tabela com dados de `pessoa` 57
TABELA 6.2 – Tabela com dados de `empregado` 57
TABELA 6.3 – Tabela com tipos de dados básicos do DB2 61
TABELA 6.4 – Histórico do salário de um empregado 63
TABELA 6.5 – Histórico do salário de um empregado 63
TABELA 6.6 – Histórico do salário de um empregado 63
TABELA 6.7 – Tabela resumo do mapeamento para o DB2 64

Resumo

As aplicações que lidam com dados temporais e versionados podem ser modeladas através do Modelo Temporal de Versões. No entanto, para que se possa utilizar esse modelo, é necessário que bases de dados tradicionais sejam estendidas para bases temporais versionadas, habilitando dessa forma, a manipulação desses dados.

O padrão XML tem sido amplamente utilizado para publicar e trocar dados pela internet. Porém, pode ser utilizado também para a formalização de conceitos, dados, esquemas, entre outros.

Com a especificação do Modelo Temporal de Versões em XML, é possível gerar automaticamente um *script* SQL com as características do modelo, de forma a ser aplicado a um banco de dados, tornando-o apto a trabalhar com os conceitos de tempo e de versão. Para isso, é necessário criar regras de transformação (XSLT), que serão aplicadas às especificações definidas para o modelo. O resultado final (*script* SQL) será executado em uma base de dados que implemente os conceitos de orientação a objetos, transformando essa base em uma base temporal versionada.

Cada banco de dados possui sua própria linguagem de definição de dados. Para gerar o *script* em SQL com as características do Modelo Temporal de Versões, regras de transformação deverão ser definidas para os bancos que utilizarão o modelo, observando sua sintaxe específica. Essas diversas regras serão aplicadas à mesma especificação do modelo em XML. O resultado será o *script* em SQL definido na sintaxe de cada base de dados.

Palavras-chave: Versão, Modelo Temporal de Versão, objeto-relacional, XML, XSLT, DB2.

TITLE: “A META SCHEMA FOR THE SPECIFICATION OF THE TEMPORAL VERSION MODEL IN XML”

Abstract

Applications that deal with temporal and versioned data can be modeled by way of the Temporal Versions Model. However, in order one may be able to use this approach, it is required to extend traditional databases to versioned temporal ones, in such a way that these kinds of data can be handled.

The XML standard has been widely used for publishing and for exchanging data on the Internet. Nonetheless, it would also be suitable to formalize concepts, data, schemes and so on, so forth.

The specification of the version temporal model in XML enables the automatic generation of a SQL script encompassing the model characteristics. Such a script, if applied to a database, enables it to deal with the time and version concepts. That is achieved through the creation of appropriate transformation rules (XSLT), which are applied to the specifications defined for the model. The SQL script resulting from this process is then executed on a database that implements the object-oriented concepts, which turns that base into a versioned temporal one.

Each database has its own data definition language. Therefore, in order to generate a SQL script featuring the version temporal model characteristics, transformation rules must be defined on the basis of the syntax inherent in the database that will use it. These several rules will be applied to the XML specification of the same model, resulting in the SQL script defined on the basis of the data definition language of each database.

Keywords: version, Temporal Version Model, object-relacional, XML, XSLT, DB2.

1 Introdução

Diversas aplicações, nas mais diferentes áreas, necessitam armazenar o histórico e a evolução de seus dados. O histórico dos dados se refere aos valores que um determinado dado assume no decorrer da sua existência. A evolução dos dados se refere às diversas alternativas estabelecidas para um dado. Essas alternativas são denominadas versões, e aplicam-se geralmente a aplicações que lidam com projetos.

Modelos de dados tradicionais não implementam de forma natural os conceitos de tempo e de versão. Portanto, estudos foram realizados de forma a propor modelos específicos para esses conceitos.

Os modelos temporais implementam características essenciais para a recuperação dos dados em um determinado período. Isso é possível através da associação dos tempos de validade e/ou transação aos dados, uso de pontos no tempo e intervalos de tempo, entre outras. Os modelos temporais propostos são geralmente de modelos de dados relacionais [NAV 89, SAR 90, TAN 93], entidade-relacionamento [ANT 97] ou orientados a objetos [EDE 94, ROS 91].

Como visto anteriormente, as versões lidam com alternativas de projetos. Conseqüentemente, as pesquisas realizadas se concentram nas áreas de CAD (*Computer Aided Design*), CASE (*Computer Aided Software Design*) e SCM (*Software Configuration Management*) [AGR 91, CON 98, GOL 95]. No entanto, começa a despontar o interesse do uso de versões em outras áreas, como por exemplo, hipertexto de documentos [BIE 98], gerência de projetos [MOE 94], entre outras.

Os modelos acima citados, trabalham somente com um dos conceitos, tempo ou versão. Visando unir esses dois conceitos, foi proposto em [MOR 2001] o Modelo Temporal de Versões (TVM). Esse modelo é baseado no paradigma de orientação a objetos e permite que o usuário defina aplicações que lidam com tempo e versão.

Como não existe um banco de dados específico que implemente as características definidas no TVM, faz-se necessária a extensão dos bancos de dados tradicionais para armazenar esses dados.

Uma forma de realizar essa extensão é através da utilização do padrão XML [W3C 2000], que tem sido amplamente utilizado para publicação e troca de dados pela WEB. No entanto, sendo essa linguagem um formato universal, é possível utilizá-la também para formalização de conceitos, esquemas, dados, etc.

Várias ferramentas manipulam dados XML. Entre elas pode-se destacar a XSLT [W3C 2001], que é uma linguagem de transformação de documentos XML para outros formatos.

O objetivo desse trabalho consiste na especificação do TVM em XML, resultando dessa forma, na formalização do modelo. Essa especificação é feita através da definição de uma estrutura em XML que represente as classes do Modelo Temporal de Versões com suas propriedades e métodos. Também são especificados os metadados, a linguagem de definição de dados e a linguagem de consulta definidos para o modelo.

Os metadados são utilizados para armazenar os dados específicos de tempo e de versão do modelo e dos esquemas dos usuários. São armazenados, por exemplo, quais propriedades e relacionamentos são temporais, e quais clas- ses são

temporais versionadas, entre outras informações. Esses metadados são necessários para tornar a implementação independente do banco utilizado.

A linguagem de definição dos dados será também especificada em XML para auxiliar o usuário na construção da sua aplicação. A representação em XML da linguagem de consulta é realizada unicamente para completar a especificação do modelo.

O resultado dessa especificação (formalização) do modelo poderá ser utilizado de duas formas: como base para que o usuário possa estender o banco de dados para temporal versionado, ou através da utilização em conjunto com a XSLT.

No primeiro caso, o usuário deverá criar manualmente todas as classes do modelo, suas propriedades e métodos e deverá também criar os metadados. No segundo caso, serão criados arquivos de transformação para o modelo e para a linguagem de definição dos dados. O produto final será um *script* em SQL com as instruções de criação das classes e dos metadados do modelo, e uma documentação em HTML para indicar a sintaxe que deverá ser utilizada pelo usuário para criar seus esquemas. O resultado do *script* gerado poderá sofrer algumas modificações por parte do usuário e deverá ser aplicado em um banco de dados que implemente os conceitos de orientação a objetos.

Como cada banco de dados possui uma sintaxe específica para a definição dos dados, deverá ser criado um arquivo XSLT para cada um desses bancos. Nesse trabalho, foram definidas regras para gerar o *script* e a documentação em HTML para o DB2. No entanto, outras regras poderão ser definidas, por exemplo, para o Oracle e para o SQL Server.

Uma possível utilização da linguagem de consulta em XML seria criar regras de transformação de uma consulta escrita nessa linguagem para gerar uma mesma consulta em OQL ou em SQL. É importante ressaltar, que esse não é o foco do trabalho, apresentando-se portanto, apenas a especificação em XML.

Esse trabalho está dividido da seguinte forma: o capítulo 2 apresenta alguns trabalhos que abordam a especificação de esquemas em XML; o capítulo 3 apresenta o Modelo Temporal de Versões com suas as principais características que serão utilizadas no decorrer desse trabalho; o capítulo 4 descreve a linguagem de consulta definida para o TVM; o capítulo 5 apresenta a linguagem de transformação (XSLT) com as principais funcionalidades utilizadas nesse trabalho; o capítulo 6 apresenta a proposta de especificação em XML do modelo e da linguagem de definição dos dados, as características objeto-relacional do DB2 e as regras definidas para o mapeamento do TVM e da DDL para esse banco de dados.

Os anexos apresentados ao final da dissertação apresentam:

- Anexo 1 : gramática da XTVQL.
- Anexo 2 : gramática e documento do TVM em XML, e especificação da DDL em XML.
- Anexo 3 : XSLT definida para o mapeamento do modelo para o DB2 e *script* com a sintaxe do banco, e XSLT e documentação da DDL.
- Anexo 4 : métodos das classes implementados.

2 Estado da Arte

Com a padronização de XML como linguagem para publicação e troca de informações pela internet, e devido a sua grande aceitação por parte dos usuários de diversas áreas, vários trabalhos vêm sendo desenvolvidos com o intuito de verificar de que forma a comunidade como um todo, pode se beneficiar dessa linguagem. Em relação à área de sistemas de informação, onde esse trabalho está inserido, existe um questionamento comum: como mapear XML para banco de dados [BOU 2001]. No entanto, há interesse também em manipular dados no formato XML e em seguida, inseri-los em um banco de dados, ou ainda, extrair dados de um banco de dados no formato XML.

Grandes avanços já foram realizados nesse sentido e os principais bancos de dados, como por exemplo, Oracle, DB2 e SQL Server, já permitem que dados XML possam ser inseridos, alterados, excluídos e consultados.

Como o objetivo desse trabalho é especificar (mapear) o Modelo Temporal de Versões, definido com base no paradigma de orientação a objetos, em XML, esse capítulo irá apresentar alguns trabalhos relacionados ao mapeamento de esquemas de banco relacionais e objeto-relacionais para essa linguagem. Nas próximas seções, serão apresentados de forma resumida, os trabalhos considerados mais relevantes.

2.1 Linguagem EXPRESS

A linguagem EXPRESS é uma linguagem utilizada para a definição de esquemas. Ela faz parte do padrão STEP (ISO 10303) e é amplamente usada para definir modelos de dados de aplicações industriais de grande escala, incluindo aplicações de manufatura e engenharia [RIV 98].

Está sendo desenvolvido um trabalho que utiliza o padrão XML para codificar tanto a definição de esquemas definidos em EXPRESS quanto as instâncias que estão de acordo com o esquema definido, isto é, representar a semântica dessa linguagem em XML.

Fazendo-se uma pequena comparação entre XML e EXPRESS, observa-se que EXPRESS é uma linguagem poderosa e rica para a definição de uma estrutura para o qual um conjunto de dados deverá estar em conformidade. Os mecanismos oferecidos para a definição de tipos de objetos e suas propriedades, que serão usadas em um determinado conjunto de dados, e as restrições impostas para tais objetos são mais completas do que os mecanismos oferecidos pela DTD em XML. Por outro lado, a sintaxe da instância XML é mais rica do que a sintaxe correspondente em EXPRESS.

Uma das grandes limitações da linguagem EXPRESS deve-se ao fato de ela não ser facilmente interpretada por humanos ou por máquinas, que não tenham prévio conhecimento do esquema utilizado. Em contraste, um documento XML, sem um esquema definido, pode ser extremamente útil para os usuários ou aplicações. Outro ponto positivo que pode ser apresentado é que documentos XML são auto-descritivos. Esse é um dos pontos que permitiram a sua rápida disseminação.

A possibilidade de definir construtores XML, que representem a semântica completa de um esquema EXPRESS, habilita que dados STEP sejam trocados e compartilhados através da WEB, tornando-os acessíveis às ferramentas que manipulam dados XML, tais como XLL e XSLT.

As principais razões que se leva a especificar esses dados em XML [BAR 2001] estão listadas abaixo.

- Ao contrário da EXPRESS, XML é amplamente usado e tem a vantagem de possuir vários softwares disponíveis que processam esses dados.
- XML define uma sintaxe padrão para representar dados estruturados.
- A popularidade da XML pode ajudar a tornar modelos de dados EXPRESS estabelecidos mais acessíveis para novos usuários.

Inicialmente serão apresentadas as características da linguagem EXPRESS e em seguida a sua representação em XML.

A linguagem EXPRESS é uma linguagem de modelagem de dados que combina idéias de entidade-atributo-relacionamento com idéias de modelagem de objetos. Seu conceito primário é o tipo entidade (`entity`), cuja função é modelar o domínio de objetos conceitual ou do mundo real, e a coleção de unidades de informação que os descrevem. Um tipo entidade tem atributos (`attribute`) que modelam as unidades de informação descritiva, e cada atributo tem um tipo de dado que especifica a natureza e valores da unidade de informação. Os tipos de dados podem ser os tipos computacionais comuns (boolean, integer, real, string, enumeration), ou tipos de entidade, ou agregados set, list, array) de qualquer um desses tipos. EXPRESS também suporta tipos definidos, que são novos tipos de dados definidos pelo usuário.

Alguns exemplos serão apresentadas de definições em EXPRESS acompanhadas de sua possível representação em XML. Considere inicialmente a declaração de um novo tipo de dado:

```
TYPE person_name = STRING
END_TYPE

<datatype name="person_name">
   <STRING/>
</datatype>
```

Outro exemplo é a definição de entidades:

```
ENTITY point;
   x, y: REAL
END_ENTITY

<entity id="e1" name="point">
   <attribute name="x">
      <real_value>3.1</real_value>
   </attribute>
   <attribute name="y">
      <real_value>5.7</real_value>
   </attribute>
</datatype>
```

Também é possível definir funções para manipular os atributos das entidades. Nesse caso, não será mostrado como se define o corpo da função por não ser de fácil leitura e por não ser relevante ao trabalho.

```
FUNCTION distance (p1, p2: point): REAL;
  RETURN corpo_da_funcao
END_FUNCTION

<function name="distance">
   <REAL/>
   <parameter name="p1">
      <type href="#point"/>
   </parameter>
   <parameter name="p2">
      <type href="#point"/>
   </parameter>
   <return>
      corpo_da_funcao
   </return>
</function>
```

Como EXPRESS se baseia também no modelo de objetos, é permitido declarar subtipos de entidades. No exemplo ilustrado a seguir, apenas será definida parte da estrutura, a que apresenta a representação dos subtipos em XML. As demais declarações, já foram vistas anteriormente.

```
ENTITY line_segment;
   line_start, line_end: point;
END_ENTITY

ENTITY long_line_segment;
   SUBTYPE OF (line_segment)
END_ENTITY

<typehierarchy>
   <supertype href="#line_segment"/>
   <subtype href="#long_line_segment" />
</typehierarchy>
```

As regras para a criação da estrutura dos documentos XML inicialmente é a DTD. No entanto, algumas definições ficam um tanto limitadas, como por exemplo, a definição de subtipos. Um estudo para desenvolver um mapeamento padrão de modelos EXPRESS para XML Schema está sendo realizado de forma que se possa tirar vantagem da estrutura hierarquica de XML.

2.2 Modelo de Dados PCO

O comércio eletrônico aumenta a demanda de integração de serviços pelo compartilhamento de catálogos. Os catálogos HTML têm sido explorados nos últimos anos pelo comércio eletrônico. No entanto, a falta de semântica em HTML dificulta a integração de serviços através de *websites*. Recentemente, XML tem se apresentado como uma nova linguagem de modelagem para integração de aplicações [KUR 2000].

XML pode definir especificações semânticas através da DTD ou através do padrão XML Schema. Entretanto, uma questão de interoperabilidade ainda permanece: por exemplo, quando diferentes comerciantes definem independentemente DTD's / Schemas para seus catálogos, as terceiras partes envolvidas não podem entender os relacionamentos semânticos entre eles. Para mediadores de internet, interoperabilidade de esquemas, ou compartilhamento semântico automático, é obrigatório estabelecer a aplicação integrada de catálogos XML através de *websites*.

Uma nova linguagem de modelagem para comércio eletrônico, denominada PCO (*Portable Compound Object*), foi desenvolvida [KUR 2000]. O modelo PCO habilita a interoperabilidade semântica entre esquemas distribuídos pela herança semântica baseada no modelo orientado a objeto. O objetivo principal é garantir a sinonímia do compartilhamento semântico. Esse modelo foi codificado para duas linguagens baseadas em XML: PCO *specification language* (PSL), uma linguagem de esquema para o modelo, e *Portable Composite Language* (PCL).

Para melhor compreender as questões de interoperabilidade semântica, considere que dois comerciantes, loja de computadores e de câmeras, definam independentemente DTD's / Schemas para seus catálogos XML. Ambos utilizam a *tag* `<price>` para representar o preço dos produtos. Nesse caso, poderão as tags ser interpretadas da mesma forma por um terceiro agente?

A abordagem básica para a interoperabilidade utiliza a hierarquia de classe do modelo orientado a objeto. Com base no exemplo acima, se `PC` e `camera` forem projetadas como subclasses de `mercadoria` comum, o significado de cada *tag* `<price>` é compartilhado entre as classes. Conseqüentemente, um agente de compra pode igualmente interpretar os catálogos de preço do PC e da Câmera através da hierarquia de classes. A figura 2.1 apresenta essa hierarquia de classes.

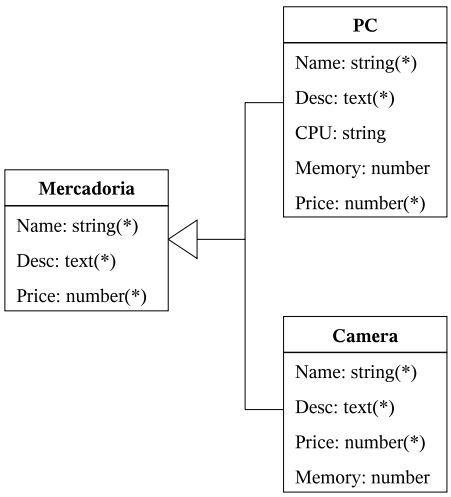


FIGURA 2.1 – Herança semântica

As propriedades `name`, `desc` e `price` estão compartilhando o mesmo significado entre as classes. O mesmo não acontece com a propriedade `memory`, que não está definida na classe `mercadoria` para ser herdada pelas demais classes.

A interoperabilidade de esquema baseada na hierarquia de classes é considera do como um mecanismo simples, uma vez que verifica apenas os homônimos que não têm o mesmo significado.

A linguagem de especificação PCO especifica as classes e tipos de dados que servirão como base para verificar a semântica dos dados. As classes são definidas pela *tag* `ClassDef`, onde serão definidos as super classes, um conjunto pré-definido de dados, `<Chunk>`, e informações de documentação. Um exemplo de definição de classes é apresentado abaixo.

```
<ClassDef name="PersonalProfile">
```

```
    <ChunkDef name="Name" />
    <ChunkDef name="Affiliation" />
    <ChunkDef name="PostalCode" />
    <ChunkDef name="Address" />
    <ChunkDef name="Phone" type="string/phone" />
</ClassDef>

<ClassDef name="NameCard">
    <super class="PersonalProfile" />
    <ChunkDef name="Photo" type="image/jpeg" />
</ClassDef>
```

A principal contribuição desse trabalho é apresentar uma abordagem para interoperabilidade semântica sem uso de ontologias. Baseia-se no modelo orientado a objetos e especifica uma estrutura em XML para definir as classes e dados que serão utilizados para padronizar o uso das informações.

2.3 XPERANTO

Esse é um trabalho desenvolvido por um grupo da IBM, cujo objetivo é servir como uma camada *middleware* que suporta a publicação de dados XML para usuários que preferem lidar diretamente com XML a trabalhar com esquemas de uma fonte específica, como por exemplo objeto-relacional e com uma determinada linguagem de consulta [CAR 2000].

XPERANTO oferece uma interface de consulta uniforme baseada em XML que será aplicada sobre um banco de dados objeto-relacional. Essa interface permite que usuários consultem e reestruturem os conteúdos do banco com dados XML, ignorando as tabelas SQL e a linguagem de consulta.

Na área de comércio eletrônico, existe uma reconhecida necessidade de criar documentos XML através da combinação de uma ou mais tabelas objeto-relacional. A abordagem utilizada é a da criação de uma visão XML virtual do esquema e dados do banco, onde serão realizadas as consultas. A grande vantagem do XPERANTO é que pode ser usado o mesmo *framework* para modelar tanto dados relacionais quanto meta dados relacionais. Não publica somente dado relacionais com XML, mas também estruturas, incluindo tabelas tipadas e co- lunas, OID's e referências, herança e coleções.

Será apresentado nessa seção apenas o mapeamento do esquema do banco de dados objeto-relacional para XML. Para isso, utilizou-se a linguagem XML Schema [W3C 2001a]. Considere a DDL abaixo como exemplo de um esquema objeto-relacional:

```
create table book as (bookID char(30), name varchar(255), publisher varchar(30))
create table publisher as (name varchar(30), address varchar(255))
create table author_type as (bookID char(30), first Vvarchar(30),
                             last varchar(30))
create table author of author_type as (ref is ssn user generated)
```

A estrutura especificada em XML Schema para o esquema anterior é apresentada a seguir:

```
<simpleType name="string255" source="string">
    <maxLength value="255" />
</simpleType>

<simpleType name="string30" source="string">
```

```
   <maxLength value="30" />
</simpleType>

<complexType name="bookTupleType>
   <element name="bookId"    type="string30" />
   <element name="name"      type="string255" />
   <element name="publisher" type="string30" />
</complexType>

<complexType name="bookSetType>
   <element name="bookTuple" type="bookTupleType" maxOccurs="*" />
</complexType>

<element name="book" type="bookSetType" />

<complexType name="author_type>
   <element name="bookId"    type="string30" />
   <element name="first"     type="string30" />
   <element name="last"      type="string30" />
</complexType>

<complexType name="authTupleType>
   <attribute name="ssn" type="ID" />
</complexType>

<complexType name="authSetType">
   <element name="authTuple" type="authTupleType" maxOccurs="*" />
</complexType>

<element name="author" type="authSetType" />
```

A versão da XML Schema utilizada no exemplo acima, não é a mais atual. Algumas modificações foram realizadas na linguagem para definição dos tipos complexos, derivação de tipos e derivação por extensão. A tabela 2.1 apresenta um pequeno sumário de como foi feito o mapeamento do esquema do banco de dados para XML Schema.

TABELA 2.1 – Resumo mapeamento esquema do banco de dados para XML Schema

Esquema BD	XML Schema
Tabelas ou tabelas tipadas	Tipos complexos
Propriedades	Elementos
Tipos de dados	Tipos simples ou derivados
Herança	Derivação por extensão

Em resumo, o principal objetivo de XPERANTO é permitir que desenvolvedores XML publiquem dados objeto-relacional no formato padrão XML, sem que os mesmos tenham que lidar com os esquemas nativos do banco de dados ou com a linguagem de consulta SQL.

2.4 Conclusão

Os trabalhos relacionados nessa seção apresentam exemplos de representação de esquemas em XML. As áreas de aplicação são diferentes, mas o ponto em comum entre eles é a representação de esquemas objeto-relacional. Outros trabalhos que envolvem também a especificação de esquemas em XML estão definidos em [BAR 99, SHI 99, FAN 2000, BOU 2001].

A especificação de esquemas em XML pode ser utilizada para que dados sejam trocados via WEB, permitindo que usuários lidem diretamente com dados XML ao invés de trabalhar com a sintaxe particular de cada banco de dados e também, possibilita a interoperabilidade entre aplicações.

Não existe uma padronização no que diz respeito à representação de esquemas objeto-relacional em XML. Todos esses trabalhos definem uma forma diferente para representar os dados. Nenhum apresenta uma estrutura completa para definir todas as características de aplicações objeto-relacionais. No entanto, o mais próximo da representação completa é a linguagem EXPRESS, que permite definir funções, hierarquia de classes, tipos definidos pelo usuário, entre outros.

A estrutura definida para representar os esquemas e algumas nomenclaturas, foram utilizadas nesse trabalho como uma base para a definição da estrutura que especificará o modelo temporal de versões.

3 Modelo Temporal de Versões

Algumas aplicações necessitam armazenar o histórico dos seus dados e suas diversas representações, que podem ser revisões ou alternativas criadas, denominadas versões. Bancos de dados tradicionais não representam de forma clara e natural esses tipos de dados, fazendo-se necessária a extensão dos mesmos para um modelo que suporte dados versionados e temporais.

O Modelo Temporal de Versões (TVM - *Temporal Versions Model*) [MOR 2001b] é uma extensão do modelo de versões definido em [GOL 95]. Foi acrescentada a esse modelo de versões a dimensão temporal. Dessa forma, é possível armazenar o histórico dos objetos, bem como a evolução de suas propriedades e relacionamentos. O tipo de versionamento utilizado por essa extensão é o de instâncias, isto é, as modificações dos valores das propriedades serão armazenadas dentro das próprias instâncias. Dessa forma, as versões irão se diferenciar somente nos valores de algumas de suas propriedades.

Uma das principais vantagens da união dos conceitos de versão e tempo é a possibilidade de definir um objeto e, para cada objeto definido, armazenar sua evolução e alternativas criadas (versões) e guardar, para cada versão, o histórico das modificações ocorridas nos seus elementos dinâmicos.

A seguir, serão apresentadas as principais características do Modelo Temporal de Versões, definidas em [MOR 2001], que serão utilizadas no decorrer desse trabalho.

3.1 Características do Modelo

O TVM foi baseado no modelo de versões descrito em [GOL 95]. Um dos principais objetivos desse novo modelo é o de permitir a definição de classes que trabalhem com tempo e versão. Para tanto, acrescentou-se ao modelo base a dimensão temporal. Versão representa a descrição de um objeto em um certo instante de tempo ou sob um certo ponto de vista que é considerado relevante para uma determinada aplicação [GOL 95a]. Tempo permite a representação do histórico dos dados. A união desses conceitos, tempo e versão, proporciona uma modelagem mais completa e natural de determinadas aplicações, tais como aplicações de projeto de sistemas de engenharia (*CAD-Computer Aided Design*), de software (*CASE-Computer Aided Software Engineering*), de automação de escritórios e de bancos de dados históricos [GOL 95b].

Esse modelo define classes temporais versionadas que lidam com alternativas de dados e com dados dinâmicos. No entanto, o modelo não trabalha somente com esses tipos de classes. Ele também permite que o usuário defina classes de aplicações que não necessitem armazenar o histórico e a evolução de seus dados: geralmente são classes de uma base já existente ou classes auxiliares. Como resultado, tem-se que, classes definidas como não temporais e não versionadas irão apenas guardar os valores atuais dos dados, enquanto as temporais versionadas armazenam todo o histórico e toda a evolução dos dados. A hierarquia de classes do modelo é apresentada na figura 3.1.

A hierarquia do modelo define quatro classes básicas: *Object*, *Temporal Ob-*

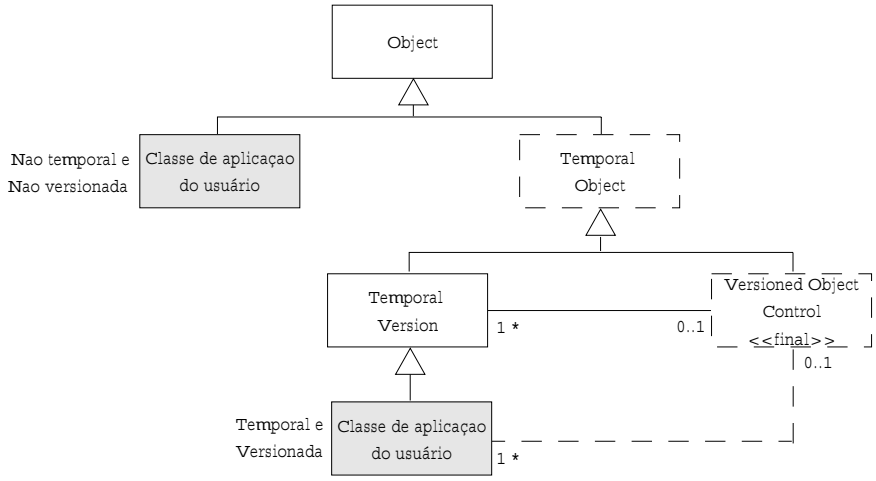


FIGURA 3.1 – Hierarquia de Classes do TVM

ject, *Temporal Version* e *Versioned Object Control*. A classe *Object* contém as propriedades e operações comuns às classes temporais versionadas, e às classes não temporais e não versionadas. Assim como *TemporalObject* e *TemporalVersion*, é uma classe abstrata, não sendo possível sua instanciação direta.

TemporalObject representa os aspectos temporais do modelo. Armazena o estado de vida do objeto, ou seja, informa se o mesmo está ativo ou não. Um objeto é dito ativo quando não foi excluído logicamente da base de dados. *TemporalVersion* armazena os dados relacionados às versões e *VersionedControlObject*, os dados de controle das versões. Essa última classe é uma classe final, não sendo portanto permitida a sua especialização, isto é, a definição de sub-classes. As classes *Temporal Object* e *VersionedObjectControl* são controladas pelo sistema gerenciador, tornando-se transparentes para os usuários.

As classes da aplicação do usuário poderão ser definidas como sub-classes de *Object*, se forem classes auxiliares ou que não necessitem utilizar as características de tempo e versão, ou como sub-classes de *TemporalVersion*, se necessitarem armazenar o histórico e a evolução dos dados. As classes definidas como temporal versionadas irão se relacionar através de herança por extensão e as demais classes, não temporais e não versionadas, irão se relacionar através de herança por extensão ou de herança por refinamento.

A herança por extensão difere da herança por refinamento por permitir que a instanciação ocorra em um determinado nível da hierarquia sem que se conheça previamente os dados a serem inseridos em toda a hierarquia. Nesse caso, cada propriedade se refere a um nível específico da hierarquia, modelando somente aspectos relevantes do objeto [GOL 95a].

Um ponto importante a ser ressaltado é a impossibilidade de definição de classes somente temporais ou somente versionadas: ou elas são temporais versionadas ou são não temporais e não versionadas. Isso se deve ao fato do modelo ter sido definido para representar ambas as características de tempo e versão nos objetos. No entanto, é plenamente aceitável pelo modelo que uma classe definida como sub-classe de *TemporalObject* não tenha versões associadas a ela, ou seja, seja um objeto sem versões. Isso se deve ao fato de que a característica de um objeto

ser ou não versionado é de cada objeto e não da classe.

Uma restrição imposta pelo modelo é a impossibilidade de alterar dinamicamente um objeto definido como temporal versionado para não temporal e não versionado, e vice-versa. O usuário deverá ter consciência disso no momento em que estiver fazendo a definição do esquema da sua aplicação, na fase de projeto.

As propriedades e operações das classes da hierarquia do TVM serão apresentados nas seções seguintes, bem como as características temporais e das versões do modelo.

3.1.1 Características das versões

Versões representam as alternativas de projeto de uma determinada aplicação e a evolução dos seus dados. Cada versão possui um estado que reflete seu estágio de desenvolvimento e/ou consistência [GOL 95b]. Uma versão pode mudar de estado durante o seu tempo de vida por meio de alguns eventos. Os estados que uma versão pode assumir são:

- *Working*: versão temporária que pode sofrer diversas alterações antes de atingir um estado mais estável. É possível realizar consultas, alterações e exclusão;
- *Stable*: versão que já atingiu um estado mais avançado e consistente. Pode ser compartilhada, consultada, mas não alterada. A exclusão só é permitida se não possuir nenhuma versão sucessora;
- *Consolidated*: é uma versão final que não pode ser alterada e nem excluída. Apenas pode ser compartilhada;
- *Deactivated*: é uma versão que teve seu estado de vida finalizado. Apenas pode ser consultada.

A figura 3.2 [MOR 2001] apresenta os estados das versões e os eventos que ocasionam a transição entre os mesmos.

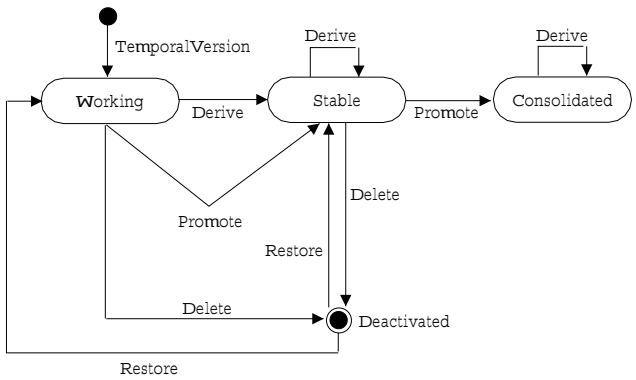


FIGURA 3.2 – Diagrama de estados de uma versão

Uma versão é criada com estado *working*. Quando é derivada de uma ou mais versões, suas antecessoras são automaticamente promovidas para *stable*, impedindo que sejam feitas modificações em versões que serviram de base para a

construção de outras. Versões com estado *working* podem também ser explicitamente promovidas pelo usuário para o estado *stable*, podem ser excluídas, assumindo o estado *deactivated*, e restauradas, voltando para o seu estado *working*.

Versões *stable* podem ser derivadas, excluídas - assumindo dessa forma o estado *deactivated*, restauradas para o estado *stable* e explicitamente promovidas pelo usuário para o estado *consolidated*. As versões cujo estado seja *consolidated* somente podem ser derivadas.

Vale ressaltar que quando se fala que uma versão poderá ser excluída, significa que será realizada uma exclusão lógica, que é o tipo de exclusão implementada pelo modelo. A exclusão física pode causar a perda de dados relevantes, ocasionando uma quebra do histórico dos dados, e portanto, caberá ao DBA decidir quando efetuar tais exclusões.

Quando uma versão é excluída logicamente, assume o estado *deactivated* e tem seu tempo de vida finalizado.

O relacionamento entre as versões de um mesmo objeto pode ser representado através de um gráfico acíclico dirigido, isto significa que cada versão pode ter várias versões sucessoras e várias antecessoras.

Como visto no início dessa seção, o relacionamento utilizado entre classes definidas como temporal versionada é o de herança por extensão, isto é, versões são admitidas em vários níveis da hierarquia de herança.

Através dessa hierarquia, pode-se estabelecer correspondências entre versões de um objeto em uma classe e versões de seu objeto ascendente na super-classe. A correspondência estabelece uma restrição de integridade, que é especificada quando da definição do relacionamento de herança entre uma classe e sua super-classe.

A correspondência definida entre as versões em uma classe e aquelas em sua super-classe está ilustrada na figura 3.3 e pode ser do tipo:

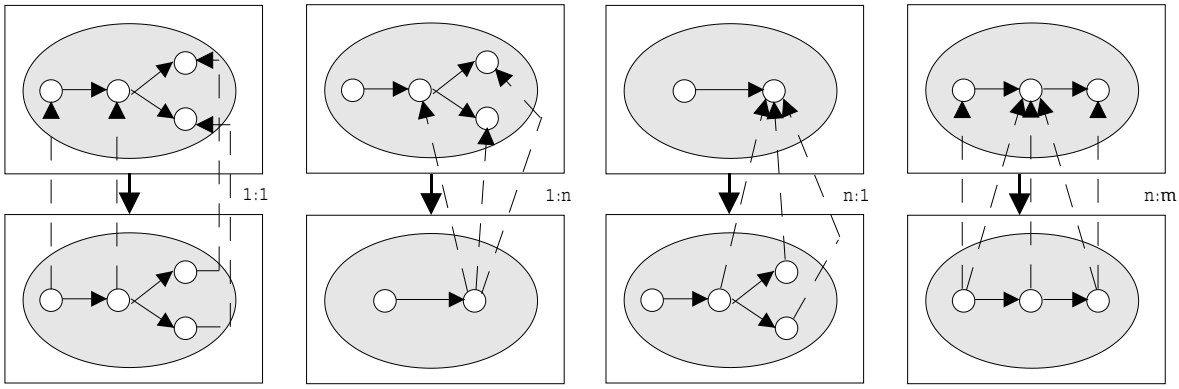


FIGURA 3.3 – Tipos de correspondências entre versões

- 1:1 - cada versão na sub-classe deverá corresponder a exatamente uma versão na super-classe;
- n:1 - várias versões na sub-classe poderão corresponder a uma mesma versão na super-classe, mas cada versão na sub-classe só poderá corresponder a uma versão na super-classe;

- 1:n - várias versões na super-classe poderão corresponder a uma versão na sub-classe e só uma versão na sub-classe poderá corresponder a cada versão na super-classe;
- n:m - várias versões na sub-classe poderão estar relacionadas com uma versão na super-classe e cada versão na sub-classe poderá corresponder várias versões na super-classe.

3.1.2 Características temporais

O Modelo Temporal de Versões é um modelo bitemporal, sendo usados tanto o tempo de transação (quando a informação é inserida no banco de dados) como o de validade (quando a informação é válida na aplicação modelada), com variação discreta e temporalidade representada através do rótulo temporal elemento temporal (conjunto disjunto de intervalos temporais).

O tempo é relacionado a objetos, versões, propriedades e relacionamentos. Objetos têm uma linha de tempo associada a cada versão. Esse tempo pode ser tempo ramificado para objetos, e linear para versões.

Cada objeto tem um estado de vida que está associado à propriedade *alive*, definido na classe *Temporal Object*. Essa propriedade inicialmente recebe o valor *true* (no momento da criação do objeto) e é alterado para *false* no momento da exclusão (lógica). Ao inicializar o estado de vida do objeto é também incluído o seu tempo de validade inicial.

As propriedades e relacionamentos dos objetos podem ser definidos pelo usuário como estáticos, quando possuírem um único valor a ser armazenado, ou temporalizados, quando se desejar manter o histórico dos valores assumidos pelas propriedades ou relacionamentos. Uma classe temporal versionada pode ter ambos os tipos de propriedades e relacionamentos.

Propriedades e relacionamentos definidos como temporalizados têm seus valores relacionados a rótulos de tempos de validade inicial e final, e tempos de transação inicial e final pré-definidos. Para tanto, algumas regras devem ser observadas:

- tempo de validade inicial deve ser maior ou igual ao tempo de vida inicial da versão a qual a propriedade ou relacionamento pertence;
- tempo de transação inicial deve ser maior ou igual ao tempo de vida inicial da versão a qual a propriedade ou relacionamento pertence;
- 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 uma versão deve ser maior ou igual ao maior tempo de transação e validade final de suas propriedades ou relacionamentos;
- tempo de vida final de um objeto versionado deve ser maior que o tempo de validade inicial e maior ou igual que o tempo de validade de suas versões.

Para trabalhar com os rótulos temporais, foram definidas classes que modelam esses rótulos para propriedades e relacionamentos. A figura 3.4 apresenta essas classes com suas respectivas propriedades.

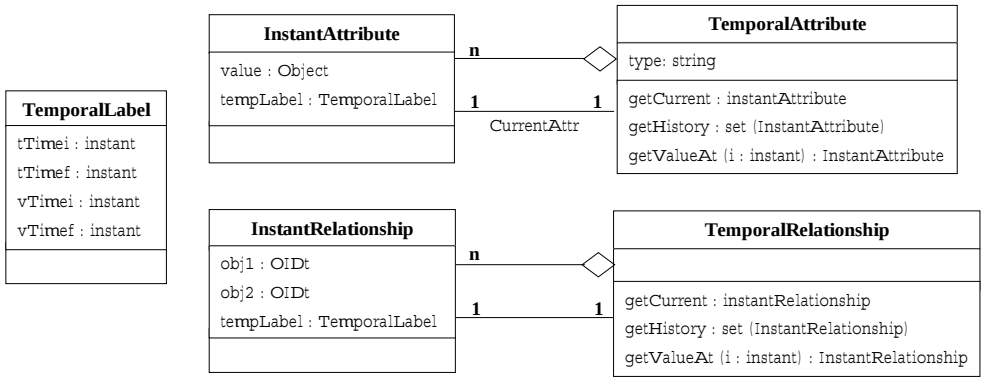


FIGURA 3.4 – Classes que modelam os rótulos temporais para propriedades e relacionamentos

A classe *TemporalLabel* é responsável pelo armazenamento dos rótulos temporais tempo de validade inicial e final e tempo de transação inicial e final. A figura 3.5 mostra os métodos implementados por essa classe.

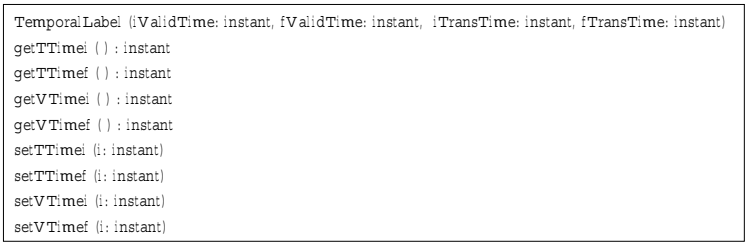


FIGURA 3.5 – Métodos da classe *TemporalLabel*

A classe *InstantAttribute* é responsável pelo armazenamento do valor da propriedade com seu respectivo rótulo temporal, enquanto a classe *InstantRelationship* é responsável pelo armazenamento dos identificadores dos objetos ou versões que fazem parte de um relacionamento e seus rótulos temporais. A figura 3.6 mostra os métodos implementados respectivamente por essas classes.

A classe *TemporalAttribute* é um agregado dos valores assumidos pelas propriedades dinâmicas dos objetos durante o seu tempo de vida. Possui um relacionamento com a classe *InstantAttribute* indicando o valor corrente da propriedade. Da mesma forma, a classe *TemporalRelationship* é um agregado dos valores assumidos pelos relacionamentos dos objetos durante seu tempo de vida. Possui também um relacionamento *InstantRelationship* que indica o relacionamento corrente. A figura 3.7 mostra os métodos implementados respectivamente por essas classes.

3.1.3 Exemplo

Para facilitar a visualização da aplicação do TVM, considere o esquema abaixo especificado. Esse esquema representa configurações de computadores e é composto por três classes definidas como temporal versionadas: `computador`,

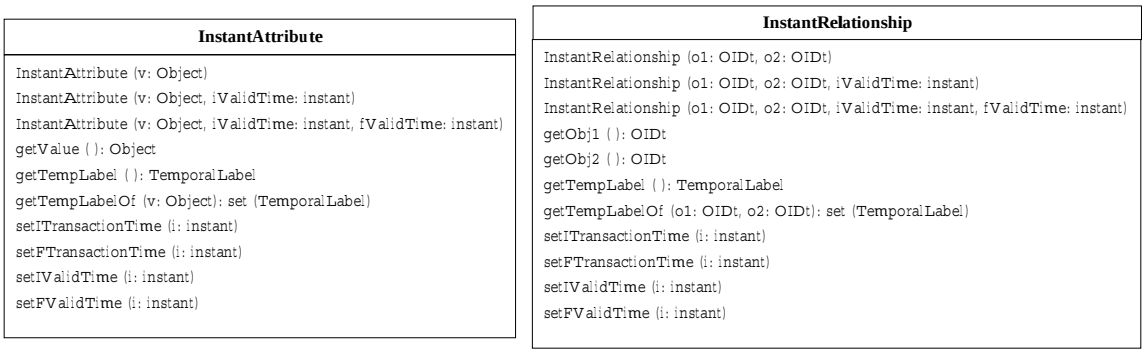


FIGURA 3.6 – Métodos da classe *InstantAttribute* e *InstantRelationship*

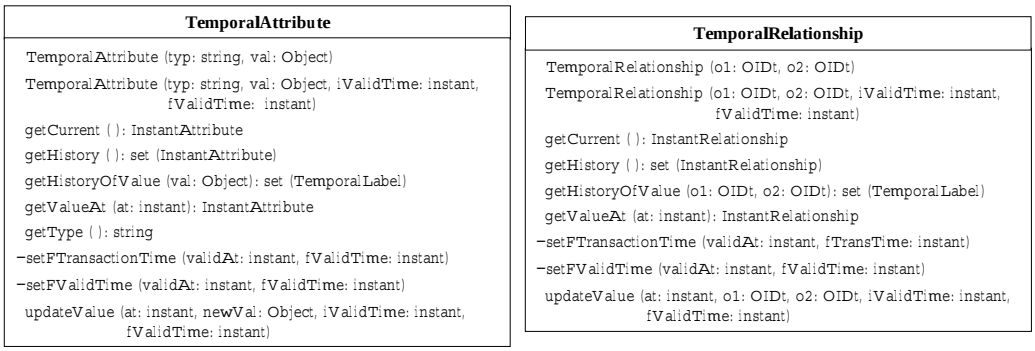
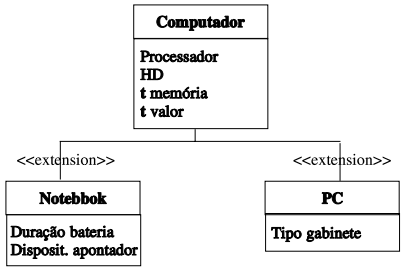


FIGURA 3.7 – Métodos da classe *TemporalAttribute* e *TemporalRelationship*

`notebook` e `PC`. As classes estão relacionadas entre si através da herança por extensão e as versões estão modeladas em mais de um nível de abstração, como ilustra a figura 3.8.



A classe `computador` possui as propriedades `processador`, `HD`, `memoria` e `valor`, sendo as duas últimas, propriedades temporais. A classe `notebook` possui as propriedades `duração bateria` e `dispositivo apontador`. Ambas as classes possuem versões associadas a seus objetos versionados e cada versão tem pelo menos uma versão ascendente correspondente. Como resultado dessa correspondência entre versões, é possível que uma mesma característica definida em um nível de abstração possa ser ligada a diferentes características no nível superior da hierarquia de herança.

Considerando o exemplo da figura 3.8, as mesmas características definidas para a versão `N` de `notebook` podem ser ligadas a diferentes versões de `C` em

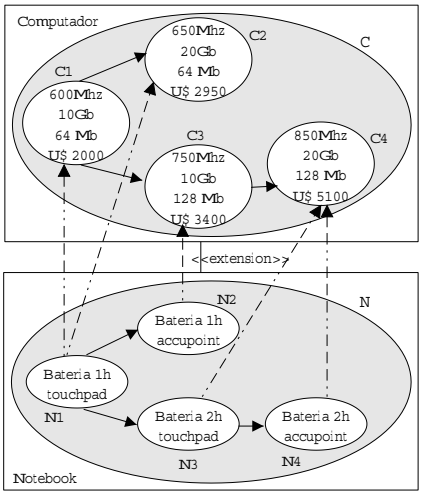


FIGURA 3.8 – Aplicação exemplo

`computador`, nível superior da hierarquia de herança. Da mesma forma, uma versão em uma super-classe pode corresponder a mais de uma verão na sub-classe, como é o caso da versão `c4` de `computador` que corresponde às versões `n3` e `n4` de `notebook`.

Assim sendo, é possível definir as características básicas de configurações de computadores em um nível de abstração e posteriormente detalhá-las nos níveis de abstração inferiores da hierarquia de herança. Essa relação de versões entre níveis de hierarquia é o que se denomina de correspondência e resulta em uma restrição de integridade. No exemplo acima, a correspondência utilizada é a de `n:m`.

Lembrando que o TVM é um modelo bitemporal, todos as propriedades temporais têm associadas a si os rótulos de tempo de transação e de validade inicial e final. Assim sendo, as propriedades `memoria` e `valor` de `computador` devem armazenar esses rótulos temporais durante o processo de inserção, alteração e exclusão lógica. Algumas considerações devem ser feitas durante esses processos:

- no momento da inserção, o usuário informa o valor da propriedade e os tempos de validade inicial (obrigatório) e final (opcional). Caso o valor de validade final não seja informado, será armazenado `null`;

- sendo a validade final da propriedade `null`, o valor é considerado válido até que um novo valor seja definido ou que o objeto seja logicamente excluído;

- os tempos de transação inicial e final são fornecidos pelo SGBD. Em uma inserção, somente o tempo de transação inicial é fornecido. O tempo de transação final é armazenado quando uma nova instância para o objeto é inserida, atualizada ou excluída logicamente da base de dados;

- como o histórico da propriedade é armazenado (considerando as propriedades temporais), a alteração de um valor resulta em uma nova inserção e

somente é possível para aquelas propriedades cujo tempo de transação final está em aberto [ELM 2000].

Para visualizar melhor os processos de inserção, atualização e exclusão das propriedades temporais, considere que as operações descritas abaixo foram realizadas sobre a propriedade `valor` da versão `c4` de `computador`.

- inserção do valor inicial da propriedade U$4500 em 05/01/2001 com validade inicial em 10/01/2001;
- alteração do valor em 02/03/2001 para U$4850;
- alteração do valor em 20/07/2001 para U$5100;
- exclusão do valor em 30/10/2001.

A tabela 3.1 apresenta o histórico dos valores da propriedade `valor` armazenado fisicamente.

TABELA 3.1 – Exemplo de operações realizadas sobre propriedades temporais

	Valor	TVI	TVF	TTI	TTF
1	U$4500	10/01/2001	null	05/01/2001	**02/03/2001**
2	U$4500	10/01/2001	01/03/2001	02/03/2001	null
3	U$4850	02/03/2001	null	02/03/2001	**20/07/2001**
4	U$4850	02/03/2001	19/07/2001	20/07/2001	null
5	U$5100	20/07/2001	null	20/07/2001	**30/10/2001**
6	U$5100	20/07/2001	29/10/2001	30/10/2001	30/10/2001

No momento da inserção, a primeira linha da tabela é gerada com o tempo de validade inicial (TVI) informado pelo usuário. Como o tempo de validade final (TVF) não foi definido, é armazenado o valor `null` indicando que é válido até o infinito futuro. O tempo de transação inicial (TTI) equivale ao dia em que o valor foi inserido no banco de dados, e o tempo de transação final (TTF) é `null`.

Ao alterar o valor da propriedade para `U$4850`, uma cópia do valor corrente é feita para que o mesmo possa ter sua validade finalizada (linha 2). A validade final é a validade inicial do novo valor menos 1. Nesse momento, o tempo de transação final do valor também é encerrado (linha 1). A cópia do valor antigo é feita para que não se perca a informação da validade da propriedade. O novo valor então é inserido (linha 3). O mesmo processo é aplicado na alteração do valor de `U$4850` para `U$5100` (linhas 4 e 5).

No momento da exclusão, uma nova tupla é inserida encerrando tanto o tempo de validade quanto o de transação (linha 6). O tempo de transação do valor até então válido também é encerrado (linha 5). Essa é uma das formas de implementar a exclusão, permitindo que somente os valores correntes estejam com o tempo de transação final em aberto.

3.2 Propriedades e Operações das Classes

As classes definidas no TVM têm por finalidade tratar e manipular dados com tempo e versão. Baseado no paradigma de orientação a objetos, o modelo

especifica métodos para cada classe que auxiliam a execução de funções e procedimentos de controle e integridade dos dados.

Alguns desses métodos envolvem acesso a metadados, que são responsáveis por guardar informações sobre todo o esquema do usuário. Como exemplo de dados armazenados pelos metadados pode-se citar: quais classes são definidas como temporais versionadas, que tipo de relacionamento existe entre duas classes (herança por extensão, herança por refinamento, associação, etc.), quais propriedades e relacionamentos são definidos como temporais, entre outros. Vale ressaltar que na especificação do modelo não foram definidas tais classes. Traba- lhou-se com a hipótese de que esses metadados já estavam definidos no banco de dados.

As propriedades e operações de cada uma das classes do TVM serão apresentados a seguir. A notação utilizada para a definição dos métodos é a seguinte:

- -: indica que a operação é do tipo *private*;
- #: indica que a operação é do tipo *protected*;
- negrito: indica que a operação é do tipo *final*.

Como visto na seção 3.1, a classe *Object* tem as propriedades e operações comuns a todos os objetos: é a super-classe da hierarquia do TVM. Uma de suas principais funções é a definição do identificador dos objetos, definido no modelo como *tvOID*. O *tvOID* mantém a mesma estrutura que foi especificada no modelo base definido em [GOL 95], ou seja, o identificador é formado pelo conjunto: identificador da entidade, identificador da classe e número da versão.

A classe *Object* possui um relacionamento com a classe *Name*, cuja função é armazenar apelidos para os objetos.

As propriedades e métodos da classe *Object*, bem como a classe *Name* e *OIDt* são apresentadas na figura 3.9.

A classe *Temporal Object* representa os aspectos temporais do modelo através da propriedade *alive*, que armazena o estado de vida do objeto, isto é, se o mesmo está ativo ou não. Um objeto é dito ativo quando não tiver sido excluído. A classe *VersionedObjectControl* armazena e manipula os dados de controle das versões e dos objetos versionados. Possui um relacionamento com a classe *TemporalVersion*. As propriedades e métodos de ambas as classes são apresentados na figura 3.10.

A última classe definida no modelo é a *TemporalVersion* que manipula objetos temporais versionados. Suas propriedades e métodos são apresentados na figura 3.11.

3.3 Linguagem de Definição de Dados Estendida

Para a definição de classes do esquema do usuário, fez-se necessária a extensão da linguagem de definição de dados inicialmente proposta em [GOL 95] e posteriormente atualizada em [SAG 99]. Dessa forma, o usuário poderá definir classes da aplicação que são temporais e versionadas e classes não temporais e não versionadas. A figura 3.12 apresenta a extensão da linguagem de definição de dados de forma simplificada.

A cláusula *hasVersions* indica que uma classe é temporal versionada, sendo obrigatória para a definição dessas classes. A cláusula *Temporal* só é utilizada

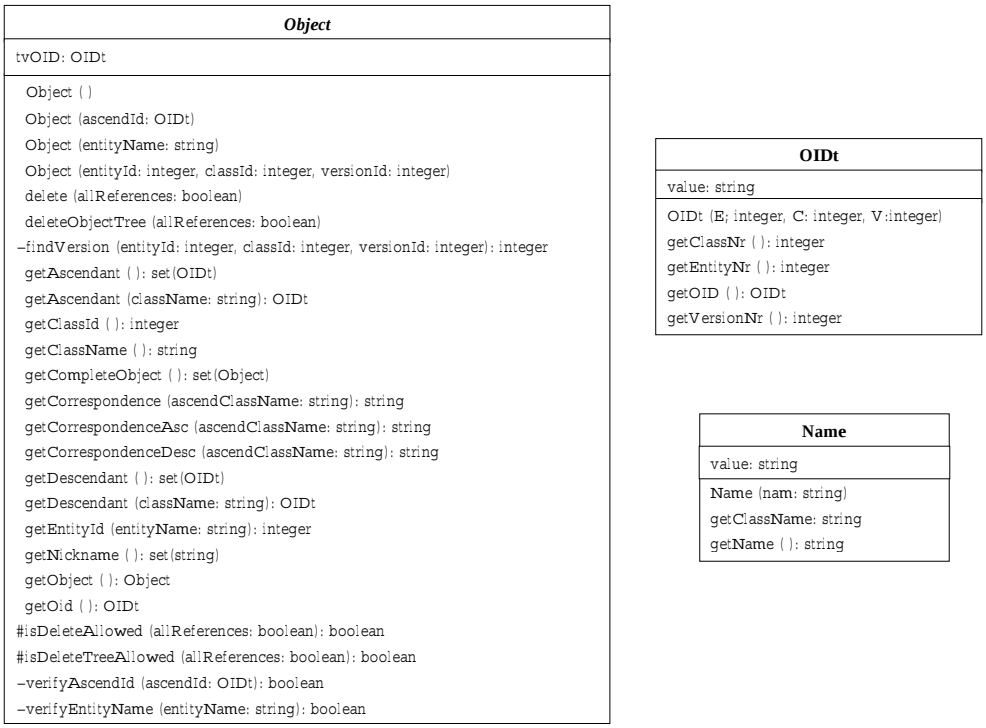


FIGURA 3.9 – Especificação das classes *Object, OIDt* e *Name*

em classes temporais versionadas e indica quais propriedades, relacionamentos e agregações são temporalizados. A cláusula *correspondence* é utilizada para estabelecer a correspondência entre a classe descendente e sua ascendente em uma hierarquia de herança por extensão e somente é permitida na definição de classes temporais versionadas. Quando a correspondência se estabelecer entre uma classe temporal versionada e uma não temporal e não versionada, as únicas correspondências permitidas são `1:n` ou `n:1`, devido ao fato de uma classe não temporal e não versionada possuir apenas uma instância para cada objeto. A cláusula *byExtension* é utilizada para indicar herança por extensão entre classes não temporais e não versionadas e é opcional.

A visibilidade das classes, propriedades e operações também é estabelecida na extensão da DDL. As classes, propriedades e operações podem ter visibilidade *public* indicando que podem ser acessados de qualquer outra classe da aplicação; *protected* indicando que somente podem ser acessadas por classes e suas subclasses e *private* indicando que somente poderão ser acessadas pela própria classe. Quanto ao tipo, podem ser *abstract* indicando que não podem ser instanciadas e *final* indicando que não poderão ser especializadas. *Static* indica que só poderão acessar dados definidos como tal.

3.4 Conclusão

A modelagem dos conceitos de tempo e versão permite que aplicações mais complexas, geralmente aplicações que lidam com alternativas de projeto e para as quais a evolução dos dados no tempo é importante, possam ser representadas

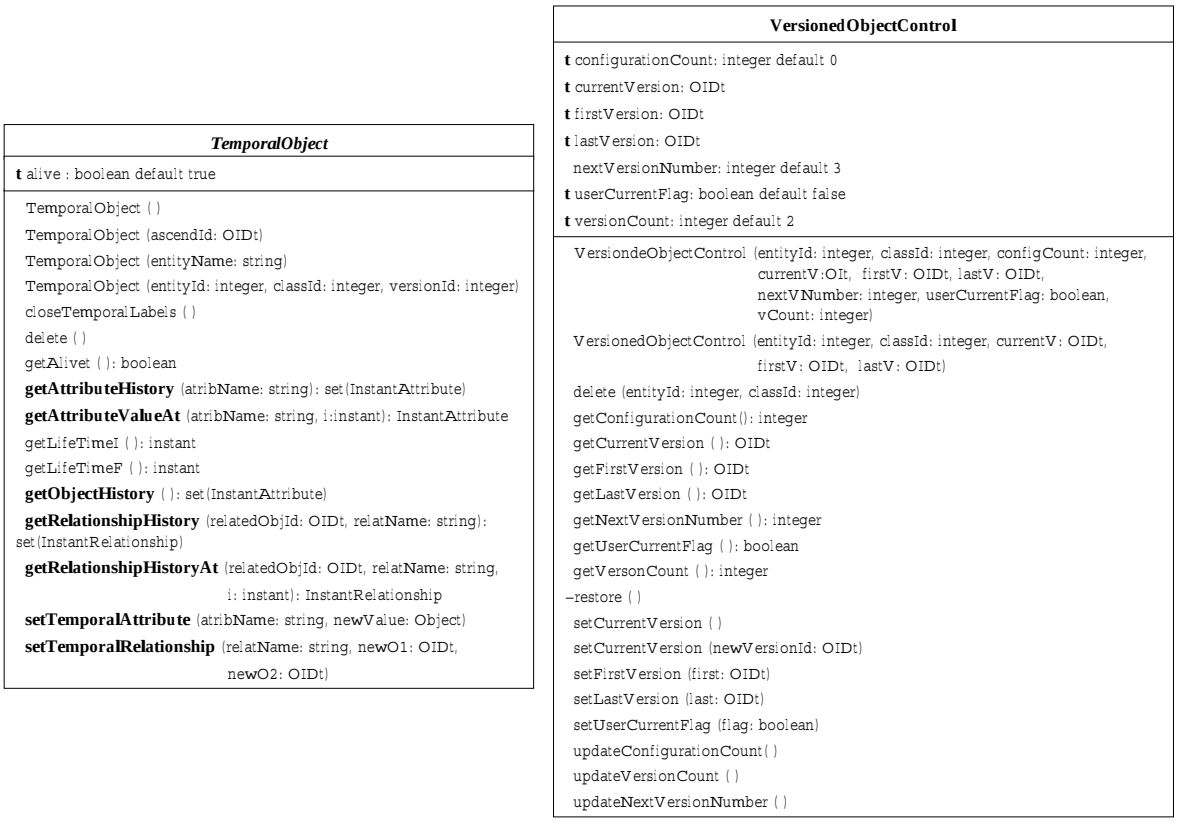


FIGURA 3.10 – Especificação das classes *TemporalObject* e *VersionedObjectControl*

de uma forma mais natural.

O Modelo Temporal de Versões faz a associação desses dois conceitos. No entanto, não trabalha apenas com aplicações que necessitem utilizar tempo e versão, o que o torna bastante vantajoso. Além de permitir armazenar versões de um objeto e para cada versão, seu tempo de vida e o histórico das modificações ocorridas nos elementos dinâmicos, permite também que usuários definam classes normais, cujos dados armazenados representam apenas o seu último estado, isto é, o seu valor atual.

Ao se definir classes temporais versionadas, deve-se ter em mente que a quantidade de versões criadas pode sobrecarregar a capacidade de armazenamento e diminuir o desempenho da base de dados. O usuário deve escolher para quais elementos deseja que o histórico seja armazenado.

Sendo o TVM um modelo bitemporal, o suporte ao tempo de transação traz consigo o potencial para acessar qualquer estado passado do banco.

Algumas restrições porém são impostas pelo modelo, como por exemplo a impossibilidade de se definir classes somente temporais ou somente versionadas. Também não é possível alterar dinamicamente uma classe definida como temporal versionada para não temporal e não versionada. Apesar disso, o modelo atende aos propósitos inicialmente estabelecidos.

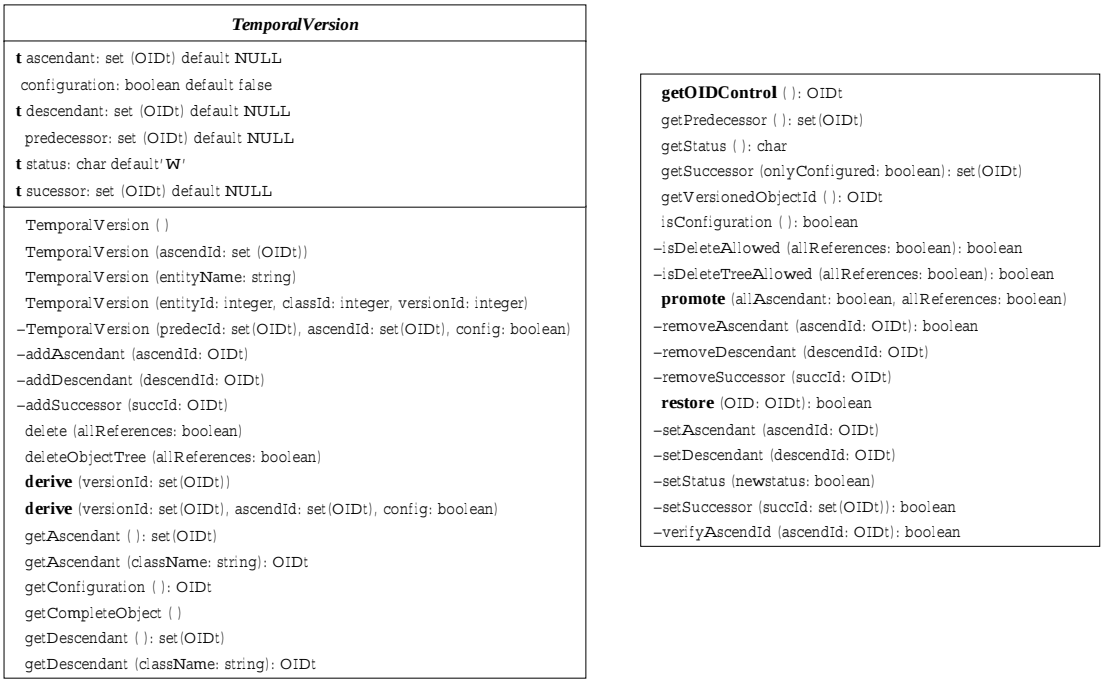


FIGURA 3.11 – Especificação da classe *TemporalVersion*

```
ClassDef ::= [public] [abstract | final]
   class className [hasversions] [inherit [byExtension] className [correspondence {(1:1 |1:n | n:1 | n:m)}]
   [temporal] aggregateOf [n] className (byValue | byReference)
   {, [temporal]  aggregateOf [n] className (byValue | byReference)}]

   ([Properties:
        {[public | private | protected] [static]
          [temporal] attributeName : attributeDomain;}+]
    [Relationships:
        {[temporal] relationshipName (0:1 | 0:n | 1:1 | 1:n | n:m)
          [inverse inverseRelationshipName] relatedClassName ;}+]
    [Operations:
        {[public | private | protected] [static]
          [abstract | final] operationDefinitions;}+]);
```

FIGURA 3.12 – Linguagem de definição de dados estendida

4 Linguagem de Consulta do TVM

Como visto no capítulo 3, o Modelo Temporal de Versões une os conceitos de tempo e versão, possibilitando uma modelagem mais natural de aplicações que necessitam armazenar as alternativas ou revisões criadas para um determinado objeto (versões), bem como o histórico desses objetos e a evolução de suas propriedades e relacionamentos.

Aplicações modeladas através do TVM necessitam recuperar seus dados por meio de uma linguagem de consulta. No entanto, a atual linguagem padrão de consulta, a SQL (*Structured Query Language*) [DAT 97], não lida com dados temporais versionados, sendo necessária a definição de uma linguagem específica para o modelo. Essa linguagem, a Linguagem de Consulta do Modelo Temporal de Versões (TVQL - *Temporal Version Query Language*) foi especificada em [MOR 2001a].

Baseada na linguagem padrão SQL, a TVQL além de permitir realizar consultas básicas efetuadas em SQL, acrescenta o suporte para recuperar dados específicos de tempo e versão. Isso se deve ao fato de o TVM possibilitar que classes da aplicação do usuário sejam definidas como não temporais e não versionadas e como temporais versionadas. Dessa forma, uma única linguagem é necessária para recuperar ambos os tipos de dados.

A TVQL define funções e propriedades específicas para a recuperação de valores relativos a tempo e versão. Sua estrutura base é a mesma da SQL: SELECT - FROM - WHERE. Permite definir *alias* tanto para as coleções de dados quanto para as propriedades. Elimina elementos duplicados através da cláusula DISTINCT e utiliza operadores relacionais, lógicos, de conjunto, de concatenação, de condições compostas e de combinação de padrões. Também é possível a utilização de funções de agregação e de agrupamento.

Como o TVM é fundamentado no paradigma de orientação à objetos, as coleções de dados especificadas na cláusula FROM são as classes definidas para a aplicação ou versões definidas para um objeto de uma classe da aplicação.

Nos tópicos seguintes serão apresentadas as características específicas de tempo e versão definidas para a TVQL, definidas em [MOR 2001a].

4.1 Características temporais

Ao se trabalhar com dados temporais, há a necessidade de se recuperar os valores que uma determinada propriedade assume durante o seu tempo de vida, ou seja, recuperar todo o histórico dessa propriedade. É importante também a possibilidade de recuperar o valor da propriedade em um determinado instante ou período de tempo, bem como o seu valor atual. Todos esses resultados poderão vir acompanhados dos seus respectivos rótulos temporais.

No caso do TVM, pode-se definir como temporais propriedades e relacionamentos. Isso significa que poderão ser realizadas consultas para recuperar o histórico de uma propriedade ou relacionamento, o seu valor em um instante específico ou seu valor atual, com seus respectivos rótulos temporais. Sendo o modelo bitemporal, os rótulos temporais retornados poderão ser os tempos de

validade inicial e final, e/ou os de transação inicial e final. A propriedade *alive*, definida na classe *TemporalObject*, informa o tempo de vida de cada objeto e é aplicada tanto a objetos quanto a versões, podendo seus valores serem retornados pelas consultas.

Antes de ser explicada as características temporais da TVQL, um pequeno exemplo será apresentado de forma a melhorar o entendimento da linguagem. Considere que se deseja recuperar o histórico dos salários de todos os funcionários alocados (hoje) no departamento de informática da empresa. A consulta é definida da seguinte forma:

```
SELECT EVER salario, viInstant.salario, vfInstant.salario
FROM empregado e, departamento d
WHERE PRESENT (departamento='Informatica') and e.dept=d.dept
```

Sendo a consulta ao histórico dos dados uma das principais operações realizadas em banco de dados temporais, definiu-se a cláusula `EVER` na TVQL que possibilita essa operação. Sua utilização pode se dar de duas formas:

- após `SELECT`: retorna o histórico das propriedades temporais considerando o histórico dos valores especificados na cláusula WHERE;
- após `WHERE` : considera o histórico dos valores especificados nessa cláusula.

Em alguns casos, é necessário combinar dados atuais com o histórico de determinadas propriedades temporais em uma mesma consulta. Como o histórico dos dados será consultado, a cláusula `EVER` deverá ser utilizada. No entanto, poderá não ser aplicada a toda a consulta. Nesses casos, a cláusula `PRESENT` deverá ser aplicada para explicitar em quais dados serão considerados somente os valores correntes.

Para recuperar unicamente valores em um determinado instante ou valores correntes (atuais), foram definidas funções e propriedades específicas para esse fim, e nesses casos, a cláusula `EVER` deverá ser omitida.

Os rótulos temporais das propriedades e relacionamentos são recuperados através das propriedades `tiInstant`, `tfInstant`, `viInstant` e `vfInstant`. Tais propriedades retornam o tempo de transação inicial, tempo de transação final, tempo de validade inicial e tempo de validade final, respectivamente. É possível também recuperar os intervalos de tempo de transação e tempo de validade através das propriedades `tInterval` e `vinterval`. Todas essas propriedades podem ser utilizadas tanto na cláusula SELECT quanto na WHERE.

As propriedades `Instant`, definidas anteriormente, podem ser combinadas com operadores relacionais ($-$, $>$, $<$, $>=$, $<=$, $<>$) formando condições temporais. Essas condições são especificadas pelo usuário na cláusula WHERE. A tabela 4.1 apresenta algumas condições que podem ser criadas pelo usuário.

Para exemplificar o uso dessas condições temporais na prática, três condições são mostradas a seguir. É importante observar que para definir uma condição temporal, a propriedade à qual será aplicada a condição deve anteceder a função temporal. No caso das condições a seguir, a primeira compara o instante inicial de transação de uma determinada propriedade com o instante atual, *now*; a segunda, verifica se o instante de validade inicial de uma propriedade é maior ou igual ao instante `10/01/2001` especificado; e a última, verifica se o instante de validade final de uma propriedade é menor que o instante armazenado em `dataContratacao`:

TABELA 4.1 – Condições temporais para instantes

Condição temporal			Definição
tiInstant tfInstant viInstant vfInstant propriedade	= > < >= <= <>	*now*	Compara o instante com o instante atual
		`valorTempo`	Compara o instante com o instante apresentado por `valorTempo`
		`propriedade`	Compara o instante com o valor de `propriedade`, sendo esta definida como TIMESTAMP

```
propriedade.tiInstant  = Now
propriedade.viInstant >= "10/01/2001"
propriedade.vfInstant  < dataContratacao
```

As propriedades `tInterval` também podem ser combinadas formando condições temporais. Outros operadores, diferentes dos utilizados pelas propriedades `tInstant`, são empregados para realizar tais comparações: `intersect`, `overlap` e `equal`. Esses operadores verificam se um intervalo intersecciona qualquer ponto de um outro intervalo, intersecciona todo o intervalo ou se os intervalos são iguais, respectivamente. A figura 4.1 apresenta o escopo desses operadores.

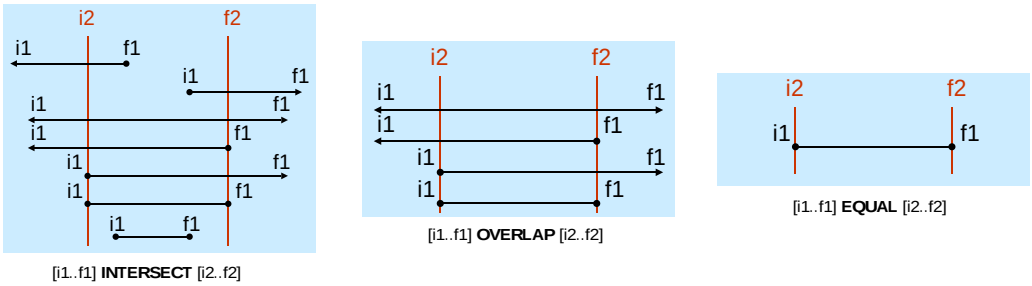


FIGURA 4.1 – Operadores de comparação para intervalos

As condições temporais para intervalos, assim como as condições para instantes, são definidas pelo usuário na cláusula WHERE. A tabela 4.2 apresenta alguns tipos de condições possíveis de serem especificadas.

TABELA 4.2 – Condições temporais para intervalos

Condição temporal			Definição

tInterval
vInterval
	`intersect`
	`overlap`
	`equal`
		`[valor1..valor2]`	Compara o intervalo de transação ou validade com o intervalo definido entre os instantes `valor1` e `valor2`

		`[..valor]`	Compara o intervalo de transação ou validade com o intervalo definido entre o infinito passado e `valor`
		`[valor..]`	Compara o intervalo de transação ou validade com o intervalo definido entre `valor` e o infinito futuro
		tInterval vInterval	Compara os intervalos pré-definidos entre eles

Novamente, para especificar essas condições temporais, a propriedade à qual será aplicada deverá anteceder a função temporal. Nos exemplos a seguir

definidos, a primeira condição verifica se o intervalo de transação de uma determinada propriedade intersecciona qualquer ponto do intervalo definido entre os instantes `valor1` e `valor2`; a segunda, verifica se o intervalo definido em `dataContratacao` é igual ao intervalo definido no infinito passado até o instante `valor`; a condição seguinte verifica se o intervalo de transação de uma propriedade sobrepõe o intervalo definido a partir do instante `valor` até o infinito futuro; também é possível comparar os intervalos de transação e validade entre si, como é visto na última condição. Nesse último caso, o intervalo de transação de uma propriedade é comparado com o intervalo de validade da mesma propriedade. No entanto, é possível a utilização de propriedades diferentes para definir tal comparação:

```
propriedade.tInterval INTERSECT [valor1..valor2]
dataContratacao       EQUAL     [..valor]
propriedade.tInterval OVERLAP   [valor..]
propriedade.tInterval EQUAL     propriedade.vInterval
```

Outra forma de realizar comparações é através dos operadores `before`, `into` e `after`. Esses operadores são utilizados por ambas as propriedades `Interval` e `Instant`. Como resultado da aplicação dessas comparações obtém-se se um determinado intervalo / instante antecede, está contido ou sucede completamente outro intervalo / instante. A figura 4.2 apresenta a abrangência desses operadores.

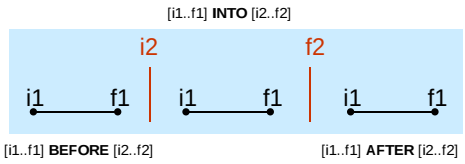


FIGURA 4.2 – Operadores de comparação para instantes e intervalos

As condições temporais aplicadas tanto a instantes quanto a intervalos deverão ser definidas na cláusula WHERE da consulta. A tabela 4.3 apresenta algumas condições que poderão ser especificadas pelo usuário.

TABELA 4.3 – Condições temporais para instantes e intervalos

Condição temporal			**Definição**
tiInstant tfInstant viInstant vfInstant propriedade tInterval vInterval	before into after	[valor1..valor2]	Verifica se o instante, intervalo ou propriedade está antes, dentro ou após o intervalo definido entre os instantes `valor1` e `valor2`
		[..valor]	Verifica se o instante, intervalo ou propriedade está antes, dentro ou após o intervalo definido entre o infinito passado e `valor`
		[valor..]	Verifica se o instante, intervalo ou propriedade está antes, dentro ou após o intervalo definido entre `valor` e o infinito futuro
		tInterval	Verifica se o instante, intervalo ou propriedade está antes, dentro ou após o intervalo de transação
		vInterval	Verifica se o instante, intervalo ou propriedade está antes, dentro ou após o intervalo de validade

Nos exemplos de condições temporais a seguir definidos, verifica-se na primeira condição, se o intervalo de validade de uma determinada propriedade

está definido antes do intervalo entre `valor1` e `valor2`; na segunda condição se o instante definido em `dataContratacao` está contido no intervalo entre infinito passado e `valor`; e na última, se o instante de transação inicial de uma propriedade foi definido após o intervalo entre `valor` e infinito futuro.

```
propriedade.vInterval BEFORE [valor1..valor2]
dataContratacao       INTO   [..valor]
propriedade.tiInstant AFTER  [valor..]
```

Como os objetos e versões têm associados a si tempos de vida inicial e final, foram definidas funções que retornam essas informações: `iLifeTime` e `fLifeTime`. Ambas as funções podem ser usadas tanto na cláusula SELECT quanto na WHERE. As duas cláusulas abaixo, exemplificam o uso dessas funções:

```
x.iLifeTime                    y.fLifeTime
```

4.2 Características de versões

A TVQL permite consultar dados referentes aos objetos e suas versões. Para isso, é necessário que o usuário especifique na cláusula FROM quais objetos e versões deseja consultar. Os objetos são definidos através dos nomes das classes às quais pertencem e as versões, através da função `versions`.

Outras funções foram definidas de forma a facilitar o acesso às informações sobre as versões: funções que recuperam o estado das versões, funções de navegação na hierarquia de herança e de navegação na hierarquia de derivação.

Os estados de uma versão são verificados através das funções `isWorking`, `isStable`, `isConsolidated` e `isDeactivated`. Como se trata de um modelo temporal, é possível também recuperar os estados das versões em um determinado instante. As funções que executam essas operações são as mesmas anteriores acrescidas do sufixo `At`. Essas funções porém, requerem um parâmetro: o ins- tante para a consulta. A seguir, serão apresentados alguns exemplos de uso das funções.

Considere `x` e `y` as versões dos objetos `o1` e `o2` respectivamente. As seguintes funções poderiam ser definidas para recuperar do objeto `o1` as versões que estão em trabalho e as que foram desativadas em `10/01/2001`, e do objeto `o2`, as versões estáveis e as que foram promovidas para o estado consolidado em `01/10/2001`:

```
x.isWorking                      y.isStable
x.isDeactivatedAt ("10/01/2001")  y.isConsolidatedAt ("01/10/2001")
```

As funções de navegação na hierarquia de herança são utilizadas para pesquisar o conjunto de versões que pode estar associado a uma determinada versão. Essas funções, `isAscendantOf` e `isDescendantOf`, recuperam as versões ascendentes e descendentes de uma outra versão respectivamente. O parâmetro necessário para ambas as funções é a versão que identifica o objeto ascendente ou descendente.

Com base nas versões e objetos definidos anteriormente, pode-se especificar por exemplo, uma função para recuperar as versões do objeto `o1` que são descendentes da versão do objeto `o2` e recuperar as versões de `o1` que são ascendentes da versão de `o2`. Esses exemplos estão ilustrados a seguir:

```
x.isDescendantOf (y)          x.isAscendantOf (y)
```

Para navegar na hierarquia de derivação foram definidas as seguintes funções: `isFirst`, `isLast`, `isSuccessorOf`, `isPredecessorOf`, `isCurrent`, `isUserCurrent` e `isConfiguration`. Essas funções verificam respectivamente se uma versão é a primeira ou última no grafo de derivação, se é sucessora ou antecessora de uma determinada versão, se é a versão corrente e se o usuário especificou a versão corrente. Somente as funções `isSuccessorOf` e `isPredecessorOf` possuem parâmetro, que é usado para identificar a versão para comparação.

Como no TVM os dados `first`, `last`, `successor`, `current` e `userCurrent` estão definidos como temporais, existe a necessidade de recuperar esses dados em um determinado instante de tempo. Para isso, acrescentou-se às funções bases o sufixo `At`. O parâmetro comum recebido por todas essas funções é o instante de tempo que será utilizado na consulta.

Considerando que as versões `x` e `y` pertencem ao mesmo objeto, pode-se definir funções para verificar se a versão `x` é a primeira versão, se é a última no instante `01/10/2001`, se é a antecessora de `y` ou se é a sucessora de `y` no instante `01/10/2001`. Essas funções são apresentadas a seguir:

```
x.isFirst                     x.isPredecessorOf (y)
x.isLast ("01/10/2001")       x.isSuccessorOfAt (y, "01/10/2001")
```

No TVM, todos os objetos / versões recebem um identificador único, o `tvOID`. No entanto, é possível especificar na aplicação apelidos para os objetos / versões que poderão ser utilizados nas consultas. A grande vantagem que isso adiciona a consultas é a possibilidade de tratar os objetos / versões pelo nome atribuído a eles. A função `nickname` implementa essa característica. O exemplo a seguir, verifica entre as versões do objeto `o1` a que possui o apelido `versao1`:

```
x.nickname = "versao1"
```

Um ponto importante a ser ressaltado é que todas as funções que lidam com tempo verificam, antes de mais nada, se o objeto a ser consultado é um objeto temporal versionado. Isso é feito através de uma análise semântica: os metadados são consultados para fazer tal verificação. Caso o objeto não seja temporal versionado, as consultas não serão efetuadas.

4.3 Metadados

O TVM incorpora novos conceitos à sua especificação, tais como propriedades e relacionamentos temporais, herança por extensão, versões, correspondência entre versões, classes temporais versionadas, entre outros. Quando se modela uma aplicação através do TVM, todos esses novos conceitos deverão ser definidos pelo usuário e armazenados, de modo que métodos das classes do modelo e consultas a base de dados possam utilizá-los.

Para armazenar as informações referentes a todo o esquema, ou seja, as informações acrescentadas pelo TVM, definiu-se em [MOR 2001a] um conjunto de classes que formam os metadados. A figura 4.3 apresenta essas classes.

A seguir, serão apresentadas as propriedades definidas para cada classe.

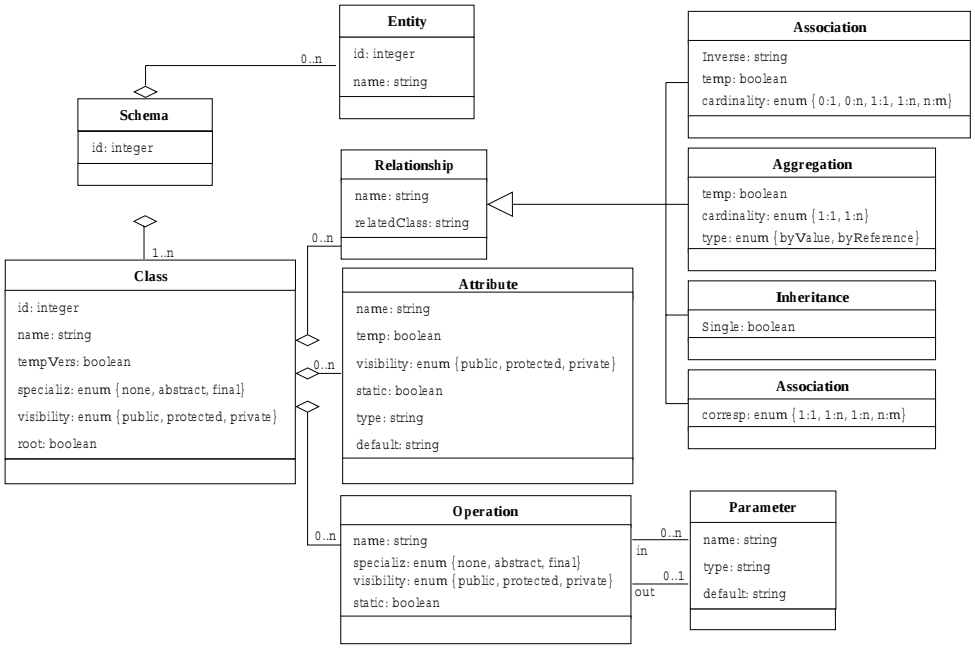


FIGURA 4.3 – Metadados do Modelo Temporal de Versões

- `Schema`: armazena os dados referentes ao esquema. Suas propriedades não foram especificadas.
- `Entity`: armazena os dados referentes às entidades do esquema. Possui um identificador e um nome. O identificador será utilizado para compor o `OID` dos objetos.
- `Class`: armazena os dados referentes às classes do esquema. Através dela pode-se recuperar o identificador e o nome da classe, se a mesma é temporal versionada, qual tipo de especificação foi definida para a classe, sua visibilidade e se é raiz de uma hierarquia de herança. O identificador será utilizado para compor o `OID` dos objetos. As especificações permitidas para uma classe são `none` (normal), `abstract` e `final`. As visibilidades que podem ser definidas são `public`, `protected` e `private`.
- `Attribute`: armazena as propriedades das classes. Pode-se recuperar o nome da propriedade, se a mesma é temporal, sua visibilidade (mesmas definidas para `class`), se é estática, qual o seu tipo de dado e valor *default* que deverá ser utilizado para casos de o usuário não informar o valor no momento da inserção.
- `Relationship`: armazena os relacionamentos entre as classes. São identificados o nome e a classe relacionada e pode ser especializada em três subtipos:
  - `Association`: possui o nome do relacionamento inverso, se é temporal e qual cardinalidade especificada para o mesmo;
  - `Aggregation`: informa se o relacionamento é temporal, a cardinalidade (`1:1` ou `1:n`) e o tipo de agregação (`byValue` ou `byReference`);

  - `Inherit`: indica se a herança por refinamento é simples ou múltipla;
  - `Extension`: indica a correspondência que será utilizada pela herança por extensão.

- `Operation`: armazena o nome da operação, sua especialização, visibilidade e se a mesma é estática. Pode possuir parâmetros de entrada e de saída.

- `Parameter`: armazena o nome, o tipo de dados e o valor *default* dos parâmetros de entrada e de saída das operações.

Para que a aplicação do usuário, definida através do TVM, tire proveito dos conceitos e recursos do modelo, é necessário que todos esses dados (metadados), referentes ao esquema, sejam inseridos após a definição do mesmo e antes que seja liberado para o usuário.

4.4 Exemplo

Para exemplificar a utilização da TVQL, considere o mesmo exemplo do capitulo 3 (figura 3.8). Pode-se observar um objeto versionado contendo versões de configurações de computadores e outro, contendo versões de dados específicos de notebook. Somente as propriedades `memoria` e `valor` de `computador` são temporais.

Considere que o histórico apresentado na figura 4.4 foi armazenado para a propriedade `valor` da versão `c1`.

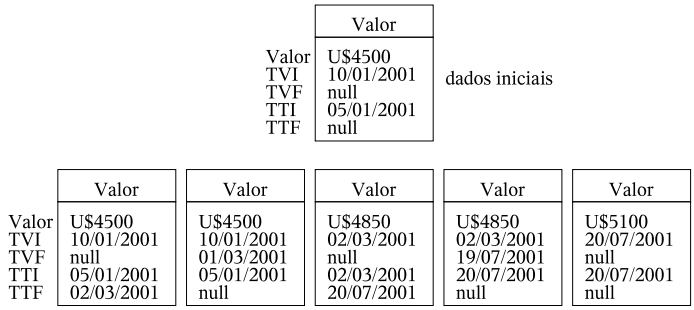


FIGURA 4.4 – Histórico da propriedade `valor` da versão `c1`

Com base na hierarquia de herança e de derivação, e no histórico acima apresentado, diversas consultas podem ser definidas para recuperar dados específicos de tempo e versão dos objetos. Alguns exemplos serão apresentados a seguir.

Suponha que o usuário deseja recuperar informações referentes às configurações de computadores cuja capacidade do HD seja superior a 10Gb. A consulta em TVQL será descrita da seguinte forma:

```
SELECT v.processador, v.HD, v.memoria, v.valor
FROM computador c, c.versions v
WHERE x.HD > 10
```

As configurações c2 e c4 retornariam como resultado dessa consulta. Para saber quais versões de notebook estão associadas às versões de configurações cuja memória é de 128Mb, a seguinte consulta é definida:

```
SELECT vn.bateria, vn.dispositivo
FROM computador c, c.versions vc, notebook n, n.versions vn
WHERE vc.memoria = 128 and
      vn.isDecendantOf(vc)
```

Como resultado têm-se as versões n2, n3 e n4 de notebook. É importante ressaltar que neste caso, como não foi utilizada a cláusula EVER, somente os valores correntes foram considerados para as propriedades temporais.

Para retornar as memórias válidas em "25/07/2001"e cujo estado é *working*, a seguinte consulta é definida:

```
SELECT v.memoria, v.valor
FROM computador c, c.versions v
WHERE v.isWorking and
      "25/07/2001" INTO v.memoria.vInterval
```

Nesse caso, a função isWorking é utilizada para verificar o estado das versões. A propriedade vInterval é usada em conjunto com o operador de condição into para comparar o instante de validade, informado na consulta, com o intervalo de validade das versões. Como a cláusula EVER não foi especificada, os valores retornados pela consulta serão os valores correntes.

Para recuperar o histórico dos valores da versão c4 de computador com seus respectivos rótulos de validade, a seguinte consulta será especificada:

```
SELECT EVER v.valor, v.valor.vInterval, v.valor.tInterval
FROM computador c, c.versions v
WHERE v.nickname = "c4"
```

O resultado da consulta será:

valor	TVI	TVF
U$4500	10/01/2001	01/03/2001
U$4850	02/03/2001	19/07/2001
U$5100	20/07/2001	null

Em algumas situações, é necessário combinar condições temporais com estados atuais do objeto. Por exemplo, deseja-se o histórico dos valores dos notebooks com bateria de duração de 2h e cuja memória seja de 128Mb. A consulta é construída da seguinte forma:

```
SELECT EVER vc.valor
FROM computador c, c.versions vc, notebook n, n.versions vn
WHERE vc.memoria = 128 and
      PRESENT (vn.duracao = 2) and
      PRESENT (vn.isDecendantOf(vc))
```

Um outro exemplo que poderia combinar condições temporais com estados atuais dos objetos seria recuperar os notebooks que em algum momento da sua vida tiveram memória de 128Mb. Nesse caso, a consulta seria definida da seguinte forma:

```
SELECT vn.bateria, vn.dispositivo
FROM computador c, c.versions vc, notebook n, n.versions vn
WHERE EVER (vc.memoria = 128) and
      vn.isDecendantOf(vc)
```

Nesse exemplo, como a cláusula EVER somente foi aplicada à propriedade memoria, será considerado o histórico dessa propriedade, sendo para as demais, considerados os valores correntes.

4.5 Conclusão

Aplicações definidas através do TVM deverão recuperar dados referentes a tempo e versão. Isso não é possível utilizando-se apenas a SQL. Para tanto, definiu-se uma linguagem específica para o modelo.

Essa linguagem é baseada no padrão SQL e utiliza basicamente a mesma sintaxe. Foram adicionadas à SQL propriedades e funções específicas para a recuperação de dados contendo tempo e versão. Uma das suas vantagens é a utilização dessa mesma linguagem para recuperar tanto dados temporais versionados quanto dados não temporais e não versionados.

A princípio, foram desenvolvidas somente funções para recuperar informações fundamentais para o usuário. Entretanto, funções adicionais, tais como função que retorna o número de versões de um objeto versionado, função que recupera a versão de acordo com o seu número, possibilidade de gerenciar diferentes granularidades através de funções de CAST entre outras, deverão ser especificadas futuramente. Apesar dessas limitações, a linguagem atende aos propósitos inicialmente estabelecidos.

A TVQL foi definida para ser independente de implementação, ou seja, não importa qual base de dados o usuário esteja usando, a consulta sempre será escrita da mesma forma. Para utilizá-la em bancos de dados tradicionais, faz-se necessário o mapeamento da mesma para a sintaxe utilizada pelo banco de dados.

5 *Extensible Stylesheet Language Transformation* - XSLT

XML (*Extensible Markup Language*) [W3C 2000] é um padrão desenvolvido pelo *XML Working Group* (W3C) utilizado principalmente para troca de documentos na WEB. Sua difusão se deve principalmente ao fato de ser uma linguagem flexível, auto-descritiva, de fácil compreensão e manipulação. Uma das grandes vantagens da XML é a separação dos dados de seu formato, permitindo, dessa forma, que um mesmo documento seja reutilizado de diferentes modos e em diferentes formatos de acordo com a necessidade do usuário / aplicação.

Várias aplicações, tais como portais web, bibliotecas digitais, comércio eletrônico, entre outras, que necessitam acesso integrado a diversas origens de informação (desde sistemas gerenciadores de banco de dados tradicionais (DBMS) a repositórios web semi-estruturados), aplicação direta dos recursos e baixo custo de manutenção em um ambiente dinâmico [CHR 2000], vêm tirando proveito dessa poderosa linguagem. No entanto, para aplicações que requerem troca de dados é necessário padronizar os documentos de forma que os mesmos sejam vistos e compreendidos igualmente por todos os envolvidos.

Geralmente, esses tipos de aplicações não precisam exatamente do mesmo conjunto de dados; em alguns casos, somente um subconjunto é necessário. Nesse contexto, a XSLT (*Extensible Stylesheet Language Tansformation*) [W3C 2001] surge como uma ferramenta capaz de transformar um documento XML em um pequeno conjunto de dados e formatar esses dados de acordo com o interesse do usuário / aplicação.

Recomendada pelo W3C, a XSLT foi inicialmente projetada para transformar um documento XML em outro [KAY 2000]. Entretanto, observou-se que essa linguagem poderia oferecer mais recursos do que o proposto a princípio. XSLT define um conjunto de regras de transformação que além de permitir transformar dados em relatórios legíveis, tal como HTML, ou em um subconjunto do documento XML original, ou ainda, em qualquer outro documento XML [OAS 2001], permite também transformar dados em *scripts* SQL, emails [MUE 2000] e em muitos outros formatos baseados em texto. É uma linguagem declarativa, expressa na sintaxe XML, sendo portanto, de fácil uso e leitura.

A XSLT oferece uma série de recursos para a manipulação de dados XML. No entanto, somente os recursos utilizados nesse trabalho serão apresentados.

5.1 Características Gerais

XSLT não é uma linguagem procedural convencional: consiste de uma coleção de regras *template* que definem a saída. Essa saída é produzida através do disparo de ações quando uma determinada condição é encontrada em uma regra *template*. A transformação dessas regras em um documento resultado é feita por um processador XSLT, cuja função é aplicar uma XSLT a um documento XML origem e produzir um documento resultado. Esse processo é mostrado de forma simplificada na figura 5.1.

Como visto anteriormente, a XSLT define um conjunto de regras *template*.

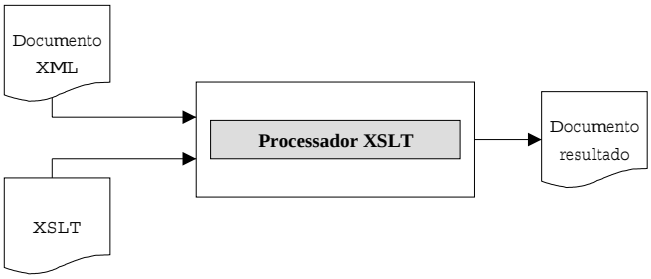


FIGURA 5.1 – Processador XSLT

As regras são compostas por uma seqüência de operações que deverão ser executadas para obter o documento resultado. Essas operações podem ser outras regras *template* ou expressões que o usuário deseja apresentar na saída. A estrutura básica das regras *template* é apresentada abaixo:

```
<xsl:template match="" >
   <!--  expressões do usuário -->
   <!--  regras template -->
</xsl:template>
```

É necessário processar todos os elementos do documento XML origem para produzir o documento resultado. O processo começa pelo elemento raiz, passando em seguida para todos os seus filhos de forma recursiva. O processador tenta encontrar um *template* que satisfaça a sua condição. No início do processo, a condição consiste em encontrar um *template* para o elemento raiz. O atributo `match` da regra *template*, composto por expressões XPATH, informa ao processador que aquela regra pertence a um determinado elemento. O mesmo processo acontece para todos os elementos do documento.

Considere o documento XML da figura 5.2. Esse documento possui descrições das tabelas de uma aplicação: nomes das tabelas, definição das respectivas propriedades, restrições e integridade referencial. Com base nesse documento, deseja-se produzir como resultado um *script* com instruções SQL CREATE TABLE para ser executado em um banco de dados DB2.

Como o processo se inicia pelo elemento raiz, o primeiro elemento a ser executado é o `tabelas`. O processador, tendo conhecimento de qual é o elemento raiz, busca uma regra *template* onde o elemento a ser processado coincida com o valor do atributo `match` da regra. Suponha que a seguinte regra tenha sido definida para o elemento `tabelas`:

```
<xsl:template match="tabelas">
   <xsl:apply-templates />
</xsl:template>
```

Através do elemento `<xsl:apply-templates />`, essa regra informa ao processador que todos os seus subelementos serão processados. O elemento `tabelas` possui somente um subelemento, `tabela`. Dessa forma, o processador tentará encontrar uma regra para esse subelemento. A regra encontrada pelo processador tem a estrutura abaixo:

```
<xsl:template match="tabela">
   CREATE TABLE <xsl:value-of select="nome"/>
```

```
<?xml version="1.0" encoding="US-ASCII"?>
<!DOCTYPE tabelas SYSTEM "tabelas.dtd">
<tabelas>
   <tabela>
      <nome> departamento </nome>
      <atributo tipo="Primary Key" nulos="nao">
         <nome> codigo </nome>
         <tipodado> integer </tipodado>
      </atributo>
      <atributo>
         <nome> nome </nome>
         <tipodado> varchar </tipodado>
         <tamanho> 30 </tamanho>
      </atributo>
   </tabela>
   <tabela>
      <nome> empregado </nome>
      <atributo tipo="Primary Key" nulos="nao">
         <nome> nome </nome>
         <tipodado> varchar </tipodado>
         <tamanho> 50 </tamanho>
      </atributo>
      <atributo tipo="Foreign Key" nulos="nao">
         <nome> dept </nome>
         <tipodado> integer </tipodado>
      </atributo>
   </tabela>
</tabelas>
```

FIGURA 5.2 – Documento XML com tabelas de uma aplicação

```
     (<xsl:apply-templates />)

     <!-- ADICIONA CHAVES PRIMARIA E ESTRANGEIRA -->
     ALTER TABLE <xsl:value-of select="nome"/>
     ADD <xsl:apply-templates select="atributo" mode="restricoes" />
   </xsl:template>
```

A regra definida para o elemento `tabela` é composta tanto por expressões do usuário quanto por outras regras *template*. Esse elemento contém as especificações das tabelas a serem geradas. Através dessa regra, o processo de criação do *script* será iniciado. Todas as expressões do usuário, como por exemplo CREATE TABLE, serão apresentadas na saída da mesma forma que foram definidas na XSLT.

O elemento `xsl:value-of` escreve o valor (conteúdo) do elemento informado no atributo `select`. Esse atributo é constituído de expressões XPATH. Como visto anteriormente, `apply-templates` indica que os subelementos do elemento corrente serão processados. Pode conter dois atributos: `select` e `mode`. O primeiro atributo sobrescreve a ação padrão de processar todos os subelementos, executando apenas o selecionado. Como elementos de um documento origem podem ser processados múltiplas vezes, produzindo diferentes resultados, o atributo `mode` seleciona a regra *template* a ser aplicada. Para isso, esse atributo deve conter o mesmo valor tanto na regra *template* quanto na `apply-templates`.

Os subelemento de `tabela` são `nome` e `atributo`. No entanto, somente foram definidas regras para o elemento `atributo`.

```
   <xsl:template match="atributo">
     <xsl:value-of select="nome"/>
     <xsl:value-of select="tipodado"/>
```

```
  <xsl:if test="string-length(tamanho)>0"> (<xsl:value-of select="tamanho"/>)
  </xsl:if>
  <xsl:if test="position()!=last()">,</xsl:if>
</xsl:template>
```

A instrução `xsl:if` permite processamento condicional em um *template*. É semelhante ao comando *if* utilizado em muitas linguagens de programação. Porém, não possui a cláusula *else*. Possui um atributo `test` que especifica a condição a ser analisada. Se o resultado da condição for verdadeiro, o seu conteúdo é processado. Caso contrário, nenhuma ação é disparada . A função `string-length`, incorporada do XPATH, retorna o número de caracteres de um *string*.

Várias regras *template* podem ser definidas para um mesmo elemento. Porém somente uma será aplicada: a mais específica. Por exemplo, poderia ser acrescentada a regra para tratar o último atributo da tabela. No momento que o último atributo fosse encontrado, a regra genérica especificada para todos os atributos seria ignorada e seria aplicada a regra específica incluindo no *script* o nome e o tipo de dado do atributo.

```
<xsl:template match="atributo[last()]">
  <xsl:value-of select="nome"/>
  <xsl:value-of select="tipodado"/>
</xsl:template>
```

A próxima regra a ser processada irá gerar as chaves primária e estrangeira das tabelas. Os elementos `atributo` deverão ser processados novamente, resultando em uma saída diferente das regras anteriores. Nesse caso, o atributo `mode` foi utilizado para indicar a regra *template* a ser aplicada.

```
<xsl:template match="atributo" mode="restricoes">
  <xsl:choose>
     <xsl:when test="@tipo='Primary Key'">
        Primary Key (<xsl:value-of select="nome" />)
     </xsl:when>
     <xsl:when test="@tipo='Foreign Key'">
        Foreign Key (<xsl:value-of select="nome" />)
     </xsl:when>
     <xsl:otherwise />
  </xsl:choose>
  <xsl:text/>
</xsl:template>
```

`xsl:choose` é outro tipo de instrução utilizada para processamento condicional. Essa regra seleciona uma entre diversas alternativas possíveis, e trata casos genéricos através do elemento `otherwise`. O atributo `test` faz a verificação condicional. O `@` é utilizado para indicar atributos dos elementos do documento origem.

Aplicando-se as regras descritas acima ao documento XML apresentado na figura 5.2, obtém-se o seguinte resultado:

```
CREATE TABLE  departamento
(codigo integer NOT NULL,
 nome varchar (30))

ALTER TABLE departamento
ADD Primary Key (codigo)

CREATE TABLE empregado
(nome varchar (50) NOT NULL,
 dept integer NOT NULL)

ALTER TABLE empregado
ADD Primary Key (nome) Foreign Key (dept)
```

5.2 Características Adicionais

Uma das funcionalidades a mais que a XSLT oferece é a possibilidade de criação de variáveis globais e locais, e utilização de parâmetros. As variáveis são usadas principalmente quando se deseja evitar a repetição de uma expressão em diversos locais. Sua sintaxe é a seguinte:

```
<xsl:variable name="" select="">
</xsl:variable>
```

O atributo `name` indica o nome da variável e o `select` informa o valor que a variável irá assumir. Variáveis definidas fora de uma regra *template* são consideradas globais, caso contrário, locais.

Suponha que no exemplo da criação do *script* das tabelas deseja-se que um tipo de dado *default* seja informado caso nenhum tipo tenha sido indicado pelo usuário. Nesse caso, pode-se criar uma variável que recebe esse valor, conforme definição abaixo. Ambas as definições são permitidas:

```
<xsl:variable name="tipoDefault" select="'string'">
<xsl:variable name="tipoDefault"> string </xsl:variable>
```

Em alguns casos, a passagem de parâmetro de uma regra *template* para outra é necessário. Para isso, utiliza-se o elemento `param` em conjunto com o elemento `with-param`. Da mesma forma que as variáveis, pode-se definir parâmetros globais e locais. Sua sintaxe é a seguinte:

```
<xsl:param name="" select="" />
```

O atributo `name` indica o nome da variável e o `select` informa o valor que a variável irá assumir. O elemento `with-param` será utilizado na chamada de uma regra *template*, quer seja pelo elemento `apply-templates` ou pelo `call-templates`. Esse último elemento chama uma regra especificada por um nome.

Voltando ao exemplo do *script* das tabelas, suponha que antes que as propriedades (atributos) sejam listadas, deseja-se colocar um comentário dizendo que aquelas propriedades pertencem à tabela `X`. As seguintes regras seriam construídas:

```
<xsl:template match="tabela">
   CREATE TABLE <xsl:value-of select="nome"/>
   <xsl:apply-templates>
      <xsl:with-param name="tabela" select="name" />
   </xsl:apply-templates>
 </xsl:template>

 <xsl:template match="atributo">
   <xsl:param name="tabela" />
   <xsl:comment>
      Propriedades da tabela <xsl:value-of select="\$tabela" />
   </xsl:comment>
   <xsl:value-of select="nome"/>
   <xsl:value-of select="tipodado"/>
   <xsl:if test="string-length(tamanho)>0"> (<xsl:value-of select="tamanho"/>)
   </xsl:if>
   <xsl:if test="position()!=last()">,</xsl:if>
 </xsl:template>
```

Tanto o valor da variável quanto o do parâmetro são acessados utilizando o prefixo $ antes dos seus nomes. O elemento comment é utilizado para escrever uma linha de comentário no documento resultado.

Outras funções úteis para a manipulação de dados sting são a concat, que permite concatenar dois ou mais valores, a substring, que retorna parte de um valor do tipo string, a substring-after que retorna parte de um string situado após a primeira ocorrência de um substring especificado, e a substring-before que retorna parte de um string situado antes da primeira ocorrência de um substring. Todas essas funções fazem parte do XPATH. A sintaxe e um exemplo de uso cada função são apresentadas a seguir:

```
concat (valor1, valor2, valor3, ...)
<xsl:value-of select="concat(nome, " ", tipodado)" />

substring (valor, inicio, tamanho)
<xsl:value-of select="substring(nome, 1, 5)" />

substring-after (valor, substring)
<xsl:value-of select="substring-after(condicao, "or")" />

substring-before (valor, substring)
<xsl:value-of select="substring-before(condicao, "or")" />
```

Considere que os exemplos serão aplicados no elemento atributo. A função concat irá retornar o nome e o tipo de dado do atributo. O segundo exemplo, função substring, retorna os cinco primeiros caracteres do nome do atributo. Para as funções substring-after e substring-before adicionou-se uma condição para cada atributo. Suponha que a condição (codigo < 100 or codigo > 500) tenha sido acrescentada ao atributo codigo fazendo com que essa propriedade possa apenas assumir valores inteiros menores que 100 ou maiores que 500. O resultado da função substring-after será codigo > 500 e o da substring-before será codigo < 100.

5.3 Conclusão

A grande difusão da XML se deve ao fato de ser essa uma linguagem auto-descritiva, flexível e de fácil manipulação, quer seja através de um simples editor ou de aplicações específicas. Definida como padrão pelo W3C, é bastante usada em aplicações WEB, principalmente para troca de informações. Sua principal vantagem se deve ao fato de possibilitar a separação de dados e apresentação.

Entretanto, um documento XML por si só, não apresenta grandes vantagens. Usuários e aplicações poderão ter acesso à informação contida no documento mesmo que o mesmo não esteja formatado adequadamente. No entanto, para que aplicações e usuários possam tirar um maior proveito dessa linguagem, é imprescindível transformá-la em formatos adequados as suas necessidades. Isso é possível através da linguagem XSLT, que transforma documentos XML em outros documentos XML, bem como em HTML, *scripts* SQL ou em outros formatos baseados em texto.

6 Especificação em XML

Desde a sua introdução, XML tem rapidamente crescido como um formato universal para publicação e troca de dados na WEB. Como um resultado, origens de dados, incluindo banco de dados objeto-relacional, estão agora lidando com uma nova classe de usuários: pessoas que gostariam de tratar diretamente com dados XML ao invés de serem forçados a lidar com um esquema de uma fonte de dados particular (por exemplo, objeto-relacional) e com uma determinada linguagem de consulta [CAR 2000].

Vários trabalhos têm sido desenvolvidos com tais objetivos, particularmente na área de banco de dados: documentos XML gerados a partir de uma base de dados, representação de visões de esquemas relacionais em documentos XML, transferência de dados entre documentos XML e banco de dados, entre tantos outros.

Sendo XML considerado um formato universal, pode-se utilizar essa linguagem como um método de formalização. Outros padrões, como por exemplo UML e ODMG podem também ser utilizados para tal fim.

Esse capítulo apresenta a especificação, ou melhor dizendo, a formalização do Modelo Temporal de Versões em XML. Essa especificação, além das características do modelo, inclui sua linguagem de definição de dados (DDL), bem como sua linguagem de consulta (TVQL). Como principais vantagens na utilização da XML destaca-se o fato de ser uma linguagem padrão, de fácil leitura e manipulação.

Como XML está sendo amplamente difundido, várias ferramentas já foram desenvolvidas para manipular dados XML, entre elas, destacando-se a XSLT, uma linguagem de transformação utilizada para gerar diferentes saídas tendo como base documentos XML.

A XSLT está sendo utilizada nesse trabalho para definir regras de transformação para as especificações do TVM e da TVQL. Como resultado, obtém-se um *script* em SQL com as características do modelo que poderá servir para estender bancos de dados com característica OO para temporais versionados, e uma documentação em HTML com instruções para a criação de esquemas do usuário que armazenarão dados com tempo e versão. No entanto, para cada Sistema de Gerenciamento de Banco de Dados deverá ser criado um conjunto de regras de transformação específico, uma vez que cada SGBD possui uma sintaxe específica para a definição e manipulação dos dados.

O presente trabalho inicialmente apresenta regras de transformação específicas para o DB2. Esse banco de dados foi o escolhido por ser o mais próximo ao padrão `SQL-92` no suporte aos tipos de dados temporais [SNO 99], e por estar esse trabalho incluso no projeto `IBM/Solectron` do qual esse grupo de pesquisa faz parte.

As seções seguintes irão apresentar como foram desenvolvidos os documentos XML e os arquivos XSLT, e os resultados obtidos com a aplicação das regras de transformação aos documentos XML gerados.

6.1 Modelo Temporal de Versões

A especificação do TVM em XML resulta na formalização do modelo, na independência de plataforma e na independência das sintaxes utilizadas pelos bancos de dados, uma vez que está sendo descrito em uma linguagem padrão. Além de serem definidas as classes do modelo, suas propriedades e relacionamentos temporais e não temporais, seus métodos, suas regras de validação e integridade, e seus procedimentos auxiliares, será também definida a linguagem de definição dos dados (DDL) de forma que o usuário tenha acesso à sintaxe que deverá ser utilizada para a construção do esquema de sua aplicação.

O resultado dessa especificação poderá ser utilizado de duas formas: (1) como base para usuários estenderem seus bancos de dados com característica OO para trabalharem com tempo e versão, sem que os mesmos tenham que consultar diretamente o modelo; (2) ou através de uma aplicação, que irá gerar um *script* em SQL, baseado no documento XML gerado com a especificação do modelo. Nesse caso, será criado um arquivo XSLT que será aplicado ao documento XML.

No primeiro caso, o usuário saberá quais classes deverá criar no banco de dados, seus relacionamentos com as demais classes, métodos, etc., baseado apenas no documento XML gerado para o modelo. A segunda opção, irá gerar automaticamente para o usuário toda a especificação do modelo na sintaxe específica de um banco de dados. A figura 6.1 apresenta de forma resumida o funcionamento da aplicação.

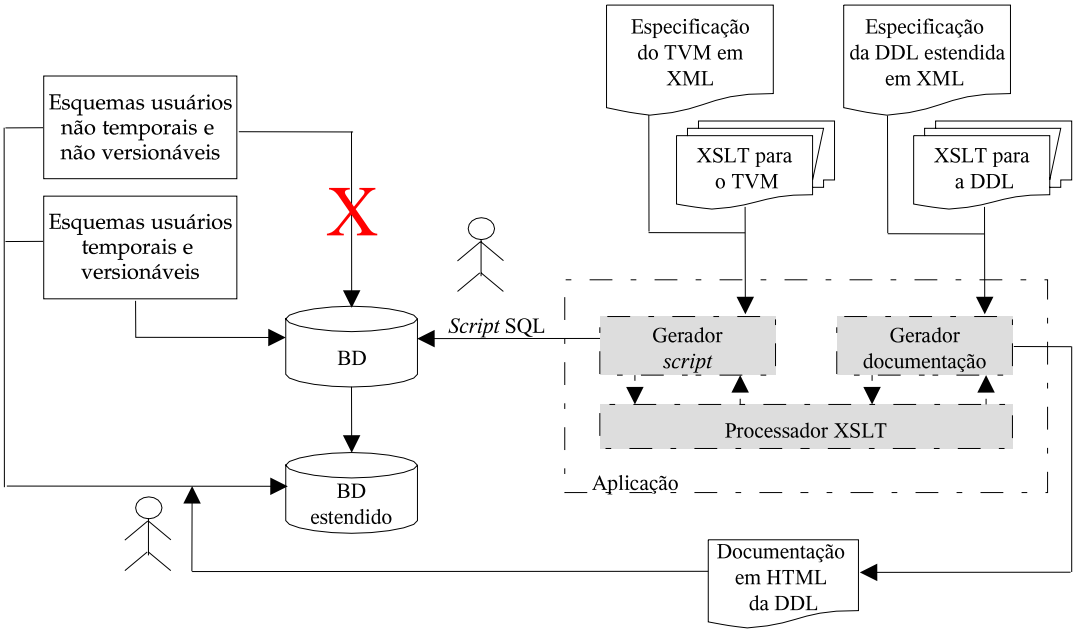


FIGURA 6.1 – Aplicação

Inicialmente serão gerados dois documentos XML: um com as características do modelo e outro com a estrutura da linguagem de definição dos dados. Esses dois documentos serão submetidos à aplicação. Em seguida, arquivos XSLT para a especificação do modelo e para a DDL serão criados e também submetidos à aplicação. Como cada banco de dados possui uma sintaxe própria para

a definição de dados, diversos arquivos XSLT poderão ser definidos: um para cada base de dados. O resultado será um *script* em SQL para a especificação do modelo com a sintaxe própria utilizada pelo banco de dados selecionado pelo usuário, e uma documentação para a DDL apresentando para o usuário a sintaxe que deverá ser utilizada para criar o esquema da sua aplicação na base de dados selecionada. O usuário poderá intervir no resultado do *script* antes que o mesmo seja executado no banco de dados para, por exemplo, incluir características não implementadas e / ou para otimizar o *script*. Esse *script* é apresentado no anexo C.

O objetivo desse trabalho é definir os documentos XML com a especificação do modelo e com a estrutura da DDL que serão utilizados pela aplicação, e as XSLT's para gerar o *script*, que irá estender o banco de dados tradicional para um temporal versionado, e a documentação da DDL. As seções seguintes irão apresentar os resultados obtidos no decorrer desse trabalho.

6.1.1 Documento XML

Dois documentos XML foram criados para especificar o modelo. Um deles para representar as características do TVM e o outro, para representar a linguagem de definição dos dados. Esses documentos estão ilustrados no anexo B.

Como o objetivo não é gerar instâncias XML e sim um único documento que apresenta a especificação do modelo, uma estrutura para definir regras de criação de documentos é desnecessária. No entanto, essa estrutura foi definida visando unicamente a validação do documento XML.

O primeiro documento a ser analisado é o documento que apresenta as características do modelo. Esse documento é composto por todas as classes da hierarquia do modelo, pelas classes que modelam os rótulos temporais, as classes que armazenam os apelidos do objeto, as classes que definem a estrutura do OID dos objetos, as classes dos metadados bem como pelos seus relacionamentos, propriedades e métodos. Sua estrutura básica é apresentada a seguir:

```
<metaSchema>
   <classes>
      <class>
         <name> </name>
         <comment> </comment>
         <superClass> </superClass>

         <attributes>
         </attributes>

         <operations>
         </operations>

         <relationships>
         <relationships>
      </class>
   </classes>

   <operations>
   </operations>
</metaSchema>
```

A especificação (ou meta esquema) do modelo, `metaSchema`, é constituída pelos elementos `classes`, onde as classes referenciadas anteriormente estão definidas com suas propriedades (`attributes`), métodos (`operations`) e relacionamentos (`relationships`), e `operations`, que representam as operações

adicionais que são necessárias para realizar o controle dos dados e que não foram definidas no TVM. Esse último elemento não está sendo utilizado nesse trabalho.

Além das propriedades, métodos e relacionamentos das classes definidas em `class`, é informado o nome da classe (`name`), a função que a mesma realiza (`comment`) e a classe da qual irá herdar propriedades e métodos (`superClass`), sendo os dois últimos elementos opcionais. Através de `superClass` é possível construir a hierarquia de herança do modelo.

Para uma melhor visualização da especificação do modelo, considere a classe `Name` extraída do documento XML e que está ilustrada abaixo. Essa classe é uma classe pública que possui somente uma propriedade, `value`, do tipo `string` não temporal, e que não permite valores nulos. Implementa três operações: `name`, `getClassName` e `getName`. A primeira é um construtor que possui um parâmetro de entrada. As outras duas operações não possuem parâmetros de entrada e retornam um valor do tipo `string`. Todos as operações são definidas como métodos e nenhuma delas sobrescreve os métodos da classe pai. A linguagem na qual serão implementadas é a SQL. O único relacionamento definido para essa classe é um relacionamento de associação com a classe `Object`. A especificação da classe `Name` é a seguinte:

```
<class visibility="public">
   <name> Name </name>
   <comment> armazena os nomes (apelidos) atribuídos aos objetos </comment>
   <attributes>
      <attribute temporal="False" nulls="False">
         <name> value </name>
         <datatype> string </datatype>
         <length> 20 </length>
      </attribute>
   </attributes>
   <operations>
      <operation type="method" override="False">
         <name> Name </name>
         <comment> construtor </comment>
         <parameters>
            <in>
               <name> nam </name>
               <datatype> string </datatype>
               <length> 20 </length>
            </in>
         </parameters>
         <body language="SQL"/>
      </operation>
      <operation type="method" override="False">
         <name> getClassName </name>
         <comment> retorna o nome da classe </comment>
         <parameters>
            <return>
               <datatype> string </datatype>
               <length> 30 </length>
            </return>
         </parameters>
         <body language="SQL"/>
      </operation>
      <operation type="method" override="False">
         <name> getName </name>
         <comment> retorna o apelido atribu_do ao objeto </comment>
         <parameters>
            <return>
               <datatype> string </datatype>
               <length> 30 </length>
            </return>
         </parameters>
         <body language="SQL"/>
      </operation>
```

```
   </operations>
   <relationships>
      <relationship type="association">
         <name> nickNameOf </name>
         <cardinality> 1:1 </cardinality>
         <relatedClassName> Object </relatedClassName>
         <inverse> nickname </inverse>
      </relationship>
   </relationships>
</class>
```

Quando as operações são definidas, é possível especificar também as restrições que serão impostas aos dados (`conditions`) e a funcionalidade das mesmas (`comment`). No exemplo acima, nenhuma restrição foi definida para os métodos da classe `Name`.

Uma observação importante a ser feita é que no modelo, as propriedades cujos tipos de dados são `string` não tiveram seu tamanho estabelecido. No entanto, durante a especificação em XML, esses tamanhos foram definidos para facilitar a geração do *script*, uma vez que os bancos de dados necessitam do tamanho da propriedade especificada.

Como pode-se observar, a estrutura especificada para o modelo é autodescritiva e intuitiva, de fácil leitura por parte do usuário. O mesmo não acontece com o documento XML especificado para a DDL. Isso se deve ao fato de se estar representando uma sintaxe: a sintaxe para a definição do esquema do usuário apresentada na figura 3.12. Esse documento foi definido para gerar, em conjunto com a XSLT da DDL, a documentação que irá auxiliar o usuário na criação das suas classes.

O documento XML com a estrutura da DDL também está especificado no anexo B. O documento foi criado levando-se em consideração a seqüencia das cláusulas da linguagem de definição dos dados especificada para o modelo. Para a definição de classes temporais versionadas, algumas cláusulas são obrigatórias e nesse caso, tentou-se diferenciar quais seriam aplicadas às classes temporais versionadas e quais seriam aplicadas às não temporais e não versionadas. O pequeno trecho, apresentado abaixo, faz parte do documento XML da DDL e será rapidamente explicado para que se entenda como o mesmo foi definido:

```
<inherit optional="true">
   <inheritedBy optional="true" temporalVersion="false"> byExtension </inheritedBy>
   <name> className </name>
   <correspondence optional="true" choice="true" temporalVersion="true">
      <choice> 1:1 </choice>
      <choice> 1:n </choice>
      <choice> n:1 </choice>
      <choice> n:n </choice>
   </correspondence>
</inherit>
```

A cláusula `inherit` é uma cláusula opcional aplicada tanto a classes temporais versionadas quanto a classes não temporais e não versionadas. Essa cláusula poderá vir acompanhada opcionalmente por `byExtension`, se a classe que estiver sendo definida não for temporal versionada. Em seguida, será definido obrigatoriamente o nome da classe. A cláusula `correspondence` será aplicada opcionalmente às classes temporais versionadas, sendo permitido para essa cláusula um dos valores listados no elemento `choice`. Seguindo essa mesma linha de raciocínio, o documento XML da DDL foi gerado observando-se também quais cláusulas permitem ocorrências múltiplas ou opcionais.

Todos os documentos XML foram gerados e validados através da ferramenta XML Spy versão 4.1 [ALT 2001].

6.1.2 Sintaxe DB2

Antes de apresentar os arquivos XSLT criados para a especificação do TVM e da DDL, é importante analisar as características de orientação à objetos implementadas pelo DB2 e suas limitações. A versão utilizada nesse trabalho é a 7.2 objeto-relacional. Todas as sintaxes e especificações do DB2 podem ser encontradas em [IBM 2001, IBM 2001a, IBM 2001b, FUH 99, CAR 99, CHA 98].

Além de esquemas de aplicações tradicionais, que utilizam o modelo relacional, o DB2 permite que aplicações modeladas através dos conceitos de orientação à objetos sejam também implementadas. Dessa forma, classes definidas para tais aplicações podem ser especificadas com suas respectivas propriedades e métodos. É permitido ao usuário criar tipos estruturados para utilizar como tipos de dados de algumas propriedades, funções para manipular os dados, restrições de integridade referencial, valores únicos para propriedades, entre outras funcionalidades.

As classes são representadas no banco de dados como tabelas tipadas, isto é, tabelas associadas a um tipo estruturado específico. Inicialmente são definidos tipos estruturados utilizando o comando CREATE TYPE e em seguida define-se uma tabela que será vinculada ao tipo estruturado criado.

Para introduzir a sintaxe básica para a utilização das características objeto-relacional do DB2, considere a aplicação ilustrada na figura 6.2. O esquema define três classes: pessoa, empregado e departamento. A classe empregado é uma sub-classe de pessoa e possui um relacionamento de associação com a classe departamento.

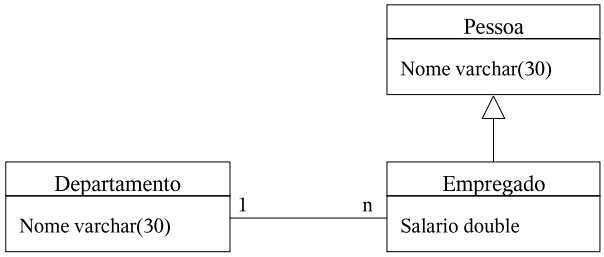


FIGURA 6.2 – Exemplo modelo de dados para DB2

A definição do modelo exemplo na sintaxe do DB2 se dará da seguinte forma:

```
CREATE TYPE departamento_t AS
 (nome varchar (30))
  INSTANTIABLE
  REF USING INTEGER
  MODE DB2SQL;

CREATE TABLE departamento OF departamento_t
 (REF IS OID USER GENERATED);

CREATE TYPE pessoa_t AS
 (nome VARCHAR (30))
```

```
  REF USING INTEGER
  MODE DB2SQL;

CREATE TABLE pessoa OF pessoa_t
 (REF IS OID USER GENERATED);

CREATE TYPE empregado_t UNDER pessoa_t AS
 (salario DOUBLE,
  dept REF (departamento_t))
  FINAL
  MODE DB2SQL;

CREATE TABLE empregado OF empregado_t
 UNDER pessoa
 INHERIT SELECT PRIVILEGES
 (dept WITH OPTIONS SCOPE departamento);
```

Para cada classe é criado um tipo estruturado e uma tabela associada a esse tipo. A cláusula `INSTANTIABLE`, cláusula *default*, indica que a classe poderá ter instâncias. Para definir classes abstratas (que não possuem instâncias), utiliza-se a cláusula `NOT INSTANTIABLE`.

A cláusula `FINAL`, definida no tipo `empregado`, indica que essa classe não poderá ter subclasses associadas a ela. O *default*, `NOT FINAL`, indica que as classes podem ter subclasses definidas.

Quando se está trabalhando com classes no DB2, deve-se obrigatoriamente definir um identificador de objeto (OID) para as instâncias das classes. O comando `REF USING INTEGER` em `CREATE TYPE` informa que o identificador que será gerado é do tipo inteiro. Caso seja omitido, o identificador, por *default*, será do tipo `varchar(16)`. O comando `REF IS OID USER GENERATED`, definido em `CREATE TABLE`, atribui o nome `OID` ao identificador e indica que o mesmo será gerado pelo usuário.

A cláusula `MODE DB2SQL` é obrigatória e especifica o modo de visão tipada. Este é o único modo atualmente suportado [IBM 2001].

A classe `empregado` tem um relacionamento com `departamento`. Esse relacionamento é implementado como uma referência através da cláusula `REF` em `CREATE TYPE`. O comando `WITH OPTIONS SCOPE` especifica a opção adicional `SCOPE` à coluna. Dessa forma, `dept` poderá utilizar o operador *dereference* para acessar diretamente os dados da tabela referenciada. Por exemplo, uma consulta para retornar os dados dos empregados e o nome do setor no qual trabalham, poderia ser escrita da seguinte forma utilizando esse operador:

```
SELECT nome, salario, dept->nome
FROM empregado
```

Uma observação importante a ser feita é que, se a instrução acima definida, for executada por linha de comando, é necessário escrevê-la entre aspas (”).

O conceito de herança por refinamento também é implementado pelo DB2. A cláusula para a definição de herança é a `UNDER` indicando a qual classe está sendo vinculada a nova classe. Todas as propriedades, métodos e o identificador definidos na classe pai são herdados pela classe filha. O único tipo de herança permitido é o de herança simples. No exemplo acima, a classe `empregado` herda todas as características definidas em `pessoa`, incluindo, inclusive, o identificador definido para a pai.

Um ponto importante que deve ser ressaltado é que quando se inclui dados em `pessoa` e em seguida deseja-se acrescentar os dados específicos de `empregado`

para essa pessoa, será gerado um novo OID para empregado diferente do gerado para pessoa, e os dados inseridos em pessoa serão novamente solicitados, parecendo se tratar de instâncias diferentes. Por exemplo: deseja-se inserir o empregado José Fernandes com salário inicial de R$ 2.500,00. As tabelas 6.1 e 6.2 indicam como o processo funcionaria.

TABELA 6.1 – Tabela com dados de pessoa

OID	Nome
30	José Fernandes

TABELA 6.2 – Tabela com dados de empregado

OID	Nome	Salario	Dept
10	José Fernandes	2.500,00	20

Observa-se que o OID de pessoa é diferente do OID de empregado. Na realidade, houve uma duplicação dos dados. Uma solução para esse problema seria definir a classe pessoa como abstrata, não permitindo que instâncias fossem incluídas na mesma. Somente empregado seria instanciado, evitando dessa forma, a duplicação de informações.

Quando se consulta dados de uma classe de uma hierarquia de herança, todos os dados das subclasses são retornados. Para recuperar somente os dados específicos de uma classe que possua subclasses, deve-se utilizar a cláusula ONLY na consulta. Abaixo, são apresentadas duas consultas: a primeira, retorna os dados referentes à classe pessoa e a sua subclasse empregado, e a segunda, retorna apenas os dados referentes à pessoa.

```
SELECT OID, NOME
FROM PESSOA

SELECT OID, NOME
FROM ONLY(PESSOA)
```

Uma das maneiras de realizar o controle de acesso às classes e suas propriedades e métodos, é através da cláusula INHERIT SELECT PRIVILEGES, especificada na definição da classe (tabela tipada). Essa cláusula estabelece que as subclasses serão inicialmente acessadas pelos mesmos usuários e grupos definidos para a super classe.

Pode-se também utilizar o comando GRANT para permitir a usuários e grupos de usuários acessarem um determinado banco de dados , ou permitir aos mesmos o direito a inserção, exclusão, alteração e consulta a dados de tabelas de um banco de dados. A sintaxe é apresentada a seguir.

```
GRANT CONNECT ON DATABASE
   TO [USER, GROUP] [nome_usuario_grupo] [PUBLIC]

GRANT [INSERT, DELETE, SELECT, UPDATE] ON nome_tabela
   TO [USER, GROUP] [nome_usuario_grupo] [PUBLIC]
```

O primeiro grant permite que um grupo ou usuário se conecte a um determinado banco de dados. Vários grupos e usuários poderão ser definidos no mesmo comando. A cláusula public indica que o privilégio será concedido a todos os usuários.

O segundo comando garante a usuários e grupos de usuários direito a inclusão, exclusão, consulta e atualização de dados em uma determinada tabela.

Ao se criar um tipo estruturado, o DB2 automaticamente gera um conjunto de métodos que podem ser usados para instanciar, recuperar e modificar os dados do tipo estruturado. São denominados respectivamente de função construct, métodos mutator e métodos observer.

A função construct é gerada sem nenhum parâmetro. Ao instanciar um objeto, todos os seus valores são inicializados com null. Um método mutator e um observer são gerados para cada propriedade do objeto. Com base no modelo criado para exemplo, o banco de dados irá gerar os métodos para as propriedades nome, salario e dept para o tipo empregado_t.

Toda classe pode ter associada a si métodos, os quais serão herdados pelas subclasses. Os métodos são definidos utilizando o comando CREATE METHOD. Porém, de acordo com [IBM 2001], os métodos só poderão ser utilizados para tipos estruturados identificados como tipos de uma determinada propriedade. Para exemplificar o uso de métodos, definiu-se para a classe empregado um método getsal para recuperar o salário de um determinado empregado:

```
ALTER TYPE empregado_t
ADD METHOD getsal (n varchar(30))
RETURNS DOUBLE
LANGUAGE SQL
READS SQL DATA
NO EXTERNAL ACTION
DETERMINISTIC;

CREATE METHOD getsal (n varchar(30)) FOR empregado_t
RETURN SELECT salario FROM empregado WHERE nome=n;
```

Primeiramente, acrescentou-se ao tipo empregado_t, a definição do método getsal através do comando ALTER TYPE e em seguida, criou-se o método propriamente dito. As cláusulas utilizadas na definição do método são apresentadas a seguir:

- RETURNS: especifica o tipo de retorno do método. No caso do exemplo, o valor retornado será do tipo double;
- LANGUAGE SQL: define que o método será escrito em SQL;
- READS SQL DATA: permite a inclusão de instruções SQL que não irão modificar os dados;
- NO EXTERNAL ACTION: especifica que o método não executará nenhuma ação que modifique o estado de um objeto não gerenciado pelo banco de dados;
- DETERMINISTIC: indica que o método sempre irá retornar o mesmo resultado após inúmeras chamadas utilizando os mesmos argumentos.

Através dos métodos, pode-se definir outros construtores além do gerado pelo banco de dados. No entanto, deve-se observar que a assinatura deve ser única.

Além dos métodos, o usuário pode definir procedimentos (procedures) e funções definidas pelo usuário (user-defined-functions) para manipular os dados dos objetos. Algumas diferenças existem nessas três formas de manipulação dos dados. A principal delas é que os métodos estão diretamente ligados às classes, e portanto, são herdados pelas suas subclasses. Os procedimentos não estão diretamente associados às classes. Possuem parâmetros de entrada e de saída, mas não retornam um tipo de dado. As user-defined-functions (UDF's), assim como os procedimentos, não estão ligados às classes. Possuem apenas parâmetros de entrada e permitem que seja retornado um valor do tipo de dado definido para a função.

Como visto anteriormente, cada classe requer um identificador de objeto. Quando se trabalha com tabelas comuns, é possível definir que esses valores serão gerados automaticamente pelo banco de dados, através do comando generated always as identity. Esse mesmo comando não é válido para gerar os OID's para as classes. Uma forma de evitar que o usuário se preocupe em gerar identificadores únicos para os objetos é através da utilização de seqüencias. Uma seqüência é um objeto do banco de dados que permite a geração automática de valores. Ao contrário de uma propriedade de coluna de identidade, a seqüência não é vinculada a uma determinada propriedade da classe.

Tendo como base a classe empregado, considere que a seguinte seqüencia tenha sido criada para gerar os valores dos OID's:

```
CREATE SEQUENCE seq_empregado AS integer
START WITH 1
INCREMENT BY 1
NO MAXVALUE
NO CYCLE
```

A seqüencia seq_empregado irá gerar valores inteiros iniciando com o valor 1 (um) e sendo incrementado a cada novo ciclo de 1 (um). A cláusula NO MAXVALUE especifica que o valor máximo gerado pela seqüencia é o máximo assumido pelo tipo associado à mesma. NO CYCLE especifica que não serão gerados valores para a seqüência quando o valor máximo ou mínimo da seqüência tiver sido atingido.

Como o valor da seqüencia não está diretamente associado ao OID definido para a classe, é necessário fazê-lo no gatilho (*trigger*) de inserção dos dados da classe. Esse gatilho será executado toda vez que o usuário instanciar um objeto. A *trigger* abaixo faz a associação do valor da seqüencia ao OID da classe empregado:

```
CREATE TRIGGER insere_empregado
NO CASCADE BEFORE INSERT ON empregado
REFERENCING NEW AS N
FOR EACH ROW MODE DB2SQL
fnc:BEGIN ATOMIC
  SET N.OID=empregado_t(NEXTVAL FOR seq_empregado);
END
```

As principais cláusulas especificadas na *trigger* são:

- NO CASCADE BEFORE: especifica que a ação definida será executada antes que qualquer mudança seja aplicada ao banco de dados;
- REFERENCING NEW AS: especifica como será identificada a nova linha a ser inserida. No exemplo acima, a linha é identificada por `N`;
- FOR EACH ROW MODE DB2SQL: especifica que a ação definida na *trigger* será aplicada uma vez para cada linha da tabela;
- NEXTVAL: gera o próximo valor da seqüencia. Como o `OID` é definido em um tipo estruturado, deverá ser feito um `cast` do valor gerado para o tipo estruturado ao qual a classe está associada, para em seguida, armazená-lo na propriedade da classe.

Algumas restrições podem ser especificadas para as propriedades das classes, como por exemplo valores nulos, valores *default* e *check constraints*. Essas restrições são incluídas no momento da criação da tabela tipada (classe) e estão especificadas abaixo [CHA 98]:

- Valores nulos (`NOT NULL`): indica que uma determinada propriedade não poderá conter valor nulo. Qualquer tentativa de inserção ou atualização dessa propriedade com valor `null` falhará;
- Valores *default*: quando uma propriedade é definida com um valor *default*, se nenhum valor for especificado para essa propriedade no momento da criação da instância, será assumido o valor *default* para a propriedade;
- *Check constraints*: essa restrição é aplicada a uma propriedade delimitando valores válidos para a mesma. Contém um predicado ou uma combinação de predicados conectados por `and` ou `or`, denominados *check condition*, que são verificados no momento de inserção ou atualização de dados.

Considere por exemplo, que na classe `empregado` a propriedade `salario` não pode assumir valores nulos e que o valor *default* definido para `dept` é 1 (um). Também é definida outra restrição para `dept`: os valores que a propriedade pode assumir estão entre 1 (um) e 10 (dez). A seguinte definição deverá ser realizada:

```
CREATE TABLE empregado OF empregado_t
UNDER pessoa
INHERIT SELECT PRIVILEGES
(salario NOT NULL,
 dept WITH DEFAULT 1,
 CHECK CONSTRAINT cc_dept CHECK (dept >= 1 and dept<=10),
 dept WITH OPTIONS SCOPE departamento);
```

A princípio, essas são as principais características objeto-relacionais utilizadas nesse trabalho e que são implementadas pelo banco de dados DB2.

A seguir serão apresentados os arquivos XSLT especificados para o TVM e para a DDL e serão definidas as regras de mapeamento utilizadas para estender a base de dados tradicional para uma base temporal versionada. Algumas características adicionais que não foram abordadas nessa seção serão definidas na seção 6.1.4.

6.1.3 Regras de mapeamento

Com base nas características objeto-relacionais do DB2, apresentadas na seção anterior, foram definidas regras de mapeamento do TVM para o DB2, de modo que usuários possam definir suas aplicações e fazer uso dos conceitos de tempo e de versão.

Inicialmente, apresenta-se os tipos de dados suportados pelo banco de dados. A tabela 6.3 apresenta os tipos básicos implementados pelo DB2.

TABELA 6.3 – Tabela com tipos de dados básicos do DB2

Datatypes
bigint
blob
character (char)
clob
date
decimal
double
integer (int)
long varchar
real
smallint
time
timestamp
varchar

O TVM especifica algumas propriedades do tipo temporal. Esses dados temporais podem ser definidos como instantes ou intervalos. Um instante é uma localização fixa na linha de tempo, e um intervalo é uma duração direcional não fixada na linha de tempo [SNO 99]. Com base nos tipos de dados implementados pelo DB2, um instante poderá ser representado por:

- *Date*: armazena o ano, mês e dia de um instante;
- *Timestamp*: armazena os mesmo dados que o tipo *date*, ou seja, ano, mês e dia de um instante, mais hora, minuto e segundo, e um número de dígitos de fração de segundo correspondendo aos microsegundos. Esse tipo deverá ser utilizado quando se desejar armazenar o tempo com uma melhor precisão;
- *Time*: armazena a hora, minuto e segundo e possui um número opcional de dígitos de fração de segundos.

Uma das principais limitações do DB2 é a ausência do tipo lógico (*boolean*) e dos tipos de coleções *set*, *bag*, *list* e *array*.

O Modelo Temporal de Versões define algumas de suas propriedades com tipos de dados não suportados pelo DB2: *boolean*, *set* e *instant*. Assim sendo, é necessário representá-los através de um dos tipos aceitos pelo banco de dados.

Os tipos definidos como lógicos podem ser especificados de duas formas: como tipo `char` ou através da definição de um tipo distinto (*user-defined distinct type*-UDT). Os tipos distintos são tipos definidos pelo usuário baseados em um dos tipos básicos (tabela reftabelaTipoDados) suportados pelo DB2. Ambas as

definições necessitam adicionar um *check constraint* à propriedade para que a mesma assuma somente os valores `F` ou `T`. Poderia ser criado o seguinte tipo distinto para definir as propriedades lógicas.

```
CREATE DISTINCT TYPE tipoBooleano AS CHAR WITH COMPARISONS
```

A cláusula `with comparisons` informa, ao DB2, que as funções de suporte às operações de comparação sobre instâncias dos tipos distintos deverão ser geradas pelo banco de dados. A palavra `boolean` não pode ser utilizada como um tipo distinto definido pelo usuário por se tratar de uma palavra reservada do banco de dados para futura implementação desse tipo de dado.

Uma propriedade poderia então ser definida através de uma das formas apresentadas abaixo:

```
propriedade CHAR

propriedade tipoBooleano
```

Optou-se nesse trabalho definir as propriedades lógicas do modelo pela primeira forma que é a mais simples para o usuário, uma vez que quando se trabalha com tipos distintos, deve-se realizar um `cast` para a manipulação de tais dados.

As propriedade definidas no TVM como `instant` podem ser especificadas de três formas: `date`, `timestamp` ou `time`. Tais propriedades foram implementadas como `timestamp` que é o tipo que armazena o valor com maior precisão.

O TVM também define algumas propriedades como `object`. Propriedades desse tipo podem assumir qualquer valor, como por exemplo, inteiro, real, boolean, string, entre outros. No mapeamento, todas essas propriedades foram definidas como `varchar`. Como o tipo de dado da propriedade é armazenado no metadados, é possível fazer um `cast` para o tipo correto da propriedade.

Para as propriedades temporais, os tipos definidos continuam os mesmos, exceto para aqueles que não são implementados pelo banco de dados. Nesse último caso, uma das opções sugeridas nesse trabalho será utilizada.

Os valores temporais serão armazenados em `InstantAttribute`. Para isso, faz-se necessário relacionar a classe do usuário que tenha propriedades temporais com essa classe. Suponha, por exemplo que na classe `empregado`, a propriedade `salario` seja temporal. Na classe `empregado` sempre estará armazenado o valor atual do salário do empregado, e na classe `InstantAttribute` estará todo o histórico da propriedade, inclusive o valor corrente.

Como uma mesma classe definida pelo usuário pode ter mais de uma propriedade temporal, adicionou-se à classe `InstantAttribute` a propriedade `property`, onde será armazenado a qual propriedade da classe do usuário o valor pertence. A tabela 6.4 apresenta um exemplo do histórico do salário de um determinado empregado armazenado fisicamente.

Os tipos de dados `set` foram divididos em dois grupos. São eles:

1. tipos `set` temporais;

2. tipos `set` não temporais.

TABELA 6.4 – Histórico do salário de um empregado

OID	tvOID	Property	Value	TVI	TVF	TTI	TTF
1	10,10,1	salario	1000	01/01/2000	null	01/01/2000	10/01/2000
2	10,10,1	salario	1000	01/01/2000	09/01/2000	01/01/2000	null
3	10,10,1	salario	1500	10/01/2000	null	10/01/2000	20/03/2000
4	10,10,1	salario	1500	10/01/2000	19/03/2000	10/01/2000	null
5	10,10,1	salario	2000	20/03/2000	null	01/01/2000	null

Os tipos `set` temporais são implementados da mesma forma que os tipos temporais simples. A classe do usuário que possui uma propriedade `set` temporal deverá estar relacionada com `InstantAttribute` de forma que o histórico dos seus dados sejam armazenados. Pode ocorrer a princípio um questionamento: como se saberá qual conjunto é válido em um determinado instante? A figura 6.5 ilustra um exemplo que explica essa questão.

TABELA 6.5 – Histórico do salário de um empregado

OID	tvOID	Property	Value	TVI	TVF	TTI	TTF
1	10,10,1	descendente	20,20,2	10/01/2000	null	10/01/2000	null
2	10,10,1	descendente	20,20,4	10/01/2000	null	10/01/2000	20/03/2000
3	10,10,1	descendente	20,20,5	15/02/2000	null	15/02/2000	null
4	10,10,1	descendente	20,20,4	10/01/2000	20/03/2000	10/01/2000	20/03/2001

No exemplo acima, temos que os descendentes válidos em janeiro são `20,20,2` e `20,20,4`. Em fevereiro, acrescentou-se `20,20,5` como descendente, ficando portanto, os três valores válidos. O conjunto corrente de descendentes é `20,20,2` e `20,20,5`.

Para os tipos `set` não temporais, será criada uma classe que armazenará o conjunto de dados pertencentes a propriedade.

TABELA 6.6 – Histórico do salário de um empregado

OID	tvOID	Value
1	30,10,1	10
2	30,10,1	20
3	30,10,1	30
4	40,10,1	1
5	40,10,1	5

Os relacionamentos de associação e agregação, definidos no TVM e nos metadados, foram implementados como referências.

O identificador dos objetos do Modelo Temporal de Versões continua sendo composto pela classe, entidade e versão. Esse identificador é gerado automaticamente no momento da instanciação. Ele é um *string* de tamanho 17, sendo que a cada 5 unidades são armazenados seqüencialmente os valores da classe, entidade e versão, separados por vírgula.

Nem todos os métodos foram implementados no DB2. No entanto, foram estabelecidas as seguintes regras para cada classe do Modelo Temporal de Versões:

- construtores e métodos sem dado de retorno: implementados através de `proce- dure`;
- métodos com dado de retorno: implementados como UDF's;
- métodos `get` das propriedades da classe sem acesso aos metadados: criação automática pelo banco de dados dos respectivos métodos `observer`;
- métodos `get` das propriedades da classe com acesso aos metadados: implementados da mesma forma que métodos com ou sem retorno de dado;
- métodos `set` das propriedades da classe: criação automática pelo banco de dados dos respectivos métodos `mutator`.

A tabela 6.7 sintetiza as regras definidas para o mapeamento do TVM para o DB2.

TABELA 6.7 – Tabela resumo do mapeamento para o DB2

Definição modelo	**Mapeamento**
Propriedades `instant`	definidas como `timestamp`
Propriedades `object`	definidas como `varchar (10)`. Antes de serem manipuladas, deverá ser feito um `cast` para o tipo apropriado
Propriedades `boolean`	definidas como `char` com `check constraint` permitindo somente a inclusão dos valores `T` e `F`
Propriedades `set` não temporais	criação de uma tabela específica para armazenar os valores da propriedade
Propriedades `set` temporais	referência a `instantAttribute`
Propriedades temporais	referência a `instantAttribute`
Relacionamento de associação	referência à tabela associada
Relacionamento de agregação	referência à tabela agregada
Herança por refinamento	cláusula `under`
Classes	tabelas tipadas
Métodos construtores e sem retorno	procedimentos
Métodos com retorno	*user-defined-function*

Criou-se um banco de dados com as classes, propriedades, métodos e metadados do modelo. Para que aplicações do usuário acessem os metadados, é necessário que privilégios de conexão ao banco de dados gerado para o modelo, e privilégios de inserção, exclusão, alteração e consulta aos dados sejam dados aos usuários e/ou grupos de usuários através do comando GRANT apresentado na seção anterior.

Os métodos das classes do TVM implementados no decorrer desse trabalho, estão apresentados no anexo D.

6.1.4 Arquivo XSLT

No capítulo 5 foram apresentadas as principais características da linguagem XSLT, o que torna redundante a explicação detalhada do arquivo de transformação definido para o mapeamento do modelo para o DB2, e do arquivo de transformação para a linguagem de definição de dados que será utilizada pelo usuário. A seguir serão introduzidos os arquivos XSLT para o TVM e para a DDL.

Modelo TVM

O arquivo XSLT definido para o TVM, toma como base as regras de mapeamento definidas na seção anterior, e é aplicado ao documento XML definido para o modelo. Esse documento está apresentado no anexo B.

As regras de mapeamento do modelo para o DB2 foram escritas utilizando a XSLT. A seguir será apresentada de forma resumida, a seqüencia seguida para a criação do arquivo de transformação.

1. Definição das variáveis e parâmetros globais.

2. Tipo `set` para dados não temporais: essa regra percorre todas as classes em busca de propriedades não temporais definidas do tipo `set`. Ao encontrar, cria uma classe com o mesmo nome da propriedade para armazenar todos os valores que podem ser associados à mesma.

3. Tipos estruturados: para cada classe definida no TVM, é criado um tipo estruturado, verificando-se o tipo de classe (abstrata, final ou normal) e herança.

4. Inclusão de relacionamentos para os tipos: são incluídos os relacionamentos definidos para as classes do TVM. Para isso, altera-se o tipo estruturado adicionando uma referência à classe associada. Cria-se a propriedade `property` na classe `InstantAttribute` para armazenar o nome das propriedades temporais.

5. Tabelas tipadas: definição das classes propriamente ditas e definição do identificador do objeto. Nesse momento, serão verificadas e definidas todas as restrições impostas pelo TVM para as propriedades.

6. Inclusão de relacionamentos para as classes: inclusão da opção adicional `scope` para que o operador *dereference* possa ser aplicado nas consultas.

7. Geração automática dos OID's: criação das seqüencias para a geração automática dos identificadores e definição das *triggers* de inserção para que seja atribuído o valor gerado ao OID.

8. Inserção dos dados do modelo nos metadados: inserção de todas as classes, propriedades, relacionamentos e métodos do Modelo Temporal de Versões nos metadados.

Os métodos definidos para o modelo não foram incluídos nesse arquivo XML. Isso se deve ao fato dos métodos não estarem definidos em SQL padrão. Inicialmente seria necessário escrevê-los em SQL padrão para em seguida, transformá-los. Porém, essa é uma atividade bastante complicada, uma vez que cada banco possui uma sintaxe específica para a definição de métodos, procedimentos e funções.

Nem todos os métodos foram implementados devido ao prazo para o término do trabalho, ficando os demais métodos como trabalhos futuros. Os métodos implementados estão apresentados no anexo D.

O resultado final obtido é um *script* especificado na sintaxe do DB2 com as principais características necessárias para a extensão de um banco de dados com característica OO para um banco temporal versionado. O usuário poderá intervir no resultado de forma a incluir características não implementadas e a otimizar o *script*. O *script* final gerado na sintaxe do DB2 é apresentado no anexo C.

Linguagem de Definição de Dados

O arquivo de transformação para a DDL apresenta de forma resumida como o usuário deverá definir suas classes e métodos no DB2 utilizando uma base de dados estendida para o Modelo Temporal de Versões. O resultado é um documento HTML com as principais diretrizes necessárias para que o usuário defina seu esquema.

A figura 6.3 apresenta a estrutura que deverá ser seguida para a definição das classes do usuário. Deve-se observar, no entanto, que as classes do modelo estarão transparentes para o usuário.

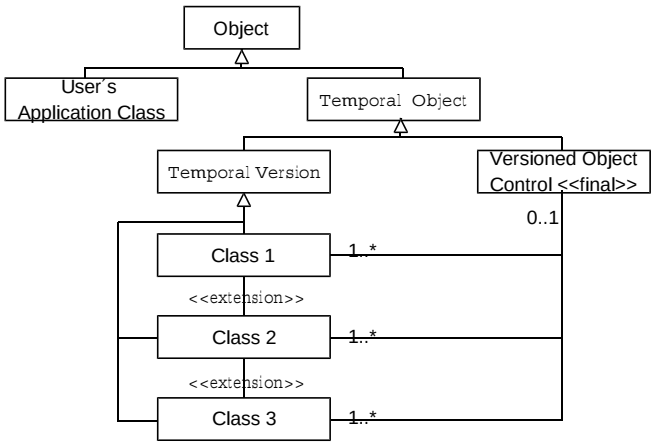


FIGURA 6.3 – Implementação das classes do usuário

Conforme figura acima, o usuário poderá definir classes temporais versionadas, que estarão ligadas diretamente à classe `TemporalVersion`. Essas classes terão um relacionamento com a classe `VersionedControlObject` que possui todos os dados de controle das versões. Classes não temporais e não versionadas, deverão estar diretamente ligadas à classe `Object`. As classes temporais versionadas só poderão se relacionar através de herança por extensão, enquanto que as classes não temporais e não versionadas poderão se relacionar através de herança por refinamento ou herança por extensão. Ambas poderão ter relacionamentos de associação e agregação.

O arquivo XSTL está ilustrado no anexo C. Em primeiro lugar é apresentado a sintaxe estendida para a definição de dados conforme especificada no TVM, e em seguida, a sintaxe que deverá ser utilizada para implementar as classes da aplicação no DB2. O documento está divido em dois grupos: classes temporais versionadas e classes não temporais e não versionadas.

Esse documento também poderá servir de base para a criar a documentação de outros bancos de dados, sendo apenas necessário a alteração da sintaxe para a criação das classes do usuário em um banco específico, como por exemplo Oracle.

6.2 Linguagem de Consulta do TVM

A especificação da TVQL em XML foi realizada com o intuito de tornar completa a formalização do TVM. Como resultado, obteve-se a XTVQL (*XML Temporal Version Query Language*), que poderá ser utilizada em conjunto com a XSLT de forma a transformar uma consulta, escrita nessa linguagem, em SQL ou OQL, por exemplo.

Imagine uma empresa que possua sua base de dados distribuída pelas suas filiais. Cada filial trabalha com um banco de dados diferente (Oracle, SQL Server, DB2 ...). Em algumas situações, a matriz requisitará a mesma informação para todas as filiais. A consulta a ser realizada é a mesma, mas, deve-se adaptá-la de acordo com a sintaxe específica de cada banco de dados, o que despenderia um esforço adicional para reescrever a consulta. Ao utilizar a linguagem de consulta especificada em XML, a matriz poderia enviar para cada filial a consulta desejada, sendo somente necessária a aplicação da XSLT correspondente à base de dados da filial.

A XTVQL, definida em [ROS 2001], incorpora todas as funções e propriedades definidas na TVQL [MOR 2001a], e permite que sejam definidas tanto consultas que lidam com tempo e versão quanto consultas que não necessitam recuperar dados específicos de tempo e versão. Uma das desvantagens dessa linguagem consiste no fato de não ser intuitiva, visto que o TVM trabalha com conceitos com os quais o usuário não está muito familiarizado. Uma forma de contornar essa limitação é através da construção de uma ferramenta de apoio a especificação de consultas em XTVQL.

Várias linguagens de consulta em XML já existem para serem aplicadas a documentos XML ou a base de dados. A TVQL teve como base as linguagens definidas em [ORA 2001, BUR 2000].

As principais características dessa linguagem serão descritas nas seções seguintes, e alguns exemplos serão apresentados com o intuito de facilitar o entendimento da XTVQL.

6.2.1 Estrutura

A linguagem XTVQL utiliza a mesma sintaxe da XML. Como conseqüência disso, uma consulta definida nessa linguagem deve ser, antes de mais nada, um documento bem formado. Além disso, todas as consultas devem seguir a gramática definida para a XTVQL. Essa gramática é apresentada no anexo A e será rapidamente explanada nessa seção.

A estrutura básica da XTVQL é bastante semelhante a da SQL. Isso se deve ao fato da TVQL ter sido definida com base nessa linguagem.

```
<XTVQL>
   <query>
      <return> ... <\return>
      <source> ... <\source>
```

```
      <condition> ... <\condition>
      <groupBy> ... <\groupBy>
      <orderBy> ... <\orderBy>
   </query>
</XTVQL>
```

O elemento raíz do documento é o `XTVQL`. Seu principal subelemento é o `query`, que é responsável pela definição da consulta propriamente dita. O elemento `return` seleciona as propriedades que deverão retornar como resultado da consulta baseado nas condições impostas em `condition`. A origem dos dados é informada em `source`. A especificação de condições para grupos e classificação do resultado é realizada através dos elementos `groupBy` e `orderBy` respectivamente.

As funções e propriedades específicas de tempo e versão, e os operadores (lógicos, relacionais, etc.) e funções de agregação, definidos em TVQL, foram divididos em dois grupos: funções temporais versionadas e funções normais. O primeiro grupo, representado pelo elemento `tvFunction`, poderá ser utilizado para compor condições que englobam dados com tempo e versão, e/ou para especificar a origem dos dados. Nesse último caso, especificar versões de um determinado objeto. O segundo grupo, `function`, poderá ser utilizado para selecionar uma propriedade a ser retornada como resultado de uma consulta e para compor condições para propriedades e grupos. Todos os dados referenciados por `function` não lidam com tempo e versão.

Além das consultas básicas, a XTVQL permite que sejam feitas outros tipos de consultas, tais como consultas aninhadas e operações sobre conjuntos, através dos operadores `union`, `intersect` e `difference`.

Um ponto importante que merece destaque é a utilização das cláusulas `EVER` e `PRESENT` definidas na TVQL na seção 4.1. De forma resumida, a cláusula `EVER` deverá ser especificada quando se desejar trabalhar com o histórico dos dados, quer seja para recuperação de dados quer seja para definição de condições. Uma vez definida no início da consulta, após a cláusula `SELECT`, considera-se que toda a consulta será realizada observando-se o histórico dos dados. Há a possibilidade de se trabalhar com o estado atual e com o histórico dos dados em uma mesma consulta. Para isso, deve-se combinar as duas cláusulas da mesma forma como é feita na TVQL.

Alguns exemplos serão apresentados na próxima seção com o intuito de facilitar a compreensão da `XTVQL`.

6.2.2 Exemplos

Voltando ao exemplo apresentado na figura 3.8, seção 3.1.3, as mesmas consultas descritas anteriormente em TVQL serão reescritas em XTVQL.

No primeiro exemplo, deseja-se recuperar informações referentes às configurações de computadores cuja capacidade do HD seja superior à 10Gb. A consulta equivalente em XTVQL será descrita da seguinte forma:

```
<XTVQL>
   <query>
      <return>
         <attribute name="processador" object="v" />
         <attribute name="HD" object="v" />
         <attribute name="memoria" object="v" />
         <attribute name="valor" object="v" />
```

```
      </return>
      <source>
         <object name="computador" />
         <tvFunction name="versions" alias="v">
            <param object="computador" />
         </tvFunction>
      </source>
      <condition>
         <function name="greater_than">
            <attribute name="HD" object="v" />
            <constant datatype="integer" value="10" />
         </function>
      </condition>
   </query>
</XTVQL>
```

As propriedades a serem retornadas estão definidas em `return` no elemento `attribute`, devendo-se obrigatoriamente indicar o objeto a qual a propriedade pertence. Os objetos (origem dos dados) são especificados em `source` através dos elementos `object`, para fontes de dados não temporais e não versionadas, e `tvFunction`, para especificar as versões dos objetos temporais versionados. Nesse último caso, o atributo `alias` é obrigatório e deverá ser re- ferenciado pelo atributo `object` do elemento `attribute`. Para as origens não temporais e não versionadas, o `alias` é opcional, sendo referenciado em `attribute` pelo nome do objeto (`object` **`name`**).

A condição imposta para a capacidade do `HD` é representada através de `function`. O nome da função indica que tipo de comparação deve ser realizada (`greater_than`). Os subelementos informam sob qual propriedade (`attribute`) será efetuada a comparação, e qual valor será considerado (`constant`).

O próximo exemplo, recupera os dados das versões de `notebook` que estão associadas às versões de configurações cuja memória é de 128Mb:

```
<XTVQL>
   <query>
      <return>
         <attribute name="bateria" object="vn" />
         <attribute name="dispositivo" object="vn" />
      </return>
      <source>
         <object name="computador" />
         <object name="notebook" />
         <tvFunction name="versions" alias="vc">
            <param object="computador" />
         </tvFunction>
         <tvFunction name="versions" alias="vn">
            <param object="notebook" />
         </tvFunction>
      </source>
      <condition>
         <function name="equal" connector="and">
            <attribute name="memoria" object="vc" />
            <constant datatype="integer" value="128" />
         </function>
         <tvFunction name="isDescendantOf">
            <param object="vn" />
            <param object="vc" />
         </tvFunction>
      </condition>
   </query>
</XTVQL>
```

A construção da consulta é realizada da mesma forma que a anterior. No entanto, observa-se a utilização de duas funções em `condition`, que estão li-

gadas entre si através do conectivo `and`, definido na primeira função pelo atributo `connector`. A segunda função, que é uma função temporal versionada, verifica as versões de `notebook` que são descendentes de `computador`. O primeiro subelemento indica o objeto a ser verificado (versões de `notebook`) e o segundo, o objeto no qual será feita a verificação (versões de `computador`).

A próxima consulta retorna as memórias válidas em ”25/07/2001”cujo estado é *working*:

```
<XTVQL>
   <query>
      <return>
         <attribute name="memoria" object="v" />
         <attribute name="valor" object="v" />
      </return>
      <source>
         <object name="computador" />
         <tvFunction name="versions" alias="v">
            <param object="computador" />
         </tvFunction>
      </source>
      <condition>
         <tvFunction name="isWorking" connector="and">
            <param object="v" />
         </tvFunction>
         <tvFunction name="into">
            <constant datatype="date" value="25/07/2001" />
            <tvFunction name="vInterval">
               <attribute name="memoria" object="v" />
            </tvFunction>
         </tvFunction>
      </condition>
   </query>
</XTVQL>
```

Nessa consulta, duas funções temporais versionadas foram definidas no elemento `condition`. A primeira, verifica quais versões estão com o estado em trabalho. A segunda, verifica se o instante de validade informada na consulta está incluso no intervalo de validade da propriedade `memoria`. Nesse caso, uma nova função temporal versionada para recuperar o intervalo de validade de `memoria` é requerida.

Para recuperar o histórico dos valores da versão `c4` de `computador` com seus respectivos rótulos de validade, a seguinte consulta seria especificada:

```
<XTVQL>
   <query>
      <return state="ever">
         <attribute name="valor" object="v" />
         <tvFunction name="vInterval">
            <attribute name="valor" object="v" />
         </tvFunction>
         <tvFunction name="tInterval">
            <attribute name="valor" object="v" />
         </tvFunction>
      </return>
      <source>
         <object name="computador" />
         <tvFunction name="versions" alias="v">
            <param object="computador" />
         </tvFunction>
      </source>
      <condition>
         <function name="equal">
            <tvFunction name="nickname">
               <param object="v"/>
            </tvFunction>
```

```
            <constant datatype="character" value="c4" />
         </function>
      </condition>
   </query>
</XTVQL>
```

O que deve ser observado nessa consulta é a presença da palavra `ever` em `return`, indicando que o histórico dos dados deverá ser retornado. Como o estado de retorno foi aplicado apenas no início da consulta, toda a consulta (propriedades e condições) será realizada levando em consideração esse estado de retorno (histórico).

Em nenhuma das consultas anteriores, o estado de retorno foi mencionado. Quando isso acontece, fica subentendido que o retorno será referente ao estado corrente dos dados.

As funções temporais versionadas `vInterval` e `tInterval` foram especificadas para recuperar o intervalo de validade e de transação inicial e final da propriedade `valor`. Quando se define uma consulta para recuperar o histórico dos dados não está implícito que os intervalos deverão ser retornados juntamente com os valores da propriedade, sendo necessário portanto, defini-los explicitamente.

Outra nova função temporal versionada definida em `condition` é a `nickname`. Essa função é utilizada apenas para retornar o apelido do objeto.

Em algumas situações, é necessário combinar condições temporais com estados atuais do objeto. Por exemplo, deseja-se o histórico dos valores dos notebooks com bateria de duração de 2h e cuja memória seja de 128Mb. A consulta é construída da seguinte forma:

```
<XTVQL>
   <query>
      <return state="ever">
         <attribute name="valor" object="vc" />
      </return>
      <source>
         <object name="computador" />
         <object name="notebook" />
         <tvFunction name="versions" alias="vc">
            <param object="computador" />
         </tvFunction>
         <tvFunction name="versions" alias="vn">
            <param object="notebook" />
         </tvFunction>
      </source>
      <condition>
         <function name="equal" connector="and">
            <attribute name="memoria" object="vc" />
            <constant datatype="integer" value="128" />
         </function>
         <function name="equal" connector="and" state="present">
            <attribute name="duracao" object="vn" />
            <constant datatype="integer" value="2" />
         </function>
         <tvFunction name="isDescendantOf" state="present">
            <param object="vn" />
            <param object="vc" />
         </tvFunction>
      </condition>
   </query>
</XTVQL>
```

Da mesma forma que a consulta anterior, a palavra `ever` foi utilizada para recuperar o histórico dos dados. No entanto, não será aplicada a toda a consulta,

uma vez que o estado `present` foi definido para a condição de verificação da propriedade `duracao` e da função `isDescendantOf`. Como resultado, tem-se que o histórico dos dados será aplicado a todas propriedades e condições exceto àqueles que explicitamente forem definidos com o estado `present`.

Um outro exemplo combinando condições temporais com estados atuais dos objetos é a recuperação dos notebooks que em algum momento da sua vida tiveram memória de 128Mb:

```
<XTVQL>
   <query>
      <return state="ever">
         <attribute name="bateria" object="vn" />
         <attribute name="dispositivo" object="vn" />
      </return>
      <source>
         <object name="computador" />
         <object name="notebook" />
         <tvFunction name="versions" alias="vc">
            <param object="computador" />
         </tvFunction>
         <tvFunction name="versions" alias="vn">
            <param object="notebook" />
         </tvFunction>
      </source>
      <condition>
         <function name="equal" connector="and" state="ever">
            <attribute name="memoria" object="vc" />
            <constant datatype="integer" value="128" />
         </function>
         <tvFunction name="isDescendantOf">
            <param object="vn" />
            <param object="vc" />
         </tvFunction>
      </condition>
   </query>
</XTVQL>
```

Nesse último exemplo, como a cláusula `ever` não foi aplicada a toda a consulta (no elemento `query`), será considerado o estado corrente dos dados, exceto nos elementos onde a cláusula `ever` estiver explicitamente definida, em `return` e em `function` aplicada à `memoria`.

Outras consultas mais complexas poderão ser definidas em XTVQL, porém, não serão vistos todos os casos pelo fato de esse não ser o foco principal do trabalho.

6.3 Conclusão

XML tem se tornado o formato universal para a representação e troca de informações pela internet. Em diversas aplicações, dados XML são gerados de repositórios legados (bancos relacionais, banco orientado a objetos, etc.), ou exportados para uma determinada aplicação [FAN 2001]. Como um padrão amplamente aceito, XML também está sendo utilizado como uma ferramenta para formalização de conceitos, esquemas, modelos e dados.

A especificação do Modelo Temporal de Versões em XML usou a funcionalidade de formalização da linguagem. Dessa forma, o modelo torna-se independe da plataforma na qual será aplicado e da sintaxe utilizada pelos bancos de dados para definição e manipulação dos dados. Além do modelo propriamente dito,

foram especificadas também a linguagem de definição de dados estendida para o modelo e a linguagem de consulta dos dados. Ambas as linguagens foram descritas em XML para tornar completa a formalização do TVM.

Aplicando-se a linguagem de transformação, XSLT, às especificações em XML, obteu-se um *script* em SQL com as características do TVM e uma documentação para auxiliar o usuário na construção dos seus esquemas.

O banco de dados selecionado inicialmente para aplicar esse *script* foi o DB2, por ser o mais próximo ao padrão `SQL-92` no suporte aos tipos de dados temporais. No entanto, outros bancos poderão fazer uso da especificação proposta nessa trabalho, bastando, apenas, que gerem arquivos de transformação próprios para o banco de dados selecionado.

O DB2 possui pequenas restrições quanto a tipos de dados. Não implementa o tipo lógico e os tipos de coleção. É importante ressaltar que dentre as soluções encontradas para mapear o modelo para o DB2, os métodos implementados como *stored procedures* e a cláusula `final`, definida nos tipos estruturados, foram testados, porém, sem sucesso devido, possivelmente, a problemas de configuração.

Apesar dos problemas encontrados e de os métodos não estarem todos implementados, as principais funcionalidades do modelo estão mapeadas possibilitando a sua utilização.

7 Conclusão

O Modelo Temporal de Versões permite que aplicações que trabalhem com tempo e versão sejam modeladas de forma mais natural. Esse modelo está baseado no paradigma de orientação a objetos. É um modelo bitemporal, ou seja, associa os tempos de transação e de validade aos dados. O tempo está relacionado aos objetos, propriedades e relacionamentos.

Para que aplicações possam utilizar esse modelo, é necessário que seja feito um mapeamento do TVM para um determinado banco de dados. Isso pode ser feito diretamente pelo usuário ou através de uma aplicação.

Esse trabalho propõe realizar esse mapeamento utilizando o padrão XML. Criou-se uma estrutura em XML de forma a representar as classes do modelo, bem como suas propriedades e métodos. Além de especificar as características básicas do modelo, também foram especificados os metadados e as linguagens de manipulação de dados e de consulta.

A especificação do Modelo Temporal de Versões em XML, traz consigo algumas vantagens, dentre elas, pode-se destacar:

- independência de plataforma na qual será aplicada o modelo;
- independência de sintaxe de definição de dados utilizada pelos bancos de dados;
- não se detém a restrições de determinados bancos de dados;
- formalização do modelo através de um formato padrão.

É importante destacar que a linguagem de consulta do modelo foi definida em XML apenas para tornar completa a especificação do TVM.

Como XML se tornou rapidamente um formato padrão para aplicações na internet, várias ferramentas foram desenvolvidas com o intuito de manipular esses dados. Aqui, destaca-se a XSLT, que é uma linguagem de transformação de documentos XML para outros formatos.

A utilização da XSLT em conjunto com as especificações definidas para o modelo resulta em um *script* SQL com a linguagem de definição de dados utilizada por um determinado banco de dados.

Esse trabalho inicialmente criou regras para gerar o *script* para o DB2. Esse banco foi o escolhido por ser o mais próximo ao padrão SQL-92 no suporte aos dados temporais. As regras criadas para a transformação da especificação do Modelo Temporal de Versões para DB2 foram geradas observando as restrições impostas por esse banco de dados. A versão do banco de dados utilizada nesse trabalho foi a 7.2. Essa versão implementa características objeto-relacionais.

As principais restrições vistas foram a ausência do tipo lógico e de tipos de coleção. Além disso, as *stored procedures* especificadas para implementar alguns métodos das classes do modelo não puderam ser testadas por possíveis problemas de configuração. Vários obstáculos foram encontrados no momento da definição de regras para o mapeamento dos dados do TVM para o DB2. A cláusula FINAL, utilizada para a criação de tipos estruturados, foi testada sem

sucesso. Essa cláusula define classes que não podem ter subclasses associadas a ela.

A principal contribuição desse trabalho consiste na especificação de uma estrutura em XML para representar o modelo, na sua formalização e na especificação de regras para estender o DB2 para uma base temporal versionada.

Como trabalhos futuros, sugere-se o seguinte:

- incluir na especificação em XML do modelo, os métodos das classes em SQL padrão;
- definir regras de transformação para outros bancos de dados, como por exemplo Oracle e SQL Server;
- especificar os métodos que não foram implementados para o DB2;
- incluir na XSLT, regras para a criação automática dos métodos;
- implementar a aplicação proposta nesse trabalho.

Apesar de não estar completa a especificação do TVM em XML, uma vez que os métodos não foram incorporados à estrutura, as principais características necessárias para o funcionamento do modelo foram implementadas.

Anexo A Gramática da XTVQL

```
<!ENTITY % trueFalse "(True|False) 'False'">
<!ENTITY % visibility "(public|private|protected) 'public'">
<!ELEMENT metaSchema (classes, operations?)>
<!ELEMENT classes (class+)>
<!ELEMENT class (name, comment?, superClass?, attributes?,
                 operations?, relationships?)>
<!ATTLIST class type (abstract | final | none) "none"
               visibility %visibility;
               root %trueFalse;
               metadata %trueFalse;
               multipleInheritance %trueFalse;>
<!ELEMENT attributes (attribute+)>
<!ELEMENT attribute (name, datatype, length?, decimal?,
                     defaultValue?, checkConstraint?)>
<!ATTLIST attribute temporal %trueFalse;
                    nulls %trueFalse;
                    visibility %visibility;
                    static %trueFalse;>
<!ELEMENT operations (operation+)>
<!ELEMENT operation (name, comment?, conditions?, parameters?,
                     body)>
<!ATTLIST operation type (procedure | function | method) "method"
                    override %trueFalse;
                    final %trueFalse;
                    visibility %visibility;>
<!ELEMENT conditions (condition+)>
<!ELEMENT parameters (in*, out*, return?)>
<!ELEMENT in (name, datatype, length?, defaultValue?)>
<!ELEMENT out (name, datatype, length?)>
<!ELEMENT return (datatype, length?)>
<!ELEMENT relationships (relationship+)>
<!ELEMENT relationship (name?, cardinality, relatedClassName,
                        passingBy?, inverse?)>
<!ATTLIST relationship type (association |
                             aggregation) "association"
                       temporal %trueFalse;>
<!ELEMENT inverse (#PCDATA)>
<!ELEMENT comment (#PCDATA)>
<!ELEMENT superClass (#PCDATA)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT datatype (#PCDATA)>
<!ELEMENT length (#PCDATA)>
<!ELEMENT decimal (#PCDATA)>
<!ELEMENT defaultValue (#PCDATA)>
<!ELEMENT checkConstraint (#PCDATA)>
<!ELEMENT body (#PCDATA)>
<!ATTLIST body language (SQL | java | C)"SQL">
<!ELEMENT condition (#PCDATA)>
<!ELEMENT cardinality (#PCDATA)>
<!ELEMENT passingBy (#PCDATA)>
<!ELEMENT relatedClassName (#PCDATA)>
```

Anexo B Gramática e Documento do TVM

Gramática do TVM

```
<!ENTITY % trueFalse "(True|False) 'False'">
<!ENTITY % visibility "(public|private|protected) 'public'">
<!ELEMENT metaSchema (classes, operations?)>
<!ELEMENT classes (class+)>
<!ELEMENT class (name, comment?, superClass?, attributes?, operations?,
                 relationships?)>
<!ATTLIST class
        type (abstract | final | none) "none"
        visibility %visibility;
        root %trueFalse;
        metadata %trueFalse;
        multipleInheritance %trueFalse;
>
<!ELEMENT attributes (attribute+)>
<!ELEMENT attribute (name, datatype, length?, decimal?, defaultValue?,
                     checkConstraint?)>
<!ATTLIST attribute
        temporal %trueFalse;
        nulls %trueFalse;
        visibility %visibility;
        static %trueFalse;>
<!ELEMENT operations (operation+)>
<!ELEMENT operation (name, comment?, conditions?, parameters?, body)>
<!ATTLIST operation
        type (procedure | function | method) "method"
        override %trueFalse;
        final %trueFalse;
        visibility %visibility;
>
<!ELEMENT conditions (condition+)>
<!ELEMENT parameters (in*, out*, return?)>
<!ELEMENT in (name, datatype, length?, defaultValue?)>
<!ELEMENT out (name, datatype, length?)>
<!ELEMENT return (datatype, length?)>
<!ELEMENT relationships (relationship+)>
<!ELEMENT relationship (name?, cardinality, relatedClassName, passingBy?,
                        inverse?)>
<!ATTLIST relationship
        type (association | aggregation) "association"
        temporal %trueFalse;
>
<!ELEMENT inverse (#PCDATA)>
<!ELEMENT comment (#PCDATA)>
<!ELEMENT superClass (#PCDATA)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT datatype (#PCDATA)>
<!ELEMENT length (#PCDATA)>
<!ELEMENT decimal (#PCDATA)>
<!ELEMENT defaultValue (#PCDATA)>
<!ELEMENT checkConstraint (#PCDATA)>
<!ELEMENT body (#PCDATA)>
<!ATTLIST body
        language (SQL | java | C) "SQL"
>
<!ELEMENT condition (#PCDATA)>
<!ELEMENT cardinality (#PCDATA)>
<!ELEMENT passingBy (#PCDATA)>
<!ELEMENT relatedClassName (#PCDATA)>
```

Documento XML

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<!DOCTYPE metaSchema SYSTEM "d:\ufrgs\dissertacao\esquema\esquema7.dtd">
<?xml-stylesheet type="text/xsl"
                 href="D:\UFRGS\Dissertacao\Esquema\docExemplo4_3.xsl"?>
<metaSchema>
   <classes>
      <!-- METADADOS -->
      <class visibility="public" metadata="True">
         <name> Schema </name>
         <comment> armazena os dados do esquema </comment>
         <attributes>
            <attribute temporal="False" nulls="False">
               <name> id </name>
               <datatype> integer </datatype>
            </attribute>
            <attribute temporal="False" nulls="False">
               <name> name </name>
               <datatype> string </datatype>
               <length> 30 </length>
            </attribute>
         </attributes>
         <relationships>
            <relationship type="aggregation">
               <cardinality> 0..n </cardinality>
               <relatedClassName> Entity </relatedClassName>
            </relationship>
            <relationship type="aggregation">
               <cardinality> 1..n </cardinality>
               <relatedClassName> Class </relatedClassName>
            </relationship>
         </relationships>
      </class>

      <class visibility="public" metadata="True">
         <name> Entity </name>
         <comment> armazena todas as entidades do esquema </comment>
         <attributes>
            <attribute temporal="False" nulls="False">
               <name> id </name>
               <datatype> integer </datatype>
            </attribute>
            <attribute temporal="False" nulls="False">
               <name> name </name>
               <datatype> string </datatype>
               <length> 30 </length>
            </attribute>
         </attributes>
      </class>

      <class visibility="public" metadata="True">
         <name> Class </name>
         <comment> armazena as classes do esquema </comment>
         <attributes>
            <attribute temporal="False" nulls="False">
               <name> id </name>
               <datatype> integer </datatype>
            </attribute>
            <attribute temporal="False" nulls="False">
               <name> name </name>
               <datatype> string </datatype>
               <length> 30 </length>
            </attribute>
            <attribute temporal="False" nulls="False">
               <name> tempVers </name>
               <datatype> boolean </datatype>
            </attribute>
            <attribute temporal="False" nulls="True">
               <name> specializ </name>
               <datatype> enum </datatype>
               <checkConstraint> IN ('none', 'abstract', 'final')
               </checkConstraint>
            </attribute>
            <attribute temporal="False" nulls="True">
               <name> visibility </name>
```

```
         <datatype> enum </datatype>
         <checkConstraint> IN ('public', 'protected', 'private')
         </checkConstraint>
      </attribute>
      <attribute temporal="False" nulls="True">
         <name> root </name>
         <datatype> boolean </datatype>
      </attribute>
   </attributes>
   <relationships>
      <relationship type="aggregation">
         <cardinality> 0..n </cardinality>
         <relatedClassName> Relationship </relatedClassName>
      </relationship>
      <relationship type="aggregation">
         <cardinality> 0..n </cardinality>
         <relatedClassName> Attribute </relatedClassName>
      </relationship>
      <relationship type="aggregation">
         <cardinality> 0..n </cardinality>
         <relatedClassName> Operation </relatedClassName>
      </relationship>
   </relationships>
</class>

<class visibility="public" metadata="True">
   <name> Relationship </name>
   <comment> armazena os relacionamentos entre as classes da
             aplicacao </comment>
   <attributes>
      <attribute temporal="False" nulls="True">
         <name> name </name>
         <datatype> string </datatype>
         <length> 30 </length>
      </attribute>
      <attribute temporal="False" nulls="False">
         <name> relatedClass </name>
         <datatype> string </datatype>
         <length> 30 </length>
      </attribute>
   </attributes>
</class>

<class visibility="public" metadata="True">
   <name> Association </name>
   <comment> armazena os relacionamentos de associacao </comment>
   <superClass> Relationship </superClass>
   <attributes>
      <attribute temporal="False" nulls="True">
         <name> inverse </name>
         <datatype> string </datatype>
         <length> 30 </length>
      </attribute>
      <attribute temporal="False" nulls="False">
         <name> temp </name>
         <datatype> boolean </datatype>
      </attribute>
      <attribute temporal="False" nulls="False">
         <name> cardinality </name>
         <datatype> enum </datatype>
         <checkConstraint> IN ('0..1', '0..n', '1..1', '1..n', 'n..m')
         </checkConstraint>
      </attribute>
   </attributes>
</class>

<class visibility="public" metadata="True">
   <name> Aggregation </name>
   <comment> armazena os relacionamentos de agregacao </comment>
   <superClass> Relationship </superClass>
   <attributes>
      <attribute temporal="False" nulls="False">
         <name> temp </name>
```

```
         <datatype> boolean</datatype>
      </attribute>
      <attribute temporal="False" nulls="True">
         <name> type </name>
         <datatype> enum </datatype>
         <checkConstraint> IN ('byValue', 'byReference') </checkConstraint>
      </attribute>
      <attribute temporal="False" nulls="False">
         <name> cardinality </name>
         <datatype> enum </datatype>
         <checkConstraint> IN ('1..1', '1..n') </checkConstraint>
      </attribute>
   </attributes>
</class>

<class visibility="public" metadata="True">
   <name> Inheritance </name>
   <comment> armazena os relacionamentos de heranca por
             refinamento </comment>
   <superClass> Relationship </superClass>
   <attributes>
      <attribute temporal="False" nulls="False">
         <name> single </name>
         <datatype> boolean </datatype>
      </attribute>
   </attributes>
</class>

<class visibility="public" metadata="True">
   <name> Extension </name>
   <comment> armazena os relacionamentos de heranca por
             extensao </comment>
   <superClass> Relationship </superClass>
   <attributes>
      <attribute temporal="True" nulls="False">
         <name> corresp </name>
         <datatype> enum </datatype>
         <checkConstraint> IN ('1:1', '1:n', 'n:1', 'n:m')
         </checkConstraint>
      </attribute>
   </attributes>
</class>

<class visibility="public" metadata="True">
   <name> Attribute </name>
   <comment> armazena os dados dos atributos </comment>
   <attributes>
      <attribute temporal="False" nulls="False">
         <name> name </name>
         <datatype> string </datatype>
         <length> 30 </length>
      </attribute>
      <attribute temporal="False" nulls="False">
         <name> temp </name>
         <datatype> boolean </datatype>
      </attribute>
      <attribute temporal="False" nulls="True">
         <name> visibility </name>
         <datatype> enum </datatype>
         <checkConstraint> IN ('public', 'protected', 'private')
         </checkConstraint>
      </attribute>
      <attribute temporal="False" nulls="True">
         <name> type </name>
         <datatype> string </datatype>
         <length> 10 </length>
      </attribute>
      <attribute temporal="False" nulls="True">
         <name> default </name>
         <datatype> string </datatype>
         <length> 5 </length>
      </attribute>
      <attribute temporal="False" nulls="True">
```

```
         <name> static </name>
         <datatype> boolean </datatype>
      </attribute>
   </attributes>
</class>

<class visibility="public" metadata="True">
   <name> Operation </name>
   <comment> armazena as operacoes da aplicacao </comment>
   <attributes>
      <attribute temporal="False" nulls="False">
         <name> name </name>
         <datatype> string </datatype>
         <length> 30 </length>
      </attribute>
      <attribute temporal="False" nulls="True">
         <name> specializ </name>
         <datatype> enum </datatype>
         <checkConstraint> IN ('none', 'abstract', 'final')
         </checkConstraint>
      </attribute>
      <attribute temporal="False" nulls="True">
         <name> visibility </name>
         <datatype> enum </datatype>
         <checkConstraint> IN ('public', 'protected', 'private')
         </checkConstraint>
      </attribute>
      <attribute temporal="False" nulls="True">
         <name> static </name>
         <datatype> boolean </datatype>
      </attribute>
   </attributes>
   <relationships>
      <relationship type="association">
         <name> in </name>
         <cardinality> 0..n </cardinality>
         <relatedClassName> Parameter </relatedClassName>
      </relationship>
      <relationship type="association">
         <name> out </name>
         <cardinality> 0..n </cardinality>
         <relatedClassName> Parameter </relatedClassName>
      </relationship>
   </relationships>
</class>

<class visibility="public" metadata="True">
   <name> Parameter </name>
   <comment> armazena os parametros das operacoes </comment>
   <attributes>
      <attribute temporal="False" nulls="False">
         <name> name </name>
         <datatype> string </datatype>
         <length> 30 </length>
      </attribute>
      <attribute temporal="False" nulls="True">
         <name> default </name>
         <datatype> string </datatype>
         <length> 5 </length>
      </attribute>
      <attribute temporal="False" nulls="False">
         <name> type </name>
         <datatype> string </datatype>
         <length> 10 </length>
      </attribute>
   </attributes>
</class>

<!-- OIDt, NAME e TIPOS TEMPORAIS -->
<class type="abstract" visibility="public">
   <name> OIDt </name>
   <comment> tipo estruturado que define a estrutura do
             OID (identificador da entidade, da classe,
```

```
            numero da versao) </comment>
   <attributes>
      <attribute temporal="False" nulls="False">
         <name> value </name>
         <datatype> string </datatype>
         <length> 15 </length>
      </attribute>
   </attributes>
   <operations>
      <operation type="method" override="False">
         <name> OIDt </name>
         <comment> construtor </comment>
         <parameters>
            <in>
               <name> E </name>
               <datatype> integer </datatype>
            </in>
            <in>
               <name> C </name>
               <datatype> integer </datatype>
            </in>
            <in>
               <name> V </name>
               <datatype> integer </datatype>
            </in>
         </parameters>
         <body language="SQL"/>
      </operation>
      <operation type="method" override="False">
         <name> getClassNr </name>
         <comment> retorna o identificador da classe </comment>
         <parameters>
            <return>
               <datatype> integer </datatype>
            </return>
         </parameters>
         <body language="SQL"/>
      </operation>
      <operation type="method" override="False">
         <name> getEntityNr </name>
         <comment> retorna o identificador da entidade </comment>
         <parameters>
            <return>
               <datatype> integer </datatype>
            </return>
         </parameters>
         <body language="SQL"/>
      </operation>
      <operation type="method" override="False">
         <name> getOID </name>
         <comment> retorna o identificador gerado do objeto
                   (EntityNr, ClassNr, VersionNr) </comment>
         <parameters>
            <return>
               <datatype> OIDt </datatype>
            </return>
         </parameters>
         <body language="SQL"/>
      </operation>
      <operation type="method" override="False">
         <name> getVersionNr </name>
         <comment> retorna o numero da versao </comment>
         <parameters>
            <return>
               <datatype> integer </datatype>
            </return>
         </parameters>
         <body language="SQL"/>
      </operation>
   </operations>
</class>

<class visibility="public">
```

```
   <name> Name </name>
   <comment> armazena os nomes (apelidos) atribuidos aos
             objetos </comment>
   <attributes>
      <attribute temporal="False" nulls="False">
         <name> value </name>
         <datatype> string </datatype>
         <length> 20 </length>
      </attribute>
   </attributes>
   <operations>
      <operation type="method" override="False">
         <name> Name </name>
         <comment> construtor </comment>
         <parameters>
            <in>
               <name> nam </name>
               <datatype> string </datatype>
               <length> 20 </length>
            </in>
         </parameters>
         <body language="SQL"/>
      </operation>
      <operation type="method" override="False">
         <name> getClassName </name>
         <comment> retorna o nome da classe </comment>
         <parameters>
            <return>
               <datatype> string </datatype>
               <length> 30 </length>
            </return>
         </parameters>
         <body language="SQL"/>
      </operation>
      <operation type="method" override="False">
         <name> getName </name>
         <comment> retorna o apelido atribu_do ao objeto </comment>
         <parameters>
            <return>
               <datatype> string </datatype>
               <length> 30 </length>
            </return>
         </parameters>
         <body language="SQL"/>
      </operation>
   </operations>
   <relationships>
      <relationship type="association">
         <name> nickNameOf </name>
         <cardinality> 1:1 </cardinality>
         <relatedClassName> Object </relatedClassName>
         <inverse> nickname </inverse>
      </relationship>
   </relationships>
</class>

<class type="abstract" visibility="public">
   <name> TemporalLabel </name>
   <comment> tipo estruturado que define os rotulos
             temporais </comment>
   <attributes>
      <attribute temporal="False" nulls="False">
         <name> tTimei </name>
         <datatype> instant </datatype>
      </attribute>
      <attribute temporal="False" nulls="False">
         <name> tTimef </name>
         <datatype> instant </datatype>
      </attribute>
      <attribute temporal="False" nulls="False">
         <name> vTimei </name>
         <datatype> instant </datatype>
      </attribute>
```

```
   <attribute temporal="False" nulls="True">
      <name> vTimef </name>
      <datatype> instant </datatype>
   </attribute>
</attributes>
<operations>
   <operation type="method" override="False">
      <name> TemporalLabel </name>
      <comment> construtor </comment>
      <parameters>
         <in>
            <name> iValidTime </name>
            <datatype> instant </datatype>
         </in>
         <in>
            <name> fValidTime </name>
            <datatype> instant </datatype>
         </in>
         <in>
            <name> iTransTime </name>
            <datatype> instant </datatype>
         </in>
         <in>
            <name> fTransTime </name>
            <datatype> instant </datatype>
         </in>
      </parameters>
      <body language="SQL"/>
   </operation>
   <operation type="method" override="False">
      <name> getTTimei </name>
      <comment> retorna o instante de transa__o inicial </comment>
      <parameters>
         <return>
            <datatype> instant </datatype>
         </return>
      </parameters>
      <body language="SQL"/>
   </operation>
   <operation type="method" override="False">
      <name> getTTimef </name>
      <comment> retorna o instante de transacao final </comment>
      <parameters>
         <return>
            <datatype> instant </datatype>
         </return>
      </parameters>
      <body language="SQL"/>
   </operation>
   <operation type="method" override="False">
      <name> getVTimei </name>
      <comment> retorna o instante de validade inicial  </comment>
      <parameters>
         <return>
            <datatype> instant </datatype>
         </return>
      </parameters>
      <body language="SQL"/>
   </operation>
   <operation type="method" override="False">
      <name> getVTimef </name>
      <comment> retorna o instante de validade final </comment>
      <parameters>
         <return>
            <datatype> instant </datatype>
         </return>
      </parameters>
      <body language="SQL"/>
   </operation>
   <operation type="method" override="False">
      <name> setTTimef </name>
      <comment> atualiza o instante de transa__o final </comment>
      <parameters>
```

```
            <in>
               <name> i </name>
               <datatype> instant </datatype>
            </in>
         </parameters>
         <body language="SQL"/>
      </operation>
      <operation type="method" override="False">
         <name> setVTimef </name>
         <comment> atualiza o instante de validade final </comment>
         <parameters>
            <in>
               <name> i </name>
               <datatype> instant </datatype>
            </in>
         </parameters>
         <body language="SQL"/>
      </operation>
   </operations>
</class>

<class visibility="public">
   <name> InstantAttribute </name>
   <comment> armazena 1o valor do atributo com seus rotulos
             temporais dos atributos </comment>
   <attributes>
      <attribute temporal="False" nulls="False">
         <name> value </name>
         <datatype> Object </datatype>
      </attribute>
      <attribute temporal="False" nulls="False">
         <name> tempLabel </name>
         <datatype> TemporalLabel </datatype>
      </attribute>
   </attributes>
   <operations>
      <operation type="method" override="False">
         <name> InstantAttribute </name>
         <comment> construtor </comment>
         <parameters>
            <in>
               <name> val </name>
               <datatype> Object </datatype>
            </in>
         </parameters>
         <body language="SQL"/>
      </operation>
      <operation type="method" override="False">
         <name> InstantAttribute </name>
         <comment> construtor </comment>
         <parameters>
            <in>
               <name> val </name>
               <datatype> Object </datatype>
            </in>
            <in>
               <name> iValidTime </name>
               <datatype> instant </datatype>
            </in>
            <in>
               <name> fValidTime </name>
               <datatype> instant </datatype>
            </in>
         </parameters>
         <body language="SQL"/>
      </operation>
      <operation type="method" override="False">
         <name> getValue </name>
         <comment> retorna o valor corrente </comment>
         <parameters>
            <return>
               <datatype> Object </datatype>
            </return>
```

```
         </parameters>
         <body language="SQL"/>
      </operation>
      <operation type="method" override="False">
         <name> getTempLabel </name>
         <comment> retorna os rotulos temporais do valor corrente
         </comment>
         <parameters>
            <return>
               <datatype> TemporalLabel </datatype>
            </return>
         </parameters>
         <body language="SQL"/>
      </operation>
      <operation type="method" override="False">
         <name> getTempLabelOf </name>
         <comment> retorna todos os rotulos temporais de um determinado
                   objeto </comment>
         <parameters>
            <in>
               <name> val </name>
               <datatype> Object </datatype>
            </in>
            <return>
               <datatype> set(TemporalLabel) </datatype>
            </return>
         </parameters>
         <body language="SQL"/>
      </operation>
      <operation type="method" override="False">
         <name> setFTransactionTime </name>
         <comment> atualiza o instante de transacao final do atributo
         </comment>
         <parameters>
            <in>
               <name> i </name>
               <datatype> instant </datatype>
            </in>
         </parameters>
         <body language="SQL"/>
      </operation>
      <operation type="method" override="False">
         <name> setFValidTime </name>
         <comment> atualiza o instante de validade final do atributo
         </comment>
         <parameters>
            <in>
               <name> i </name>
               <datatype> instant </datatype>
            </in>
         </parameters>
         <body language="SQL"/>
      </operation>
   </operations>
</class>

<class visibility="public">
   <name> InstantRelationship </name>
   <comment> armazena os identificadores dos objetos que fazem parte do
             relacionamento e seus rotulos temporais </comment>
   <attributes>
      <attribute temporal="False" nulls="False">
         <name> obj1 </name>
         <datatype> OIDt </datatype>
      </attribute>
      <attribute temporal="False" nulls="False">
         <name> obj2 </name>
         <datatype> OIDt </datatype>
      </attribute>
      <attribute temporal="False" nulls="False">
         <name> tempLabel </name>
         <datatype> TemporalLabel </datatype>
      </attribute>
```

```
</attributes>
<operations>
   <operation type="method" override="False">
      <name> InstantRelationship </name>
      <comment> construtor </comment>
      <parameters>
         <in>
            <name> o1 </name>
            <datatype> OIDt </datatype>
         </in>
         <in>
            <name> o2 </name>
            <datatype> OIDt </datatype>
         </in>
      </parameters>
      <body language="SQL"/>
   </operation>
   <operation type="method" override="False">
      <name> InstantRelationship </name>
      <comment> construtor </comment>
      <parameters>
         <in>
            <name> o1 </name>
            <datatype> OIDt </datatype>
         </in>
         <in>
            <name> o2 </name>
            <datatype> OIDt </datatype>
         </in>
         <in>
            <name> iValidTime </name>
            <datatype> instant </datatype>
         </in>
         <in>
            <name> fValidTime </name>
            <datatype> instant </datatype>
         </in>
      </parameters>
      <body language="SQL"/>
   </operation>
   <operation type="method" override="False">
      <name> getObj1 </name>
      <comment> retorna o primeiro objeto que faz parte do
                relacionamento </comment>
      <parameters>
         <return>
            <datatype> OIDt </datatype>
         </return>
      </parameters>
      <body language="SQL"/>
   </operation>
   <operation type="method" override="False">
      <name> getObj2 </name>
      <comment> retorna o segundo objeto que faz parte do
                relacionamento </comment>
      <parameters>
         <return>
            <datatype> OIDt </datatype>
         </return>
      </parameters>
      <body language="SQL"/>
   </operation>
   <operation type="method" override="False">
      <name> getTempLabel </name>
      <comment> retorna os r_tulos temporais do relacionamento
      </comment>
      <parameters>
         <return>
            <datatype> TemporalLabel </datatype>
         </return>
      </parameters>
      <body language="SQL"/>
   </operation>
```

```
      <operation type="method" override="False">
         <name> getTempLabelOf </name>
         <comment> retorna todos os r_tulos temporais de um determinado
                   relacionamento </comment>
         <parameters>
            <in>
               <name> o1 </name>
               <datatype> OIDt </datatype>
            </in>
            <in>
               <name> o2 </name>
               <datatype> OIDt </datatype>
            </in>
            <return>
               <datatype> set(TemporalLabel) </datatype>
            </return>
         </parameters>
         <body language="SQL"/>
      </operation>
      <operation type="method" override="False">
         <name> setFTransactionTime </name>
         <comment> atualiza o instante de transação final do relacionamento
         </comment>
         <parameters>
            <in>
               <name> i </name>
               <datatype> instant </datatype>
            </in>
         </parameters>
         <body language="SQL"/>
      </operation>
      <operation type="method" override="False">
         <name> setFValidTime </name>
         <comment> atualiza o instante de validade final do relacionamento
         </comment>
         <parameters>
            <in>
               <name> i </name>
               <datatype> instant </datatype>
            </in>
         </parameters>
         <body language="SQL"/>
      </operation>
   </operations>
</class>

<class visibility="public">
   <name> TemporalAttribute </name>
   <comment> armazena o conjunto de valores assumidos pelos atributos
             temporais </comment>
   <attributes>
      <attribute temporal="False" nulls="False">
         <name> type </name>
         <datatype> string </datatype>
         <length> 10 </length>
      </attribute>
   </attributes>
   <operations>
      <operation type="method" override="False">
         <name> TemporalAttribute </name>
         <comment> construtor </comment>
         <parameters>
            <in>
               <name> typ </name>
               <datatype> string </datatype>
               <length> 10 </length>
            </in>
            <in>
               <name> val </name>
               <datatype> Object </datatype>
            </in>
         </parameters>
         <body language="SQL"/>
```

```
</operation>
<operation type="method" override="False">
   <name> TemporalAttribute </name>
   <comment> construtor </comment>
   <parameters>
      <in>
         <name> typ </name>
         <datatype> string </datatype>
         <length> 10 </length>
      </in>
      <in>
         <name> val </name>
         <datatype> Object </datatype>
      </in>
      <in>
         <name> iValidTime </name>
         <datatype> instant </datatype>
      </in>
      <in>
         <name> fValidTime </name>
         <datatype> instant </datatype>
      </in>
   </parameters>
   <body language="SQL"/>
</operation>
<operation type="method" override="False">
   <name> getCurrent </name>
   <comment> retorna o valor corrente do atributo </comment>
   <parameters>
      <return>
         <datatype> InstantAttribute </datatype>
      </return>
   </parameters>
   <body language="SQL"/>
</operation>
<operation type="method" override="False">
   <name> getHistory </name>
   <comment> retorna o historico dos valores do atributo e seus
             respectivos rotulos temporais </comment>
   <parameters>
      <return>
         <datatype> set(InstantAttribute) </datatype>
      </return>
   </parameters>
   <body language="SQL"/>
</operation>
<operation type="method" override="False">
   <name> getHistoryOfValue </name>
   <comment> retorna o historico dos valores do atributo </comment>
   <parameters>
      <in>
         <name> val </name>
         <datatype> Object </datatype>
      </in>
      <return>
         <datatype> set(TemporalLabel) </datatype>
      </return>
   </parameters>
   <body language="SQL"/>
</operation>
<operation type="method" override="False">
   <name> getValueAt </name>
   <comment> retorna os valores do atributo em um determinado
             instante </comment>
   <parameters>
      <in>
         <name> at </name>
         <datatype> instant </datatype>
      </in>
      <return>
         <datatype> instantAttribute </datatype>
      </return>
   </parameters>
```

```
         <body language="SQL"/>
      </operation>
      <operation type="method" override="False" visibility="private">
         <name> setFTransactionTime </name>
         <comment> atualiza o instante de transacao final do atributo
         </comment>
         <parameters>
            <in>
               <name> validAt </name>
               <datatype> instant </datatype>
            </in>
            <in>
               <name> fTransTime </name>
               <datatype> instant </datatype>
            </in>
         </parameters>
         <body language="SQL"/>
      </operation>
      <operation type="method" override="False">
          <name> setFValidTime </name>
          <comment> atualiza o instante de validade final do atributo
          </comment>
          <parameters>
             <in>
                <name> validAt </name>
                <datatype> instant </datatype>
             </in>
             <in>
                <name> fvalidTime </name>
                <datatype> instant </datatype>
             </in>
          </parameters>
          <body language="SQL"/>
      </operation>
      <operation type="method" override="False">
         <name> updateValue </name>
         <comment> atualiza os tempos de validade inicial e final do
                   atributo </comment>
         <parameters>
            <in>
               <name> at </name>
               <datatype> instant </datatype>
            </in>
            <in>
               <name> newVal </name>
               <datatype> Object </datatype>
            </in>
            <in>
               <name> ivalidTime </name>
               <datatype> instant </datatype>
            </in>
            <in>
               <name> fvalidTime </name>
               <datatype> instant </datatype>
            </in>
         </parameters>
         <body language="SQL"/>
      </operation>
   </operations>
   <relationships>
      <relationship type="association">
         <name> currentAttr </name>
         <cardinality> 1:1 </cardinality>
         <relatedClassName> instantAttribute </relatedClassName>
      </relationship>
      <relationship type="aggregation">
         <cardinality> n </cardinality>
         <relatedClassName> instantAttribute </relatedClassName>
         <passingBy> reference </passingBy>
      </relationship>
   </relationships>
</class>
```

```
<class visibility="public">
   <name> TemporalRelationship </name>
   <comment> armazena o conjunto de valores assumidos pelos relacionamentos
             temporais </comment>
   <operations>
      <operation type="method" override="False">
         <name> TemporalRelationship </name>
         <comment> construtor </comment>
         <parameters>
            <in>
               <name> o1 </name>
               <datatype> OIDt </datatype>
            </in>
            <in>
               <name> o2 </name>
               <datatype> OIDt </datatype>
            </in>
         </parameters>
         <body language="SQL"/>
      </operation>
      <operation type="method" override="False">
         <name> TemporalRelationship </name>
         <comment> construtor </comment>
         <parameters>
            <in>
               <name> o1 </name>
               <datatype> OIDt </datatype>
            </in>
            <in>
               <name> o2 </name>
               <datatype> OIDt </datatype>
            </in>
            <in>
               <name> iValidTime </name>
               <datatype> instant </datatype>
            </in>
            <in>
               <name> fValidTime </name>
               <datatype> instant </datatype>
            </in>
         </parameters>
         <body language="SQL"/>
      </operation>
      <operation type="method" override="False">
         <name> getCurrent </name>
         <comment> retorna o relacionamento corrente do objeto </comment>
         <parameters>
            <return>
               <datatype> InstantRelationship </datatype>
            </return>
         </parameters>
         <body language="SQL"/>
      </operation>
      <operation type="method" override="False">
         <name> getHistory </name>
         <comment> retorna todos os relacionamentos do objeto e seus
                   respectivos rotulos temporais </comment>
         <parameters>
            <return>
               <datatype> set(TnstantRelationship) </datatype>
            </return>
         </parameters>
         <body language="SQL"/>
      </operation>
      <operation type="method" override="False">
         <name> getHistoryOfValue </name>
         <comment> retorna o historico dos relacionamentos do objetos
                   passados como parametro </comment>
         <parameters>
           <in>
              <name> o1 </name>
              <datatype> OIDt </datatype>
           </in>
```

```
      <in>
         <name> o2 </name>
         <datatype> OIDt </datatype>
      </in>
      <return>
         <datatype> set(TemporalLabel) </datatype>
      </return>
   </parameters>
   <body language="SQL"/>
</operation>
<operation type="method" override="False">
   <name> getValueAt </name>
   <comment> retorna os relacionamentos de um ojeto em um determinado
             instante </comment>
   <parameters>
      <in>
         <name> at </name>
         <datatype> instant </datatype>
      </in>
      <return>
         <datatype> InstantRelationship </datatype>
      </return>
   </parameters>
   <body language="SQL"/>
</operation>
<operation type="method" override="False" visibility="private">
   <name> setFTransactionTime </name>
   <comment> atualiza o instante de transação final do relacionamento
   </comment>
   <parameters>
      <in>
         <name> validAt </name>
         <datatype> instant </datatype>
      </in>
      <in>
         <name> fTransTime </name>
         <datatype> instant </datatype>
      </in>
   </parameters>
   <body language="SQL"/>
</operation>
<operation type="method" override="False" visibility="private">
   <name> setFValidTime </name>
   <comment> atualiza o instante de validade final do relacionamento
   </comment>
   <parameters>
      <in>
         <name> validAt </name>
         <datatype> instant </datatype>
      </in>
      <in>
         <name> fValidTime </name>
         <datatype> instant </datatype>
      </in>
   </parameters>
   <body language="SQL"/>
</operation>
<operation type="method" override="False">
   <name> updateValue </name>
   <comment> atualiza os tempos de validade inicial e final do
             relacionamento </comment>
   <parameters>
      <in>
         <name> at </name>
         <datatype> instant </datatype>
      </in>
      <in>
         <name> newO1 </name>
         <datatype> OIDt </datatype>
      </in>
      <in>
         <name> NewO2 </name>
         <datatype> OIDt </datatype>
```

```
            </in>
            <in>
               <name> iValidTime </name>
               <datatype> instant </datatype>
            </in>
            <in>
               <name> fValidTime </name>
               <datatype> instant </datatype>
            </in>
         </parameters>
         <body language="SQL"/>
      </operation>
   </operations>
   <relationships>
      <relationship type="association">
         <name> currentRel </name>
         <cardinality> 1:1 </cardinality>
         <relatedClassName> InstantRelationship </relatedClassName>
      </relationship>
      <relationship type="aggregation">
         <cardinality> n </cardinality>
         <relatedClassName> InstantRelationship </relatedClassName>
         <passingBy> Reference </passingBy>
      </relationship>
   </relationships>
</class>

<!-- CLASSES DO MODELO -->
<class type="abstract" root="True">
   <name> Object </name>
   <comment> classe com atributos e operacoes comuns a todos os
             objetos </comment>
   <attributes>
      <attribute temporal="False" nulls="False">
         <name> OID </name>
         <datatype> OIDt </datatype>
      </attribute>
   </attributes>
   <operations>
      <operation type="method" override="False">
         <name> Object </name>
         <comment> construtor </comment>
         <body language="SQL"/>
      </operation>
      <operation type="method" override="False">
         <name> Object </name>
         <comment> construtor </comment>
         <parameters>
            <in>
               <name> ascendId </name>
               <datatype> OIDt </datatype>
            </in>
         </parameters>
         <body language="SQL"/>
      </operation>
      <operation type="method" override="False">
         <name> Object </name>
         <comment> construtor </comment>
         <parameters>
            <in>
               <name> entityName </name>
               <datatype> string </datatype>
               <length> 30 </length>
            </in>
         </parameters>
         <body language="SQL"/>
      </operation>
      <operation type="method" override="False">
         <name> Object </name>
         <comment> construtor </comment>
         <parameters>
            <in>
               <name> entityId </name>
```

```
            <datatype> integer </datatype>
         </in>
         <in>
            <name> classId </name>
            <datatype> integer </datatype>
         </in>
         <in>
            <name> versionId </name>
            <datatype> integer </datatype>
         </in>
      </parameters>
      <body language="SQL"/>
   </operation>
   <operation type="method" override="False">
      <name> delete </name>
      <comment> deleta logicamente o objeto </comment>
      <conditions>
         <condition> se allReferences for true, entao deletar todos
                     os descendentes </condition>
         <condition> nao poderao ser removidos objetos que estejam
                     sendo referidos por outros </condition>
      </conditions>
      <parameters>
         <in>
            <name> allReferences </name>
            <datatype> boolean </datatype>
         </in>
      </parameters>
      <body language="SQL"/>
   </operation>
   <operation type="method" override="False">
      <name> deleteObjectTree </name>
      <comment> deleta logicamente o objeto e seus descendentes </comment>
      <conditions>
         <condition> se allReferences for true, entao deletar todos
                     os descendentes </condition>
      </conditions>
      <parameters>
         <in>
            <name> allReferences </name>
            <datatype> boolean </datatype>
         </in>
      </parameters>
      <body language="SQL"/>
   </operation>
   <operation type="method" override="False" visibility="private">
      <name> findVersion </name>
      <comment> retorna o numero da versao a ser criada </comment>
      <parameters>
         <in>
            <name> entityId </name>
            <datatype> integer </datatype>
         </in>
         <in>
            <name> classId </name>
            <datatype> integer </datatype>
         </in>
      </parameters>
      <body language="SQL"/>
   </operation>
   <operation type="method" override="False">
      <name> getAscendant </name>
      <comment> retorna todas as versoes ascendentes do objeto </comment>
      <parameters>
         <return>
            <datatype> set (OIDt) </datatype>
         </return>
      </parameters>
      <body language="SQL"/>
   </operation>
   <operation type="method" override="False">
      <name> getAscendant </name>
      <comment> retorna o ascendente da classe correspondente </comment>
```

```
   <parameters>
      <in>
         <name> className </name>
         <datatype> string </datatype>
         <length> 30 </length>
      </in>
      <return>
         <datatype> OIDt </datatype>
      </return>
   </parameters>
   <body language="SQL"/>
</operation>
<operation type="method" override="False" visibility="private">
   <name> getClassId </name>
   <comment> retorna o identificador da classe </comment>
   <parameters>
      <return>
         <datatype> integer </datatype>
      </return>
   </parameters>
   <body language="SQL"/>
</operation>
<operation type="method" override="False">
   <name> getClassName </name>
   <comment> retorna o nome da classe a qual o objeto pertence </comment>
   <parameters>
      <return>
         <datatype>string </datatype>
         <length> 30 </length>
      </return>
   </parameters>
   <body language="SQL"/>
</operation>
<operation type="method" override="False">
   <name> getCompleteObject </name>
   <comment> retorna todos os atributos de um objeto  </comment>
   <parameters>
      <return>
         <datatype> set (object) </datatype>
      </return>
   </parameters>
   <body language="SQL"/>
</operation>
<operation type="method" override="False">
   <name> getCorrespondence </name>
   <comment> retorna a correspond_ncia entre o objeto e a classe
             especificada </comment>
   <parameters>
      <in>
         <name> ascendClassName </name>
         <datatype> string </datatype>
         <length> 30 </length>
      </in>
      <return>
         <datatype> string </datatype>
         <length> 30 </length>
      </return>
   </parameters>
   <body language="SQL"/>
</operation>
<operation type="method" override="False">
   <name> getCorrespondenceAsc </name>
   <comment> retorna a cardinalidade do ascendente na correspondencia
             entre o objeto e a classe especificada </comment>
   <parameters>
      <in>
         <name> ascendClassName </name>
         <datatype> string </datatype>
         <length> 30 </length>
      </in>
      <return>
         <datatype> string </datatype>
         <length> 30 </length>
```

```
      </return>
   </parameters>
   <body language="SQL"/>
</operation>
<operation type="method" override="False">
   <name> getCorrespondenceDesc </name>
   <comment> retorna a cardinalidade do descendente na correspondencia
             entre o objeto e a classe especificada </comment>
   <parameters>
      <in>
         <name> ascendClassName </name>
         <datatype> string </datatype>
         <length> 30 </length>
      </in>
      <return>
         <datatype> string </datatype>
         <length> 30 </length>
      </return>
   </parameters>
   <body language="SQL"/>
</operation>
<operation type="method" override="False">
   <name> getDescendant </name>
   <comment> retorna o conjunto dos descendentes do objeto </comment>
   <parameters>
      <return>
         <datatype> set(OIDt) </datatype>
      </return>
   </parameters>
   <body language="SQL"/>
</operation>
<operation type="method" override="False">
   <name> getDescendant </name>
   <comment> retorna o conjunto dos descendentes do objeto de uma
             determinada classe </comment>
   <parameters>
      <in>
         <name> className </name>
         <datatype> string </datatype>
         <length> 30 </length>
      </in>
      <return>
         <datatype> OIDt </datatype>
      </return>
   </parameters>
   <body language="SQL"/>
</operation>
<operation type="method" override="False">
   <name> getEntityId </name>
   <comment> retorna o identificador da entidade </comment>
   <parameters>
      <in>
         <name> entityName </name>
         <datatype> string </datatype>
         <length> 30 </length>
      </in>
      <return>
         <datatype> integer </datatype>
      </return>
   </parameters>
   <body language="SQL"/>
</operation>
<operation type="method" override="False">
   <name> getNickName </name>
   <comment> retorna o apelido dos objetos da classe </comment>
   <parameters>
      <return>
         <datatype> set(string) </datatype>
         <length> 30 </length>
      </return>
   </parameters>
   <body language="SQL"/>
</operation>
```

```
<operation type="method" override="False">
   <name> getObject </name>
   <comment> retorna os valores dos atributos de um determinado
             objeto </comment>
   <parameters>
      <return>
         <datatype> object </datatype>
      </return>
   </parameters>
   <body language="SQL"/>
</operation>
<operation type="method" override="False">
   <name> getOID </name>
   <comment> retorna o identificador do objeto </comment>
   <parameters>
      <return>
         <datatype> OIDt </datatype>
      </return>
   </parameters>
   <body language="SQL"/>
</operation>
<operation type="method" override="False" visibility="protected">
   <name> isDeleteAllowed </name>
   <comment> retorna se um objeto pode ser excluido ou nao </comment>
   <conditions>
      <condition> se allReferences for true, entao deletar todos os
                  descendentes </condition>
      <condition> nao poderao ser removidos objetos que estejam sendo
                  referidos por outros </condition>
   </conditions>
   <parameters>
      <in>
         <name> allReferences </name>
         <datatype> boolean </datatype>
      </in>
      <return>
         <datatype> boolean </datatype>
      </return>
   </parameters>
   <body language="SQL"/>
</operation>
<operation type="method" override="False" visibility="protected">
   <name> isDeleteTreeAllowed </name>
   <comment> retorna se um objeto e todos os seus descendentes podem ser
             excluido ou nao </comment>
   <conditions>
      <condition> se allReferences for true, entao deletar todos os
                  descendentes </condition>
      <condition> nao poderao ser removidos objetos que estejam sendo
                  referidos por outros </condition>
   </conditions>
   <parameters>
      <in>
         <name> allReferences </name>
         <datatype> boolean </datatype>
      </in>
      <return>
         <datatype> boolean </datatype>
      </return>
   </parameters>
   <body language="SQL"/>
</operation>
<operation type="method" override="False" visibility="private">
   <name> verifyAscendId </name>
   <comment> retorna se um determinado objeto e seu ascendente </comment>
   <parameters>
      <in>
         <name> ascendId </name>
         <datatype> OIDt </datatype>
      </in>
      <return>
         <datatype> boolean </datatype>
      </return>
```

```
         </parameters>
         <body language="SQL"/>
      </operation>
      <operation type="method" override="False" visibility="private">
         <name> verifyEntityName </name>
         <comment> verifica se o objeto pertence a uma determinada entidade
         </comment>
         <parameters>
            <in>
               <name> entName </name>
               <datatype> string </datatype>
               <length> 30 </length>
            </in>
            <return>
               <datatype> boolean </datatype>
            </return>
         </parameters>
         <body language="SQL"/>
      </operation>
   </operations>
   <relationships>
      <relationship type="association">
         <name> nickname </name>
         <cardinality> 0:n </cardinality>
         <relatedClassName> Name </relatedClassName>
         <inverse> nicknameOf </inverse>
      </relationship>
   </relationships>
</class>

<class type="abstract">
   <name> TemporalObject </name>
   <comment> controla os dados temporais dos objetos </comment>
   <superClass> Object </superClass>
   <attributes>
      <attribute temporal="True" nulls="False">
         <name> alive </name>
         <datatype> boolean </datatype>
         <defaultValue> True </defaultValue>
      </attribute>
   </attributes>
   <operations>
      <operation type="method" override="False">
         <name> TemporalObject </name>
         <comment> Construtor </comment>
         <body language="SQL"/>
      </operation>
      <operation type="method" override="False">
         <name> TemporalObject </name>
         <comment> Construtor </comment>
         <parameters>
            <in>
               <name> ascendId </name>
               <datatype> OIDt </datatype>
            </in>
         </parameters>
         <body language="SQL"/>
      </operation>
      <operation type="method" override="False">
         <name> TemporalObject </name>
         <comment> Construtor </comment>
         <parameters>
            <in>
               <name> entityName </name>
               <datatype> string </datatype>
               <length> 30 </length>
            </in>
         </parameters>
         <body language="SQL"/>
      </operation>
      <operation type="method" override="False">
         <name> TemporalObject </name>
         <comment> Construtor </comment>
```

```
   <parameters>
      <in>
         <name> entityId </name>
         <datatype> integer </datatype>
      </in>
      <in>
         <name> classId </name>
         <datatype> integer </datatype>
      </in>
      <in>
         <name> versionId </name>
         <datatype> integer </datatype>
      </in>
   </parameters>
   <body language="SQL"/>
</operation>
<operation type="method" override="False">
   <name> classTempLabel </name>
   <comment> percorre os atributos e relacionamentos temporais
             do objeto atualizando os tempos de validade e
             transacao final de null para o tempo da transacao
   </comment>
   <body language="SQL"/>
</operation>
<operation type="method" override="True">
   <name> delete </name>
   <comment> deleta logicamente o objeto </comment>
   <body language="SQL"/>
</operation>
<operation type="method" override="False">
   <name> getAlive </name>
   <comment> retorna se o objeto esta ativo ou nao </comment>
   <parameters>
      <return>
         <datatype> boolean </datatype>
      </return>
   </parameters>
   <body language="SQL"/>
</operation>
<operation type="method" override="False" final="True">
   <name> getAttributeHistory </name>
   <comment> retorna o conjunto de valores do atributo e seus
             respectivos rotulos temporais.  Se nao for um atributo
             temporal, os rotulos retornarao null </comment>
   <parameters>
      <in>
         <name> atribName </name>
         <datatype> string </datatype>
         <length> 30 </length>
      </in>
      <return>
         <datatype> set(InstantAttribute) </datatype>
      </return>
   </parameters>
   <body language="SQL"/>
</operation>
<operation type="method" override="False" final="True">
   <name> getAttributeValueAt </name>
   <comment> retorna o valor do atributo em um determinado instante
             e o valor atual com seus respectivos rotulos temporais.
             Se nao for um atributo temporal, os rotulos retornarao
             null </comment>
   <parameters>
      <in>
         <name> atribName </name>
         <datatype> string </datatype>
         <length> 30 </length>
      </in>
      <in>
         <name> i </name>
         <datatype> instant </datatype>
      </in>
      <return>
```

```
         <datatype> InstantAttribute </datatype>
      </return>
   </parameters>
   <body language="SQL"/>
</operation>
<operation type="method" override="False">
   <name> getLifetimeI </name>
   <comment> retorna o instante de vida inicial do objeto </comment>
   <parameters>
      <return>
         <datatype> instant </datatype>
      </return>
   </parameters>
   <body language="SQL"/>
</operation>
<operation type="method" override="False">
   <name> getLifetimeF </name>
   <comment> retorna o instante de vida final do objeto </comment>
   <parameters>
      <return>
         <datatype> instant </datatype>
      </return>
   </parameters>
   <body language="SQL"/>
</operation>
<operation type="method" override="False" final="True">
   <name> getObjectHistory </name>
   <comment> retorna o historico dos objetos</comment>
   <parameters>
      <return>
         <datatype> set(instantAttribute) </datatype>
      </return>
   </parameters>
   <body language="SQL"/>
</operation>
<operation type="method" override="False" final="True">
   <name> getRelationshipHistory </name>
   <comment> retorna o conjunto de relacionamentos de um determinado
             objeto e seus rotulos temporais se relacionamento temporal
             ou o valor atual com os rotulos null se relacionamento nao
             temporal </comment>
   <parameters>
      <in>
         <name> relatedObjId </name>
         <datatype> OIDt </datatype>
      </in>
      <in>
         <name> relatedName </name>
         <datatype> string </datatype>
         <length> 30 </length>
      </in>
      <return>
         <datatype> set(InstantRelationship) </datatype>
      </return>
   </parameters>
   <body language="SQL"/>
</operation>
<operation type="method" override="False" final="True">
   <name> getRelationshipHistoryAt </name>
   <comment> retorna o conjunto de relacionamentos de um determinado
             objeto e seus rotulos temporais se relacionamento temporal
             ou o valor atual com rotulos null se relacionamento nao
             temporal </comment>
   <parameters>
      <in>
         <name> relatedObjId </name>
         <datatype> OIDt </datatype>
      </in>
      <in>
         <name> relatedName </name>
         <datatype> string </datatype>
         <length> 30 </length>
      </in>
```

```
            <in>
               <name> i </name>
               <datatype> instant </datatype>
            </in>
            <return>
               <datatype> InstantRelationship </datatype>
            </return>
         </parameters>
         <body language="SQL"/>
      </operation>
      <operation type="method" override="False" final="True">
         <name> setTemporalAttribute </name>
         <comment> atualiza o valor do atributo temporal armazenando os
                   respectivos rotulos temporais </comment>
         <parameters>
            <in>
               <name> attribName </name>
               <datatype> string </datatype>
               <length> 30 </length>
            </in>
            <in>
               <name> newValue </name>
               <datatype> Object </datatype>
            </in>
         </parameters>
         <body language="SQL"/>
      </operation>
      <operation type="method" override="False" final="True">
         <name> setTemporalRelationship </name>
         <comment> atualiza o valor do relacionamento temporal armazenando
                   os respectivos rotulos temporais </comment>
         <parameters>
            <in>
               <name> relatedName </name>
               <datatype> string </datatype>
               <length> 30 </length>
            </in>
            <in>
               <name> newO1 </name>
               <datatype> OIDt </datatype>
            </in>
            <in>
               <name> newO2 </name>
               <datatype> OIDt </datatype>
            </in>
         </parameters>
         <body language="SQL"/>
      </operation>
   </operations>
</class>

<class type="abstract">
   <name> TemporalVersion </name>
   <comment> Controla os dados das versoes dos objetos </comment>
   <superClass> TemporalObject </superClass>
   <attributes>
      <attribute temporal="True" nulls="True">
         <name> ascendant </name>
         <datatype> set(OIDt) </datatype>
         <defaultValue> NULL </defaultValue>
      </attribute>
      <attribute temporal="True" nulls="True">
         <name> descendant </name>
         <datatype> set(OIDt) </datatype>
         <defaultValue> NULL </defaultValue>
      </attribute>
      <attribute temporal="True" nulls="True">
         <name> successor </name>
         <datatype> set(OIDt) </datatype>
         <defaultValue> NULL </defaultValue>
      </attribute>
      <attribute temporal="False" nulls="True">
         <name> predecessor </name>
```

```
      <datatype> set(OIDt) </datatype>
      <defaultValue> NULL </defaultValue>
   </attribute>
   <attribute temporal="True" nulls="False">
      <name> status </name>
      <datatype> char </datatype>
      <defaultValue> 'W' </defaultValue>
      <checkConstraint> IN ('W', 'S', 'C', 'D') </checkConstraint>
   </attribute>
   <attribute temporal="False" nulls="False">
      <name> configuration </name>
      <datatype> boolean </datatype>
      <defaultValue> False </defaultValue>
   </attribute>
</attributes>
<operations>
   <operation type="method" override="False">
      <name> TemporalVersion </name>
      <comment> construtor </comment>
      <body language="SQL"/>
   </operation>
   <operation type="method" override="True">
      <name> TemporalVersion </name>
      <comment> construtor </comment>
      <parameters>
         <in>
            <name> ascendId </name>
            <datatype> set(OIDt) </datatype>
         </in>
      </parameters>
      <body language="SQL"/>
   </operation>
   <operation type="method" override="True">
      <name> TemporalVersion </name>
      <comment> construtor </comment>
      <parameters>
         <in>
            <name> entityName </name>
            <datatype> string </datatype>
            <length> 30 </length>
         </in>
      </parameters>
      <body language="SQL"/>
   </operation>
   <operation type="method" override="False">
      <name> TemporalVersion </name>
      <comment> Construtor </comment>
      <parameters>
         <in>
            <name> entityId </name>
            <datatype> integer </datatype>
         </in>
         <in>
            <name> classId </name>
            <datatype> integer </datatype>
         </in>
         <in>
            <name> versionId </name>
            <datatype> integer </datatype>
         </in>
      </parameters>
      <body language="SQL"/>
   </operation>
   <operation type="method" override="False" visibility="private">
      <name> TemporalVersion </name>
      <comment> Construtor </comment>
      <parameters>
         <in>
            <name> predecId </name>
            <datatype> set(OIDt) </datatype>
         </in>
         <in>
            <name> ascendId </name>
```

```
         <datatype> set(OIDt) </datatype>
      </in>
      <in>
         <name> config </name>
         <datatype> boolean </datatype>
      </in>
   </parameters>
   <body language="SQL"/>
</operation>
<operation type="method" override="False" visibility="private">
   <name> addAscendant </name>
   <comment> adiciona um objeto como ascendente </comment>
   <parameters>
      <in>
         <name> ascendId </name>
         <datatype> OIDt </datatype>
      </in>
      <return>
         <datatype> boolean </datatype>
      </return>
   </parameters>
   <body language="SQL"/>
</operation>
<operation type="method" override="False" visibility="private">
   <name> addDescendant </name>
   <comment> adiciona um objeto como descendente </comment>
   <parameters>
      <in>
         <name> descendId </name>
         <datatype> OIDt </datatype>
      </in>
      <return>
         <datatype> boolean </datatype>
      </return>
   </parameters>
   <body language="SQL"/>
</operation>
<operation type="method" override="False" visibility="private">
   <name> addSuccessor </name>
   <comment> adiciona um objeto como sucessor </comment>
   <parameters>
      <in>
         <name> succID  </name>
         <datatype> OIDt </datatype>
      </in>
   </parameters>
   <body language="SQL"/>
</operation>
<operation type="method" override="True">
   <name> delete </name>
   <comment> deleta logicamente um objeto </comment>
   <conditions>
      <condition> se allReferences for true, entao deletar todos
                  os descendentes </condition>
      <condition> nao poderao ser removidos objetos que estejam
                  sendo referidos por outros </condition>
   </conditions>
   <parameters>
      <in>
         <name> allReferences  </name>
         <datatype> boolean </datatype>
      </in>
   </parameters>
   <body language="SQL"/>
</operation>
<operation type="method" override="False">
   <name> deleteObjectTree </name>
   <comment> deleta um objeto e seus descendentes </comment>
   <parameters>
      <in>
         <name> allReferences  </name>
         <datatype> boolean </datatype>
      </in>
```

```
   </parameters>
   <body language="SQL"/>
</operation>
<operation type="method" override="False" final="True">
   <name> derive </name>
   <comment> permite que uma nova versao seja criada para o objeto
   </comment>
   <parameters>
      <in>
         <name> versionId </name>
         <datatype> set(OIDt) </datatype>
      </in>
   </parameters>
   <body language="SQL"/>
</operation>
<operation type="method" override="False" final="True">
   <name> derive </name>
   <comment> permite que uma nova versao seja criada para o objeto
   </comment>
   <parameters>
      <in>
         <name> versionId </name>
         <datatype> set(OIDt) </datatype>
      </in>
      <in>
         <name> ascendId </name>
         <datatype> set(OIDt) </datatype>
      </in>
      <in>
         <name> config </name>
         <datatype> boolean </datatype>
      </in>
   </parameters>
   <body language="SQL"/>
</operation>
<operation type="method" override="True">
   <name> getAscendant </name>
   <comment> retorna os ascendentes de um objeto </comment>
   <parameters>
      <return>
         <datatype> set(OIDt) </datatype>
      </return>
   </parameters>
   <body language="SQL"/>
</operation>
<operation type="method" override="True">
   <name> getAscendant </name>
   <comment> retorna o ascendente de uma determinada classe </comment>
   <parameters>
      <in>
         <name> className </name>
         <datatype> string </datatype>
         <length> 30 </length>
      </in>
      <return>
         <datatype> OIDt </datatype>
      </return>
   </parameters>
   <body language="SQL"/>
</operation>
<operation type="method" override="False">
   <name> getConfiguration </name>
   <comment>  retorna a versao configurada </comment>
   <parameters>
      <return>
         <datatype> OIDt </datatype>
      </return>
   </parameters>
   <body language="SQL"/>
</operation>
<operation type="method" override="False">
   <name> getCompleteObject </name>
   <comment>  retorna todos os atributos de um objeto </comment>
```

```
   <body language="SQL"/>
</operation>
<operation type="method" override="True">
   <name> getDescendant </name>
   <comment> retorna os descendentes de um objeto </comment>
   <parameters>
      <return>
         <datatype> set(OIDt) </datatype>
      </return>
   </parameters>
   <body language="SQL"/>
</operation>
<operation type="method" override="False">
   <name> getDescendant </name>
   <comment> retorna os descendentes de um objeto </comment>
   <parameters>
      <in>
         <name> className </name>
         <datatype> string </datatype>
         <length> 30 </length>
      </in>
      <in>
         <name> criterion </name>
         <datatype> string </datatype>
         <length> 50 </length>
      </in>
      <return>
         <datatype> OIDt </datatype>
      </return>
   </parameters>
   <body language="SQL"/>
</operation>
<operation type="method" override="False" final="True">
   <name> getOIDControl </name>
   <comment> retorna o identificador do controle do objeto
             versionado </comment>
   <parameters>
      <return>
         <datatype> OIDt </datatype>
      </return>
   </parameters>
   <body language="SQL"/>
</operation>
<operation type="method" override="False">
   <name> getPredeccessor </name>
   <comment> retorna os predecessores do objeto </comment>
   <parameters>
      <return>
         <datatype> set(OIDt) </datatype>
      </return>
   </parameters>
   <body language="SQL"/>
</operation>
<operation type="method" override="False">
   <name> getStatus </name>
   <comment> retorna o status do objeto </comment>
   <parameters>
      <return>
         <datatype> char </datatype>
      </return>
   </parameters>
   <body language="SQL"/>
</operation>
<operation type="method" override="False">
   <name> getSuccessor </name>
   <comment>  retorna os sucessores do objeto </comment>
   <parameters>
      <in>
         <name> onlyConfigured </name>
         <datatype> boolean </datatype>
      </in>
      <return>
         <datatype> set(OIDt) </datatype>
```

```
      </return>
   </parameters>
   <body language="SQL"/>
</operation>
<operation type="method" override="False">
   <name> getVersionedObjectId </name>
   <comment> retorna o identificador do objeto versionado </comment>
   <parameters>
      <return>
         <datatype> OIDt </datatype>
      </return>
   </parameters>
   <body language="SQL"/>
</operation>
<operation type="method" override="True" visibility="private">
   <name> isDeleteAllowed </name>
   <comment> retona se um objeto pode ser excluido ou nao
   </comment>
   <conditions>
      <condition> se allReferences for true, entao deletar todos
                  os descendentes </condition>
      <condition> nao poderao ser removidos objetos que estejam
                  sendo referidos por outros </condition>
   </conditions>
   <parameters>
      <return>
         <datatype> boolean </datatype>
      </return>
   </parameters>
   <body language="SQL"/>
</operation>
<operation type="method" override="True" visibility="private">
   <name> isDeleteTreeAllowed </name>
   <comment> retorna se um objeto e todos os seus descendentes podem
             ser excluidos ou nao </comment>
   <conditions>
      <condition> se allReferences for true, entao deletar todos os
                  descendentes </condition>
      <condition> nao poderao ser removidos objetos que estejam sendo
                  referidos por outros </condition>
   </conditions>
   <parameters>
      <return>
         <datatype> boolean </datatype>
      </return>
   </parameters>
   <body language="SQL"/>
</operation>
<operation type="method" override="False" final="True">
   <name> promote </name>
   <comment> permite que uma nova versao seja criada para um objeto
   </comment>
   <conditions>
      <condition> se allAscendant e true entao todos os ascendentes
                  sao promovidos automaticamente </condition>
      <condition> se allReferenced e true entao todos os objetos
                  referenciados serao promovidos </condition>
   </conditions>
   <parameters>
      <in>
         <name> allAscendant </name>
         <datatype> boolean </datatype>
      </in>
      <in>
         <name> allReferenced </name>
         <datatype> boolean </datatype>
      </in>
   </parameters>
   <body language="SQL"/>
</operation>
<operation type="method" override="False" visibility="private">
   <name> removeAscendant </name>
   <comment> remove um objeto ascendente </comment>
```

```
   <parameters>
      <in>
         <name> ascendId </name>
         <datatype> OIDt </datatype>
      </in>
   </parameters>
   <body language="SQL"/>
</operation>
<operation type="method" override="False" visibility="private">
   <name> removeDescendant </name>
   <comment> remove um objeto descendente </comment>
   <parameters>
      <in>
         <name> descendId </name>
         <datatype> OIDt </datatype>
      </in>
   </parameters>
   <body language="SQL"/>
</operation>
<operation type="method" override="False" visibility="private">
   <name> removeSuccessor </name>
   <comment> remove um objeto sucessor </comment>
   <parameters>
      <in>
         <name> succId </name>
         <datatype> OIDt </datatype>
      </in>
   </parameters>
   <body language="SQL"/>
</operation>
<operation type="method" override="False" final="True">
   <name> restore </name>
   <comment> torna um objeto desativado valido novamente </comment>
   <parameters>
      <in>
         <name> OID </name>
         <datatype> OIDt </datatype>
      </in>
      <return>
         <datatype> boolean </datatype>
      </return>
   </parameters>
   <body language="SQL"/>
</operation>
<operation type="method" override="False" visibility="private">
   <name> setAscendant </name>
   <comment> atualiza os objetos ascendentes </comment>
   <parameters>
      <in>
         <name> ascendId </name>
         <datatype> OIDt </datatype>
      </in>
   </parameters>
   <body language="SQL"/>
</operation>
<operation type="method" override="False" visibility="private">
   <name> setDecendant </name>
   <comment> atualiza os objetos descendentes </comment>
   <parameters>
      <in>
         <name> descendId </name>
         <datatype> OIDt </datatype>
      </in>
   </parameters>
   <body language="SQL"/>
</operation>
<operation type="method" override="False" visibility="private">
   <name> setSatus </name>
   <comment> atualiza status do objeto </comment>
   <parameters>
      <in>
         <name> newStatus </name>
         <datatype> char </datatype>
```

```
            </in>
         </parameters>
         <body language="SQL"/>
      </operation>
      <operation type="method" override="False" visibility="private">
         <name> setSuccessor </name>
         <comment> atualiza o objeto sucessor </comment>
         <parameters>
            <in>
               <name> succId </name>
               <datatype> set(OIDt) </datatype>
            </in>
            <return>
               <datatype> boolean </datatype>
            </return>
         </parameters>
         <body language="SQL"/>
      </operation>
      <operation type="method" override="True" visibility="private">
         <name> verifyAscendId </name>
         <comment> retorna se um determinado objeto e seu ascendente </comment>
         <parameters>
            <in>
               <name> ascendId </name>
               <datatype> OIDt </datatype>
            </in>
            <return>
               <datatype> boolean </datatype>
            </return>
         </parameters>
         <body language="SQL"/>
      </operation>
   </operations>
</class>

<class type="final" visibility="public">
   <name> VersionedObjectControl </name>
   <comment> armazena os dados de controle dos objetos versionados e de suas
             versoes </comment>
   <superClass> TemporalObject </superClass>
   <attributes>
      <attribute temporal="True" nulls="False">
         <name> configurationCount </name>
         <datatype> integer </datatype>
         <defaultValue> 0 </defaultValue>
      </attribute>
      <attribute temporal="True" nulls="True">
         <name> currentVersion </name>
         <datatype> OIDt </datatype>
      </attribute>
      <attribute temporal="True" nulls="True">
         <name> firstVersion </name>
         <datatype> OIDt </datatype>
      </attribute>
      <attribute temporal="True" nulls="True">
         <name> lastVersion </name>
         <datatype> OIDt </datatype>
      </attribute>
      <attribute temporal="False" nulls="False">
         <name> nextVersionNumber </name>
         <datatype> integer </datatype>
         <defaultValue> 3 </defaultValue>
      </attribute>
      <attribute temporal="True" nulls="False">
         <name> userCurrentVersion </name>
         <datatype> boolean </datatype>
         <defaultValue> False </defaultValue>
      </attribute>
      <attribute temporal="True" nulls="False">
         <name> versionCount </name>
         <datatype> integer </datatype>
         <defaultValue> 2 </defaultValue>
      </attribute>
```

```
</attributes>
<operations>
   <operation type="method" override="False">
      <name> VersionedObjectControl </name>
      <comment> construtor </comment>
      <parameters>
         <in>
            <name> entityId </name>
            <datatype> integer </datatype>
         </in>
         <in>
            <name> classId </name>
            <datatype> integer </datatype>
         </in>
         <in>
            <name> configCount </name>
            <datatype> integer </datatype>
         </in>
         <in>
            <name> currentV </name>
            <datatype> OIDt </datatype>
         </in>
         <in>
            <name> firstV </name>
            <datatype> OIDt </datatype>
         </in>
         <in>
            <name> lastV </name>
            <datatype> OIDt </datatype>
         </in>
         <in>
            <name> nextVNumber </name>
            <datatype> integer </datatype>
         </in>
         <in>
            <name> userCurrentFlag </name>
            <datatype> boolean </datatype>
         </in>
         <in>
            <name> vCount </name>
            <datatype> integer </datatype>
         </in>
      </parameters>
      <body language="SQL"/>
   </operation>
   <operation type="method" override="False">
      <name> VersionedObjectControl </name>
      <comment> construtor </comment>
      <parameters>
         <in>
            <name> entityId </name>
            <datatype> integer </datatype>
         </in>
         <in>
            <name> classId </name>
            <datatype> integer </datatype>
         </in>
         <in>
            <name> currentV </name>
            <datatype> OIDt </datatype>
         </in>
         <in>
            <name> firstV </name>
            <datatype> OIDt </datatype>
         </in>
         <in>
            <name> lastV </name>
            <datatype> OIDt </datatype>
         </in>
      </parameters>
      <body language="SQL"/>
   </operation>
   <operation type="method" override="False">
```

```
   <name> delete </name>
   <comment> deleta logicamente um objeto </comment>
   <parameters>
      <in>
         <name> entityId  </name>
         <datatype> integer </datatype>
      </in>
      <in>
         <name> classId  </name>
         <datatype> integer </datatype>
      </in>
   </parameters>
   <body language="SQL"/>
</operation>
<operation type="method" override="False">
   <name> getConfigurationCount </name>
   <comment> retorna o numero de versoes configuradas do objeto
   </comment>
   <parameters>
      <return>
         <datatype> integer </datatype>
      </return>
   </parameters>
   <body language="SQL"/>
</operation>
<operation type="method" override="False">
   <name> getCurrentVersion </name>
   <comment> retorna o identificador da versao corrente </comment>
   <parameters>
      <return>
         <datatype> OIDt </datatype>
      </return>
   </parameters>
   <body language="SQL"/>
</operation>
<operation type="method" override="False">
   <name> getFirstVersion </name>
   <comment> retorna o identificador da primeira versao </comment>
   <parameters>
      <return>
         <datatype> OIDt </datatype>
      </return>
   </parameters>
   <body language="SQL"/>
</operation>
<operation type="method" override="False">
   <name> getLastVersion </name>
   <comment> retorna o identificador da ultima versao </comment>
   <parameters>
      <return>
         <datatype> OIDt </datatype>
      </return>
   </parameters>
   <body language="SQL"/>
</operation>
<operation type="method" override="False">
   <name> getNextVersionNumber </name>
   <comment> retorna o proximo numero de versao </comment>
   <parameters>
      <return>
         <datatype> integer </datatype>
      </return>
   </parameters>
   <body language="SQL"/>
</operation>
<operation type="method" override="False">
   <name> getUserCurrentFlag </name>
   <comment> retorna se a versao corrente foi ou nao alterada pelo
             usuario </comment>
   <parameters>
      <return>
         <datatype> boolean </datatype>
      </return>
```

```
   </parameters>
   <body language="SQL"/>
</operation>
<operation type="method" override="False">
   <name> getVersionCount </name>
   <comment> retona a quantidade de versoes do objeto </comment>
   <parameters>
      <return>
         <datatype> integer </datatype>
      </return>
   </parameters>
   <body language="SQL"/>
</operation>
<operation type="method" override="False">
   <name> setCurrentVersion </name>
   <comment> altera a versao corrente para o identificador da ultima
             versao criada </comment>
   <body language="SQL"/>
</operation>
<operation type="method" override="False" visibility="private">
   <name> restore </name>
   <comment> torna ativo um objeto desativado </comment>
   <body language="SQL"/>
</operation>
<operation type="method" override="False">
   <name> setCurrentVersion </name>
   <comment> altera o identificador da versao corrente para o
             identificador informado </comment>
   <parameters>
      <in>
         <name> newVersionId </name>
         <datatype> OIDt </datatype>
      </in>
   </parameters>
   <body language="SQL"/>
</operation>
<operation type="method" override="False">
   <name> setFirstVersion </name>
   <comment> altera o identificador da primeira versao </comment>
   <parameters>
      <in>
         <name> first </name>
         <datatype> OIDt </datatype>
      </in>
   </parameters>
   <body language="SQL"/>
</operation>
<operation type="method" override="False">
   <name> setLastVersion </name>
   <comment> altera o identificador da ultima versao </comment>
   <parameters>
      <in>
         <name> last </name>
         <datatype> OIDt </datatype>
      </in>
   </parameters>
   <body language="SQL"/>
</operation>
<operation type="method" override="False">
   <name> setUserCurrentFlag </name>
   <comment> altera o flag que identifica se a versao corrente foi
             indicada pelo usuario ou pelo sistema </comment>
   <parameters>
      <in>
         <name> flag </name>
         <datatype> boolean </datatype>
      </in>
   </parameters>
   <body language="SQL"/>
</operation>
<operation type="method" override="False">
   <name> updateConfigurationCount </name>
   <comment> atualiza a quantidade de versoes configuradas do objeto
```

```
            </comment>
            <body language="SQL"/>
         </operation>
         <operation type="method" override="False">
            <name> updateVersionCount </name>
            <comment> atualiza a quantidade de versoes do objeto </comment>
            <body language="SQL"/>
         </operation>
         <operation type="method" override="False">
            <name> updateNextersionNumber </name>
            <comment> atualiza o numnero da proxima versao </comment>
            <body language="SQL"/>
         </operation>
      </operations>
   </class>
</classes>
</metaSchema>
```

Definição da DDL em XML

```
<?xml version="1.0"?> <!-- edited with XML Spy v4.0.1
(http://www.xmlspy.com) by Lialda Rossetti (UFRGS) --> <!DOCTYPE
ddlExtended SYSTEM "ddl5.dtd"> <?xml-stylesheet type="text/xsl"
href="D:\UFRGS\Dissertacao\Esquema\ddl1.xsl"?> <ddlExtended>
    <class>
        <type optional="true" choice="true">
            <choice> abstract </choice>
            <choice> final </choice>
        </type>
        <name> className </name>
        <versions temporalVersion="true"> hasVersions </versions>
        <inherit optional="true">
            <inheritedBy optional="true" temporalVersion="false"> byExtension </inheritedBy>
            <name> className </name>
            <correspondence optional="true" choice="true" temporalVersion="true">
                <choice> 1:1 </choice>
                <choice> 1:n </choice>
                <choice> n:1 </choice>
                <choice> n:n </choice>
            </correspondence>
        </inherit>
        <aggregateOf optional="true" multipleOccurrence="true" optionalOccurrence="true">
            <temporal optional="true" temporalVersion="true"/>
            <multiplicity optional="true"> n </multiplicity>
            <name> className </name>
            <by choice="true">
                <choice> byValue </choice>
                <choice> byReference </choice>
            </by>
        </aggregateOf>
        <properties optional="true" multipleOccurrence="true" optionalOccurrence="false">
            <static optional="true" temporalVersion="false"/>
            <temporal optional="true" temporalVersion="true"/>
            <attribute>
                <name> attributeName </name>
                <domain> attributeDomain </domain>
                <defaultValue optional="true"> value </defaultValue>
            </attribute>
        </properties>
        <relationship optional="true" multipleOccurrence="true" optionalOccurrence="false">
            <temporal optional="true" temporalVersion="true"/>
            <name> relationshipName </name>
            <cardinality choice="true">
                <choice> 0:1 </choice>
                <choice> 0:n </choice>
                <choice> 1:1 </choice>
                <choice> 1:n </choice>
                <choice> n:m </choice>
            </cardinality>
            <inverse optional="true">
```

```
                <name> inverseRelationshipName </name>
            </inverse>
            <relatedClass> relatedClassName </relatedClass>
        </relationship>
        <operations optional="true" multipleOccurrence="true" optionalOccurrence="true">
            <static optional="true"/>
            <type optional="true" choice="true">
                <choice> abstract </choice>
                <choice> final </choice>
            </type>
            <name> operationName </name>
            <parameter optional="true" multipleOccurrence="true" optionalOccurrence="true">
                <name> parameterName </name>
                <domain> parameterDomain </domain>
                <defaultValue> value </defaultValue>
            </parameter>
            <return optional="true"> returnedValueDomain </return>
        </operations>
    </class>
</ddlExtended>
```

Anexo C XSLT do TVM e da DDL e Resultados da transformação

Modelo Temporal de Versões

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
   <xsl:output method="html" encoding="UTF-8" indent="no"/>

   <!-- SOBRESCREVE REGRA PADRAO -->
   <xsl:template match="text()">
   </xsl:template>

   <!-- VARIAVEIS E PARAMETROS GLOBAIS -->
   <xsl:variable name="boolean">IN ('T', 'F')</xsl:variable>
   <xsl:variable name="tempVers">False</xsl:variable>
   <xsl:variable name="nomeRelacionamento"/>
   <xsl:variable name="nomeInverso"/>
   <xsl:param name="className" />
   <xsl:param name="operationName"/>

   <!-- RAIZ DO DOCUMENTO -->
   <xsl:template match="classes">
      <xsl:comment> TIPO SET PARA DADOS NÃO TEMPORAIS </xsl:comment>
      <xsl:apply-templates mode="SET" />

      <xsl:comment> TIPOS ESTRUTURADOS </xsl:comment>
      <xsl:apply-templates mode="CRIAR_TIPOS"/>

      <xsl:comment> INCLUSÃO RELACIONAMENTOS PARA OS TIPOS </xsl:comment>
      <xsl:apply-templates mode="REF_TIPOS" />

      <xsl:comment> TABELAS TIPADAS </xsl:comment>
      <xsl:apply-templates mode="CRIAR_CLASSES"/>

      <xsl:comment> INCLUSÃO RELACIONAMENTOS PARA AS CLASSES </xsl:comment>
      <xsl:apply-templates mode="REF_CLASSES" />

      <xsl:comment> GERAÇÃO AUTOMATICA DOS OID'S </xsl:comment>
      <xsl:apply-templates mode="CRIAR_SEQUENCIAS" />

      <xsl:comment> INSERIR DADOS DO MODELO NOS METADADOS </xsl:comment>
      INSERT INTO Schema (name) VALUES (nome);

      <xsl:comment> METADADOS CLASS </xsl:comment>
      <xsl:apply-templates mode="GRAVAR_METADADOS" />-->
   </xsl:template>

   <!-- VERIFICAR TIPO SET -->
   <xsl:template match="class" mode="SET">
      <xsl:apply-templates select="attributes" mode="SET">
         <xsl:with-param name="className">
            <xsl:value-of select="normalize-space(name)"/>
         </xsl:with-param>
      </xsl:apply-templates>
   </xsl:template>

   <!-- DEFINICAO DOS TIPOS - TIPOS ESTRUTURADOS  -->
   <xsl:template match="class" mode="CRIAR_TIPOS">
      <!-- CRIACAO DOS TIPOS DAS CLASSES -->
      CREATE TYPE <xsl:value-of select="name"/>
      <xsl:if test="string-length(superClass)>0">
         UNDER <xsl:value-of select="concat('T', normalize-space(superClass))"/>
      </xsl:if> AS
      <!-- EXECUTA ATRIBUTOS -->
      <xsl:apply-templates select="attributes" />

      <!-- DEFINE TIPO DAS CLASSES -->
      <xsl:if test="@type='abstract'"> NOT INSTANTIABLE </xsl:if>
```

```
   <xsl:if test="@type='final'"> FINAL </xsl:if>

   <!-- PARA GERAR O OID -->
   <xsl:if test="string-length(superClass)=0"> REF USING INTEGER </xsl:if>
   MODE DB2SQL;
</xsl:template>

<!-- INCLUSÃO DOS RELACIONAMENTOS NOS TIPOS  -->
<xsl:template match="class" mode="REF_TIPOS">
   <xsl:apply-templates select="relationships" mode="REF_TIPOS">
      <xsl:with-param name="className">
         <xsl:value-of select="normalize-space(name)"/>
      </xsl:with-param>
   </xsl:apply-templates>

   <!-- INSERIR PROPRIEDADE property EM instantAttribute -->
   <xsl:if test="normalize-space(name)='InstantAttribute'">
      ALTER TYPE <xsl:value-of select="name" />
      ADD ATTRIBUTE property VARCHAR(30);
   </xsl:if>
</xsl:template>

<!-- DEFINIÇÃO DOS CLASSES - TABELAS TIPADAS -->
<xsl:template match="class" mode="CRIAR_CLASSES">
   <xsl:if test="@type!='abstract'">
      CREATE TABLE <xsl:value-of select="name"/> OF <xsl:value-of select="name"/>
      <xsl:if test="string-length(superClass)>0">
         UNDER <xsl:value-of select="superClass"/>
         INHERIT SELECT PRIVILEGES
      </xsl:if>
      (REF IS OID USER GENERATED,
      <xsl:apply-templates select="attributes" mode="CONSTRAINTS">
         <xsl:with-param name="className" >
            <xsl:value-of select="normalize-space(name)" />
         </xsl:with-param>
      </xsl:apply-templates>
   </xsl:if>
</xsl:template>

<!-- INCLUSÃO DOS RELACIONAMENTOS NAS CLASSES -->
<xsl:template match="class" mode="REF_CLASSES">
   <xsl:apply-templates select="relationships" mode="REF_CLASSES">
      <xsl:with-param name="className">
         <xsl:value-of select="normalize-space(name)"/>
      </xsl:with-param>
   </xsl:apply-templates>

   <!-- INSERIR REFERENCIA DE TemporalVersion EM instantAttribute -->
   <xsl:if test="normalize-space(name)='InstantAttribute'">
      ALTER TABLE <xsl:value-of select="name" />
      ALTER refTemporalVersion ADD SCOPE TemporalVersion;
   </xsl:if>
</xsl:template>

<!-- CRIAÇÃO DAS SEQUENCIAS -->
<xsl:template match="class" mode="CRIAR_SEQUENCIAS">
   <xsl:if test="@type!='abstract'">
      <xsl:variable name="nome_sequencia">
         <xsl:value-of select="concat('seq', normalize-space(name))" />
      </xsl:variable>
      <xsl:comment> CRIA SEQUENCIAS PARA OID </xsl:comment>
      CREATE SEQUENCE <xsl:value-of select="$nome_sequencia"/>
      START WITH 1
      INCREMENT BY 1
      NO MAXVALUE
      NO CYCLE;

      <xsl:comment> CRIA TRIGGER PARA INSERÇÃO DO OID </xsl:comment>
      <xsl:variable name="nome_trigger">
         <xsl:value-of select="concat('i', normalize-space(name))" />
      </xsl:variable>

      CREATE TRIGGER <xsl:value-of select="$nome_trigger"/>
```

```
      NO CASCADE BEFORE INSERT ON <xsl:value-of select="normalize-space(name)"/>
      REFERENCING NEW AS N
      FOR EACH ROW MODE DB2SQL
      BEGIN ATOMIC
         DECLARE COD INTEGER;
         SET COD = NEXTVAL FOR <xsl:value-of select="$nome_sequencia" />;
         SET N.OID=<xsl:value-of select="normalize-space(name)"/>(COD);

         <xsl:if test="attributes/attribute[normalize-space(name)='id']">
            SET N.ID=COD;
         </xsl:if>
      END
   </xsl:if>
</xsl:template>

<!-- METADADOS  -->
<xsl:template match="class" mode="GRAVAR_METADADOS">
   <xsl:if test="@metadata='False'">
      INSERT INTO Class (name, tempVers, specializ, visibility,
                         root, refSchema)
         VALUES ('<xsl:value-of select="normalize-space(name)"/>',
                 '<xsl:value-of select="substring($tempVers, 1, 1)" />',
                 '<xsl:value-of select="@type" />',
                 '<xsl:value-of select="@visibility" />',
                 '<xsl:value-of select="substring(@root, 1, 1)" />',
                 (SELECT Shema(id) FROM SCHEMA WHERE name='TVM'));

      <xsl:if test="string-length(superClass)>0">
         <xsl:comment> METADADOS RELATIONSHIP INHERITANCE </xsl:comment>
         INSERT INTO Inheritance (name, relatedClass, single, refClass)
            VALUES (null,
                   '<xsl:value-of select="normalize-space(superClass)"/>',
                   '<xsl:value-of select=
                                 "substring(@multipleInheritance, 1, 1)"/>',
                   (SELECT Class(id) FROM Class
                   WHERE name='<xsl:value-of select=
                                       "normalize-space(name)" />'));
      </xsl:if>

      <xsl:comment> METADADOS ATTRIBUTE </xsl:comment>
      <xsl:apply-templates select="attributes" mode="GRAVAR_METADADOS">
         <xsl:with-param name="className" >
            <xsl:value-of select="normalize-space(name)" />
         </xsl:with-param>
      </xsl:apply-templates>

      <xsl:comment> METADADOS RELATIONSHIP </xsl:comment>
      <xsl:apply-templates select="relationships" mode="GRAVAR_METADADOS">
         <xsl:with-param name="className" >
            <xsl:value-of select="normalize-space(name)" />
         </xsl:with-param>
      </xsl:apply-templates>

      <xsl:comment> METADADOS OPERATION </xsl:comment>
      <xsl:apply-templates select="operations" mode="GRAVAR_METADADOS">
         <xsl:with-param name="className" >
            <xsl:value-of select="normalize-space(name)" />
         </xsl:with-param>
      </xsl:apply-templates>
   </xsl:if>
</xsl:template>

<!-- VERIFICAR TIPO SET -->
<xsl:template match="attributes" mode="SET">
   <xsl:param name="className" select="$className" />
   <xsl:apply-templates select="attribute" mode="SET">
      <xsl:with-param name="className">
         <xsl:value-of select="$className"/>
      </xsl:with-param>
   </xsl:apply-templates>
</xsl:template>

<!-- INCLUSÃO DAS PROPRIEDADES NOS TIPOS ESTRUTURADOS -->
```

```
<xsl:template match="attributes">
   (<xsl:apply-templates />)
</xsl:template>

<!-- INCLUSÃO DAS RESTRIÇÕES PARA AS PROPRIEDADES -->
<xsl:template match="attributes" mode="CONSTRAINTS">
   <!-- PARAMETRO COM O NOME DA CLASSE -->
   <xsl:param name="className" select="$className" />

   <xsl:apply-templates mode="CONSTRAINT">
      <xsl:with-param name="className" select="$className" />
   </xsl:apply-templates>);
</xsl:template>

<!-- INCLUSÃO DAS PROPRIEDADES NOS METADADOS -->
<xsl:template match="attributes" mode="GRAVAR_METADADOS">
   <!-- PARAMETRO COM O NOME DA CLASSE -->
   <xsl:param name="className" select="$className" />

   <xsl:apply-templates mode="METADADOS">
      <xsl:with-param name="className" select="$className" />
   </xsl:apply-templates>
</xsl:template>

<!-- TIPO SET PARA DADOS NÃO TEMPORAIS -->
<xsl:template match="attribute" mode="SET">
   <!-- PARAMETRO COM O NOME DA CLASSE -->
   <xsl:param name="className" select="$className" />

   <xsl:if test="substring(normalize-space(datatype),1,3)='set'
                 and @temporal='False'">
      <!-- EXTRAI O TIPO DE DADO DO CONJUNTO -->
      <xsl:variable name="tipodado" >
         <xsl:value-of select="substring-before(datatype, ')')" />
      </xsl:variable>
      <xsl:variable name="tipo" >
         <xsl:value-of select="substring-after($tipodado, '(')" />
      </xsl:variable>

      <!-- ESPECIFICA TIPO PARA A PROPRIEDADE -->
      CREATE TYPE <xsl:value-of select="concat(normalize-space(name),
                                               'Set')" /> AS
         (<xsl:value-of select="name" /> <xsl:value-of select="$tipo"/>,
          <xsl:value-of select="concat('Ref',$className)" />
          REF(<xsl:value-of select="$className" />))
          REF USING INTEGER
          MODE DB2SQL;

      <!-- CRIA TABELA TIPADA PARA ARMAZENAR OS DADOS DA PROPRIEDADE -->
      CREATE TABLE <xsl:value-of select="name" />  OF
         <xsl:value-of select="concat(normalize-space(name), 'Set')" />
         (REF IS OID USER GENERATED,
         <xsl:value-of select="concat('Ref',$className)" />
         WITH OPTIONS SCOPE <xsl:value-of select="$className" />)

      <!-- CRIA SEQUENCIA PARA O OID -->
      <xsl:variable name="nome_sequencia" >
         <xsl:value-of select="concat('seq', name)" />
      </xsl:variable>
      CREATE SEQUENCE <xsl:value-of select="$nome_sequencia" />
         START WITH 1
         INCREMENT BY 1
         NO MAXVALUE
         NO CYCLE;

      <!-- TRIGGER PARA INCLUIR AUTOMATICAMENTE OS VALORES DE OID -->
      CREATE TRIGGER <xsl:value-of select="concat('i', name)" />
      NO CASCADE BEFORE INSERT ON <xsl:value-of select="name" />
      REFERENCING NEW AS N
      FOR EACH ROW MODE DB2SQL
      SET N.OID=<xsl:value-of select="name" />
                (NEXTVAL FOR <xsl:value-of select="$nome_sequencia" />)
   </xsl:if>
```

```
</xsl:template>

<!-- INCLUSÃO DAS PROPRIEDADES NOS TIPOS ESTRUTURADOS -->
<xsl:template match="attribute">
   <!-- ELIMINA AS PROPRIEDADES TIPO SET NÃO TEMPORAIS  -->
   <xsl:if test="substring(normalize-space(datatype),1,3)!='set'
                 or @temporal != 'False'">
      <xsl:value-of select="name"/>
      <xsl:choose>
         <xsl:when test="normalize-space(datatype)='string'"> VARCHAR
         </xsl:when>
         <xsl:when test="normalize-space(datatype)='boolean'"> CHAR
         </xsl:when>
         <xsl:when test="normalize-space(datatype)='instant'"> TIMESTAMP
         </xsl:when>
         <xsl:when test="normalize-space(datatype)='enum'"> VARCHAR (10)
         </xsl:when>
         <xsl:when test="normalize-space(datatype)='Object'"> VARCHAR (10)
         </xsl:when>
         <xsl:when test="substring(normalize-space(datatype),1,3)='set'">
            <!-- EXTRAI O TIPO DE DADO DO CONJUNTO -->
            <xsl:variable name="tipodado" >
               <xsl:value-of select="substring-before(datatype, ')')" />
            </xsl:variable>
            <xsl:variable name="tipo" >
               <xsl:value-of select="substring-after($tipodado, '(')" />
            </xsl:variable>
            <xsl:value-of select="$tipo" />
         </xsl:when>
         <xsl:otherwise>
            <xsl:value-of select="datatype" />
         </xsl:otherwise>
      </xsl:choose>
      <xsl:if test="string(length)>0"> (<xsl:value-of select="length" />)
      </xsl:if>
      <xsl:if test="position()!=last()" >,</xsl:if>
   </xsl:if>
</xsl:template>

<!-- INCLUSÃO DAS RESTRIÇÕES PARA AS PROPRIEDADES -->
<xsl:template match="attribute" mode="CONSTRAINT">
   <xsl:param name="className" select="$className"/>
   <xsl:if test="@nulls='False' and string-length(defaultValue)=0">
      <xsl:value-of select="name"/> NOT NULL
      <xsl:if test="position()!=last() or string-length(checkConstraint)>0
         or normalize-space(datatype)='boolean'">,</xsl:if>
   </xsl:if>

   <xsl:if test="@nulls='False' and string-length(defaultValue)>0">
      <xsl:value-of select="name"/> NOT NULL WITH DEFAULT
      <xsl:if test="normalize-space(datatype)='boolean'">
         '<xsl:value-of select="substring(normalize-space(defaultValue),
                              1, 1)" />'
      </xsl:if>
      <xsl:if test="normalize-space(datatype)!='boolean'">
         <xsl:value-of select="defaultValue" />
      </xsl:if>
      <xsl:if test="position()!=last() and string-length(checkConstraint)=0">,
      </xsl:if>
   </xsl:if>

   <xsl:if test="@nulls='True' and string-length(defaultValue)>0">
      <xsl:value-of select="name"/>  WITH DEFAULT
      <xsl:value-of select="defaultValue" />
      <xsl:if test="position()!=last() and string-length(checkConstraint)=0">,
      </xsl:if>
   </xsl:if>

   <xsl:if test="normalize-space(datatype)='boolean'">
      CONSTRAINT <xsl:value-of select="name" />
         CHECK (<xsl:value-of select="name"/> <xsl:value-of select="$boolean"/>)
         <xsl:if test="position()!=last()">,</xsl:if>
   </xsl:if>
```

```
   <xsl:if test="string-length(checkConstraint)>0">
      CONSTRAINT <xsl:value-of select="name" />
         CHECK (<xsl:value-of select="name" />
                <xsl:value-of select="checkConstraint" />)
         <xsl:if test="position()!=last()">,</xsl:if>
   </xsl:if>

   <!-- PROPRIEDADE ADICIONADA A INSTANTATTRIBUTE NÃO PODE SER NULA -->
   <!-- INSERIR REFERENCIA DE TemporalVersion EM instantAttribute -->
   <xsl:if test="position()=last() and $className='InstantAttribute'">
      , property NOT NULL,
      refTemporalVersion WITH OPTIONS SCOPE TemporalVersion
   </xsl:if>
</xsl:template>

<!-- INCLUSÃO DAS PROPRIEDADES NOS METADADOS -->
<xsl:template match="attribute" mode="METADADOS">
   <!-- PARAMETRO COM O NOME DA CLASSE -->
   <xsl:param name="className" select="$className" />

   INSERT INTO Attribute (name, temp, visibility, static, type, default, refClass)
      VALUES ('<xsl:value-of select="normalize-space(name)" />',
              '<xsl:value-of select="substring(@temporal, 1, 1)"/>',
              '<xsl:value-of select="@visibility" />',
              '<xsl:value-of select="substring(@static, 1, 1)" />',
              <xsl:text>'</xsl:text>
              <xsl:choose>
                 <xsl:when test="normalize-space(datatype)='string'">VARCHAR
                    <xsl:if test="string(length)>0"> (<xsl:value-of select="length" />)
                    </xsl:if>
                 </xsl:when>
                 <xsl:when test="normalize-space(datatype)='boolean'"> CHAR
                 </xsl:when>
                 <xsl:when test="normalize-space(datatype)='instant'"> TIMESTAMP
                 </xsl:when>
                 <xsl:when test="normalize-space(datatype)='enum'"> VARCHAR (10)
                 </xsl:when>
                 <xsl:when test="normalize-space(datatype)='Object'"> VARCHAR (10)
                 </xsl:when>
                 <xsl:when test="substring(normalize-space(datatype),1,3)='set'">
                    <!-- EXTRAI O TIPO DE DADO DO CONJUNTO -->
                    <xsl:variable name="tipodado" >
                        <xsl:value-of select="substring-before(datatype, ')')" />
                    </xsl:variable>
                    <xsl:variable name="tipo" >
                       <xsl:value-of select="substring-after($tipodado, '(')" />
                    </xsl:variable>

                    <xsl:value-of select="$tipo" />
                 </xsl:when>
                 <xsl:otherwise>
                    <xsl:value-of select="normalize-space(datatype)" />
                 </xsl:otherwise>
              </xsl:choose>
              <xsl:text>',</xsl:text>

              <xsl:if test="string-length(defaultValue)>0">
                 <xsl:choose>
                    <xsl:when test="normalize-space(datatype)='boolean'">
                       '<xsl:value-of select="substring(normalize-space(defaultValue),
                                        1, 1)" />'
                    </xsl:when>
                    <xsl:otherwise>
                       '<xsl:value-of select="normalize-space(defaultValue)" />'
                    </xsl:otherwise>
                 </xsl:choose>,
              </xsl:if>
              <xsl:if
              test="string-length(defaultValue)=0">'',</xsl:if>
              (SELECT Class(id) FROM CLASS WHERE
               name='<xsl:value-of select="$className"/>'));
</xsl:template>
```

```
<!-- INCLUSÃO DOS RELACIONAMENTOS NOS TIPOS  -->
<xsl:template match="relationships"     mode="REF_TIPOS">
   <!-- PARAMETRO COM O NOME DA CLASSE -->
   <xsl:param name="className"> <xsl:value-of select="$className" /> </xsl:param>

   <xsl:apply-templates mode="REF_TIPOS">
      <xsl:with-param name="className" > <xsl:value-of select="$className" />
      </xsl:with-param>
   </xsl:apply-templates>
</xsl:template>

<!-- INCLUSÃO DOS RELACIONAMENTOS NOS TIPOS  -->
<xsl:template match="relationship" mode="REF_TIPOS">
   <!-- PARAMETRO COM O NOME DA CLASSE -->
     <xsl:param name="className">
        <xsl:value-of select="$className" />
     </xsl:param>
     <xsl:variable name="relationshipName">
        <xsl:value-of select="normalize-space(name)" />
     </xsl:variable>

     ALTER TYPE <xsl:value-of select="relatedClassName" />
     ADD ATTRIBUTE <xsl:value-of select="concat('ref',
                   $className, $relationshipName)" />
                   REF (<xsl:value-of select="$className"/>);
</xsl:template>

<!-- INCLUSÃO DOS RELACIONAMENTOS NAS CLASSES  -->
<xsl:template match="relationships"     mode="REF_CLASSES">
   <!-- PARAMETRO COM O NOME DA CLASSE -->
   <xsl:param name="className"> <xsl:value-of select="$className" /> </xsl:param>

   <xsl:apply-templates mode="REF_CLASSES">
      <xsl:with-param name="className" > <xsl:value-of select="$className" />
      </xsl:with-param>
   </xsl:apply-templates>
</xsl:template>

<!-- INCLUSÃO DOS RELACIONAMENTOS NAS CLASSES  -->
<xsl:template match="relationship" mode="REF_CLASSES">
   <!-- PARAMETRO COM O NOME DA CLASSE -->
   <xsl:param name="className"> <xsl:value-of select="$className" /> </xsl:param>

   ALTER TABLE <xsl:value-of select="relatedClassName" />
   ALTER  <xsl:value-of select="concat('ref', $className)" /> ADD SCOPE
   <xsl:value-of select="$className"/>;
</xsl:template>

<!-- INCLUSÃO DOS RELACIONAMENTOS NOS METADADOS -->
<xsl:template match="relationships" mode="GRAVAR_METADADOS">
   <!-- PARAMETRO COM O NOME DA CLASSE -->
   <xsl:param name="className" select="$className" />

   <xsl:apply-templates select="relationship" mode="METADADOS">
      <xsl:with-param name="className" select="$className" />
   </xsl:apply-templates>
</xsl:template>

<!-- INCLUSÃO DOS RELACIONAMENTOS NOS METADADOS -->
<xsl:template match="relationship" mode="METADADOS">
   <!-- PARAMETRO COM O NOME DA CLASSE -->
   <xsl:param name="className" select="$className" />

   <!-- Verifica se relacionamento tem nome -->
   <xsl:variable name="nomeRelacionamento">
      <xsl:if test="string-length(name)>0">
         '<xsl:value-of select="normalize-space(name)"/>'
      </xsl:if>
      <xsl:if test="string-length(name)=0">null</xsl:if>
   </xsl:variable>

   <!-- Verifica se inverso tem nome -->
```

```
   <xsl:variable name="nomeInverso">
      <xsl:if test="string-length(inverse)>0">
         '<xsl:value-of select="normalize-space(inverse)"/>'
      </xsl:if>
      <xsl:if test="string-length(inverse)=0">null</xsl:if>
   </xsl:variable>

   <!-- Seleciona tipo de relacionamento -->
   <xsl:choose>
      <xsl:when test="@type='association'">
         <xsl:comment> RELACIONAMENTO DE ASSOCIACAO </xsl:comment>
         INSERT INTO Association (name, relatedClass, inverse, temp,
                                  cardinality, refClass)
            VALUES (<xsl:value-of select="$nomeRelacionamento"/>,
                    '<xsl:value-of select="normalize-space(relatedClassName)"/>',
                    <xsl:value-of select="$nomeInverso"/>,
                    '<xsl:value-of select="substring(@temporal, 1, 1)"/>',
                    '<xsl:value-of select="normalize-space(cardinality)"/>',
                    (SELECT Class(id) FROM CLASS WHERE
                     name='<xsl:value-of select="$className"/>'));
      </xsl:when>
      <xsl:otherwise>
         <xsl:comment> RELACIONAMENTO DE AGREGACAO </xsl:comment>
         INSERT INTO Aggregation (name, relatedClass, temp, cardinality,
                                  refClass)
            VALUES (<xsl:value-of select="$nomeRelacionamento"/>,
                    '<xsl:value-of select="normalize-space(relatedClassName)"/>',
                    '<xsl:value-of select="substring(@temporal, 1, 1)"/>',
                    '<xsl:value-of select="normalize-space(cardinality)"/>',
                    (SELECT Class(id) FROM CLASS WHERE
                     name='<xsl:value-of select="$className"/>'));
      </xsl:otherwise>
   </xsl:choose>
</xsl:template>

<!-- INCLUSAO DAS PROPRIEDADES NOS METADADOS -->
<xsl:template match="operations" mode="GRAVAR_METADADOS">
   <!-- PARAMETRO COM O NOME DA CLASSE -->
   <xsl:param name="className" select="$className" />

   <xsl:apply-templates select="operation" mode="METADADOS">
      <xsl:with-param name="className" select="$className" />
   </xsl:apply-templates>
</xsl:template>

<xsl:template match="operation" mode="METADADOS">
   <!-- PARAMETRO COM O NOME DA CLASSE -->
   <xsl:param name="className" select="$className" />

   INSERT INTO Operation (name, specializ, visibility, static, refClass)
      VALUES ('<xsl:value-of select="normalize-space(name)"/>',
              null,
              '<xsl:value-of select="@visibility"/>',
              null,
              (SELECT Class(id) FROM CLASS WHERE
               name='<xsl:value-of select="$className"/>'));

   <xsl:apply-templates select="parameters" >
      <xsl:with-param name="operationName">
         <xsl:value-of select="normalize-space(name)" />
      </xsl:with-param>
   </xsl:apply-templates>
</xsl:template>

<xsl:template match="parameters">
   <xsl:param name="operationName" select="$operationName" />

   <xsl:apply-templates select="in">
      <xsl:with-param name="operationName" select="$operationName" />
   </xsl:apply-templates>

   <xsl:apply-templates select="return">
      <xsl:with-param name="operationName" select="$operationName" />
```

```
   </xsl:apply-templates>
</xsl:template>

<xsl:template match="in" >
   <xsl:param name="operationName" select="$operationName" />

   INSERT INTO Parameter (name, datatype, default, refOperationin,
                          refOperationout)
      VALUES ('<xsl:value-of select="normalize-space(name)"/>',
               <xsl:text>'</xsl:text>
               <xsl:choose>
                  <xsl:when test="normalize-space(datatype)='string'">VARCHAR
                     <xsl:if test="string(length)>0">
                        (<xsl:value-of select="length" />)
                     </xsl:if>
                  </xsl:when>
                  <xsl:when test="normalize-space(datatype)='boolean'"> CHAR
                  </xsl:when>
                  <xsl:when test="normalize-space(datatype)='instant'"> TIMESTAMP
                  </xsl:when>
                  <xsl:when test="normalize-space(datatype)='enum'"> VARCHAR (10)
                  </xsl:when>
                  <xsl:when test="normalize-space(datatype)='Object'"> VARCHAR (10)
                  </xsl:when>
                  <xsl:when test="substring(normalize-space(datatype),1,3)='set'">
                     <!-- EXTRAI O TIPO DE DADO DO CONJUNTO -->
                     <xsl:variable name="tipodado" >
                        <xsl:value-of select="substring-before(datatype, ')')" />
                     </xsl:variable>
                     <xsl:variable name="tipo" >
                        <xsl:value-of select="substring-after($tipodado, '(')" />
                     </xsl:variable>
                     <xsl:value-of select="$tipo" />
                  </xsl:when>
                  <xsl:otherwise>
                     <xsl:value-of select="normalize-space(datatype)" />
                  </xsl:otherwise>
               </xsl:choose>
               <xsl:text>',</xsl:text>
               <xsl:choose>
                  <xsl:when test="string-length(normalize-space(defaultValue))>0">
                     '<xsl:value-of select="normalize-space(name)"/>',
                  </xsl:when>
                  <xsl:otherwise> null, </xsl:otherwise>
               </xsl:choose>
               (SELECT Operation (OID) FROM OPERATION WHERE
                name='<xsl:value-of select="$operationName" />'),
               null);
</xsl:template>

<xsl:template match="return" >
   <xsl:param name="operationName" select="$operationName" />

   INSERT INTO Parameter (name, datatype, default, refOperationin,
                          refOperationout)
      VALUES ('<xsl:value-of select="normalize-space(name)"/>',
               <xsl:text>'</xsl:text>
               <xsl:choose>
                  <xsl:when test="normalize-space(datatype)='string'">VARCHAR
                     <xsl:if test="string(length)>0">
                        (<xsl:value-of select="length" />)
                     </xsl:if>
                  </xsl:when>
                  <xsl:when test="normalize-space(datatype)='boolean'"> CHAR
                  </xsl:when>
                  <xsl:when test="normalize-space(datatype)='instant'"> TIMESTAMP
                  </xsl:when>
                  <xsl:when test="normalize-space(datatype)='enum'"> VARCHAR (10)
                  </xsl:when>
                  <xsl:when test="normalize-space(datatype)='Object'"> VARCHAR (10)
                  </xsl:when>
                  <xsl:when test="substring(normalize-space(datatype),1,3)='set'">
                     <!-- EXTRAI O TIPO DE DADO DO CONJUNTO -->
```

```
                        <xsl:variable name="tipodado" >
                           <xsl:value-of select="substring-before(datatype, ')')" />
                        </xsl:variable>
                        <xsl:variable name="tipo" >
                           <xsl:value-of select="substring-after($tipodado, '(')" />
                        </xsl:variable>
                        <xsl:value-of select="$tipo" />
                     </xsl:when>
                     <xsl:otherwise>
                        <xsl:value-of select="normalize-space(datatype)" />
                     </xsl:otherwise>
                  </xsl:choose>
                  <xsl:text>',</xsl:text>
                  <xsl:choose>
                     <xsl:when test="string-length(normalize-space(defaultValue))>0">
                        '<xsl:value-of select="normalize-space(name)"/>',
                     </xsl:when>
                     <xsl:otherwise> null, </xsl:otherwise>
                  </xsl:choose>
                  null,
                  (SELECT Operation (OID) FROM OPERATION WHERE
                   name='<xsl:value-of select="$operationName" />'));
   </xsl:template>
</xsl:stylesheet>
```

Script em SQL

```
<!-- TIPO SET PARA DADOS NÃO TEMPORAIS -->
         CREATE TYPE predecessorSet AS
            ( predecessor OIDt,
             RefTemporalVersion
             REF(TemporalVersion))
             REF USING INTEGER
             MODE DB2SQL;

         CREATE TABLE  predecessor   OF
            predecessorSet
            (REF IS OID USER GENERATED,
            RefTemporalVersion
            WITH OPTIONS SCOPE TemporalVersion)

         CREATE SEQUENCE seq predecessor
            START WITH 1
            INCREMENT BY 1
            NO MAXVALUE
            NO CYCLE;

         CREATE TRIGGER i predecessor
         NO CASCADE BEFORE INSERT ON  predecessor
         REFERENCING NEW AS N
         FOR EACH ROW MODE DB2SQL
         SET N.OID= predecessor (NEXTVAL FOR seq predecessor )

      <!-- TIPOS ESTRUTURADOS -->
      CREATE TYPE  Schema  AS
       ( id  integer , name  VARCHAR ( 30 ) )
      REF USING INTEGER
      MODE DB2SQL;

      CREATE TYPE  Entity  AS
      ( id  integer , name  VARCHAR ( 30 ) )
      REF USING INTEGER
      MODE DB2SQL;

      CREATE TYPE  Class  AS
      ( id  integer , name  VARCHAR ( 30 ), tempVers  CHAR, specializ  VARCHAR (10),
        visibility  VARCHAR (10) , root  CHAR )
      REF USING INTEGER
      MODE DB2SQL;
```

```
CREATE TYPE  Relationship  AS
( name  VARCHAR ( 30 ), relatedClass  VARCHAR ( 30 ) )
REF USING INTEGER
MODE DB2SQL;

CREATE TYPE  Association
   UNDER TRelationship AS
( inverse  VARCHAR ( 30 ), temp  CHAR, cardinality  VARCHAR (10))
MODE DB2SQL;

CREATE TYPE  Aggregation
   UNDER TRelationship AS
( temp  CHAR, type  VARCHAR (10), cardinality  VARCHAR (10))
MODE DB2SQL;

CREATE TYPE  Inheritance
   UNDER TRelationship AS
( single  CHAR )
MODE DB2SQL;

CREATE TYPE  Extension
   UNDER TRelationship AS
( corresp  VARCHAR (10))
MODE DB2SQL;

CREATE TYPE  Attribute  AS
( name  VARCHAR ( 30 ), temp  CHAR, visibility  VARCHAR (10),
  type  VARCHAR ( 10 ), default  VARCHAR ( 5 ), static  CHAR)
REF USING INTEGER
MODE DB2SQL;

CREATE TYPE  Operation  AS
( name  VARCHAR ( 30 ), specializ  VARCHAR (10),
  visibility  VARCHAR (10), static  CHAR)
REF USING INTEGER
MODE DB2SQL;

CREATE TYPE  Parameter  AS
( name  VARCHAR ( 30 ), default  VARCHAR ( 5 ),
   type  VARCHAR ( 10 ) )
REF USING INTEGER
MODE DB2SQL;

CREATE TYPE  OIDt  AS
( value  VARCHAR( 15 ))
NOT INSTANTIABLE
REF USING INTEGER
MODE DB2SQL;

CREATE TYPE  Name  AS
( value  VARCHAR ( 20 ))
REF USING INTEGER
MODE DB2SQL;

CREATE TYPE  TemporalLabel  AS
( tTimei  TIMESTAMP, tTimef  TIMESTAMP,
  vTimei  TIMESTAMP, vTimef  TIMESTAMP)
NOT INSTANTIABLE
REF USING INTEGER
MODE DB2SQL;

CREATE TYPE  InstantAttribute  AS
( value  VARCHAR (10), tempLabel  TemporalLabel )
REF USING INTEGER
MODE DB2SQL;

CREATE TYPE  InstantRelationship  AS
( obj1  OIDt , obj2  OIDt , tempLabel  TemporalLabel )
REF USING INTEGER
MODE DB2SQL;

CREATE TYPE  TemporalAttribute  AS
( type  VARCHAR ( 10 ))
```

```
REF USING INTEGER
MODE DB2SQL;

CREATE TYPE  TemporalRelationship  AS ()
REF USING INTEGER
MODE DB2SQL;

CREATE TYPE  Object  AS
( OID  OIDt )
NOT INSTANTIABLE
REF USING INTEGER
MODE DB2SQL;

CREATE TYPE  TemporalObject
   UNDER TObject AS
( alive  CHAR)
NOT INSTANTIABLE
MODE DB2SQL;

CREATE TYPE  TemporalVersion
   UNDER TTemporalObject AS
( ascendant OIDt, descendant OIDt, successor OIDt, status  char ,
  configuration  CHAR)
NOT INSTANTIABLE
MODE DB2SQL;

CREATE TYPE  VersionedObjectControl
   UNDER TTemporalObject AS
( configurationCount  integer , currentVersion  OIDt , firstVersion  OIDt ,
  lastVersion  OIDt , nextVersionNumber  integer , userCurrentVersion  CHAR,
  versionCount  integer )
FINAL
MODE DB2SQL;

<!-- INCLUSÃO RELACIONAMENTOS PARA OS TIPOS -->
ALTER TYPE  Entity
ADD ATTRIBUTE refSchema REF (Schema);

ALTER TYPE  Class
ADD ATTRIBUTE refSchema REF (Schema);

ALTER TYPE  Relationship
ADD ATTRIBUTE refClass REF (Class);

ALTER TYPE  Attribute
ADD ATTRIBUTE refClass REF (Class);

ALTER TYPE  Operation
ADD ATTRIBUTE refClass REF (Class);

ALTER TYPE  Parameter
ADD ATTRIBUTE refOperationin REF (Operation);

ALTER TYPE  Parameter
ADD ATTRIBUTE refOperationout REF (Operation);

ALTER TYPE  Object
ADD ATTRIBUTE refNamenickNameOf REF (Name);

ALTER TYPE  InstantAttribute
ADD ATTRIBUTE property VARCHAR(30);

ALTER TYPE  instantAttribute
ADD ATTRIBUTE refTemporalAttributecurrentAttr
 REF (TemporalAttribute);

ALTER TYPE  instantAttribute
ADD ATTRIBUTE refTemporalAttribute
 REF (TemporalAttribute);

ALTER TYPE  InstantRelationship
ADD ATTRIBUTE refTemporalRelationshipcurrentRel
 REF (TemporalRelationship);
```

```
   ALTER TYPE  InstantRelationship
   ADD ATTRIBUTE refTemporalRelationship
    REF (TemporalRelationship);

   ALTER TYPE  Name
   ADD ATTRIBUTE refObjectnickname REF (Object);

<!-- TABELAS TIPADAS -->
      CREATE TABLE  Schema  OF  Schema
      (REF IS OID USER GENERATED,
       id  NOT NULL, name  NOT NULL);

      CREATE TABLE  Entity  OF  Entity
      (REF IS OID USER GENERATED,
       id  NOT NULL, name  NOT NULL );

      CREATE TABLE  Class  OF  Class
      (REF IS OID USER GENERATED,
       id  NOT NULL, name  NOT NULL, tempVers  NOT NULL,
      CONSTRAINT  tempVers
         CHECK ( tempVers IN ('T', 'F')),
      CONSTRAINT  specializ
         CHECK ( specializ  IN ('none', 'abstract', 'final') ),
      CONSTRAINT  visibility
         CHECK ( visibility  IN ('public', 'protected', 'private') ),
      CONSTRAINT  root
         CHECK ( root IN ('T', 'F')));

      CREATE TABLE  Relationship  OF  Relationship
      (REF IS OID USER GENERATED,
       relatedClass  NOT NULL);

      CREATE TABLE  Association  OF  Association
         UNDER  Relationship
         INHERIT SELECT PRIVILEGES
      (REF IS OID USER GENERATED,
       temp  NOT NULL ,
      CONSTRAINT  temp
         CHECK ( temp IN ('T', 'F')),
      cardinality  NOT NULL,
      CONSTRAINT  cardinality
         CHECK ( cardinality  IN ('0..1', '0..n', '1..1', '1..n', 'n..m') ));

      CREATE TABLE  Aggregation  OF  Aggregation
         UNDER  Relationship
         INHERIT SELECT PRIVILEGES
      (REF IS OID USER GENERATED,
       temp  NOT NULL,
      CONSTRAINT  temp
         CHECK ( temp IN ('T', 'F')),
      CONSTRAINT  type
         CHECK ( type  IN ('byValue', 'byReference') ),
      cardinality  NOT NULL,
      CONSTRAINT  cardinality
         CHECK ( cardinality  IN ('1..1', '1..n') ));

      CREATE TABLE  Inheritance  OF  Inheritance
         UNDER  Relationship
         INHERIT SELECT PRIVILEGES
      (REF IS OID USER GENERATED,
       single  NOT NULL,
      CONSTRAINT  single
         CHECK ( single IN ('T', 'F')));

      CREATE TABLE  Extension  OF  Extension
         UNDER  Relationship
         INHERIT SELECT PRIVILEGES
      (REF IS OID USER GENERATED,
       corresp  NOT NULL,
      CONSTRAINT  corresp
         CHECK ( corresp  IN ('1:1', '1:n', 'n:1', 'n:m') ));
```

```
      CREATE TABLE  Attribute  OF  Attribute
      (REF IS OID USER GENERATED,
       name  NOT NULL, temp  NOT NULL,
      CONSTRAINT  temp
         CHECK ( temp IN ('T', 'F')),
      CONSTRAINT  visibility
         CHECK ( visibility  IN ('public', 'protected', 'private') ),
      CONSTRAINT  static
         CHECK ( static IN ('T', 'F')));

      CREATE TABLE  Operation  OF  Operation
      (REF IS OID USER GENERATED,
       name  NOT NULL,
      CONSTRAINT  specializ
         CHECK ( specializ  IN ('none', 'abstract', 'final') ),
      CONSTRAINT  visibility
         CHECK ( visibility  IN ('public', 'protected', 'private') ),
      CONSTRAINT  static
         CHECK ( static IN ('T', 'F')));

      CREATE TABLE  Parameter  OF  Parameter
      (REF IS OID USER GENERATED,
       name  NOT NULL, type  NOT NULL);

      CREATE TABLE  Name  OF  Name
      (REF IS OID USER GENERATED,
       value  NOT NULL);

      CREATE TABLE  InstantAttribute  OF  InstantAttribute
      (REF IS OID USER GENERATED,
       value  NOT NULL, tempLabel  NOT NULL, property NOT NULL,
      refTemporalVersion WITH OPTIONS SCOPE TemporalVersion);

      CREATE TABLE  InstantRelationship  OF  InstantRelationship
      (REF IS OID USER GENERATED,
       obj1  NOT NULL, obj2  NOT NULL, tempLabel  NOT NULL);

      CREATE TABLE  TemporalAttribute  OF  TemporalAttribute
      (REF IS OID USER GENERATED,
       type  NOT NULL);

      CREATE TABLE  TemporalRelationship  OF  TemporalRelationship
      (REF IS OID USER GENERATED)

      CREATE TABLE  VersionedObjectControl  OF  VersionedObjectControl
         UNDER  TemporalObject
         INHERIT SELECT PRIVILEGES
      (REF IS OID USER GENERATED,
       configurationCount  NOT NULL WITH DEFAULT 0 ,
       nextVersionNumber  NOT NULL WITH DEFAULT 3 ,
       userCurrentVersion  NOT NULL WITH DEFAULT  'F',
      CONSTRAINT  userCurrentVersion
         CHECK ( userCurrentVersion IN ('T', 'F')),
      versionCount  NOT NULL WITH DEFAULT 2);

<!-- INCLUSÃO RELACIONAMENTOS PARA AS CLASSES -->
   ALTER TABLE  Entity
   ALTER  refSchema ADD SCOPE Schema;

   ALTER TABLE  Class
   ALTER  refSchema ADD SCOPE  Schema;

   ALTER TABLE  Relationship
   ALTER  refClass ADD SCOPE Class;

   ALTER TABLE  Attribute
   ALTER  refClass ADD SCOPE Class;

   ALTER TABLE  Operation
   ALTER  refClass ADD SCOPE Class;

   ALTER TABLE  Parameter
   ALTER  refOperation ADD SCOPE Operation;
```

```
   ALTER TABLE  Parameter
   ALTER  refOperation ADD SCOPE Operation;

   ALTER TABLE  Object
   ALTER  refName ADD SCOPE Name;

   ALTER TABLE  InstantAttribute
   ALTER refTemporalVersion ADD SCOPE TemporalVersion;

   ALTER TABLE  instantAttribute
   ALTER  refTemporalAttribute ADD SCOPE TemporalAttribute;

   ALTER TABLE  instantAttribute
   ALTER  refTemporalAttribute ADD SCOPE TemporalAttribute;

   ALTER TABLE  InstantRelationship
   ALTER  refTemporalRelationship ADD SCOPE TemporalRelationship;

   ALTER TABLE  InstantRelationship
   ALTER  refTemporalRelationship ADD SCOPE TemporalRelationship;

   ALTER TABLE  Name
   ALTER  refObject ADD SCOPE Object;

<!-- GERAÇÃO AUTOMATICA DOS OID'S -->
<!-- CRIA SEQUENCIAS PARA OID -->
      CREATE SEQUENCE seqSchema
      START WITH 1
      INCREMENT BY 1
      NO MAXVALUE
      NO CYCLE;

      <!-- CRIA TRIGGER PARA INSERÇÃO DO OID -->
      CREATE TRIGGER iSchema
      NO CASCADE BEFORE INSERT ON Schema
      REFERENCING NEW AS N
      FOR EACH ROW MODE DB2SQL
      BEGIN ATOMIC
         DECLARE COD INTEGER;
         SET COD = NEXTVAL FOR seqSchema;
         SET N.OID=Schema(COD);
         SET N.ID=COD;
      END

   <!-- CRIA SEQUENCIAS PARA OID -->
      CREATE SEQUENCE seqEntity
      START WITH 1
      INCREMENT BY 1
      NO MAXVALUE
      NO CYCLE;

      <!-- CRIA TRIGGER PARA INSERÇÃO DO OID -->
      CREATE TRIGGER iEntity
      NO CASCADE BEFORE INSERT ON Entity
      REFERENCING NEW AS N
      FOR EACH ROW MODE DB2SQL
      BEGIN ATOMIC
         DECLARE COD INTEGER;
         SET COD = NEXTVAL FOR seqEntity;
         SET N.OID=Entity(COD);
         SET N.ID=COD;
      END

   <!-- CRIA SEQUENCIAS PARA OID -->
      CREATE SEQUENCE seqClass
      START WITH 1
      INCREMENT BY 1
      NO MAXVALUE
      NO CYCLE;

      <!-- CRIA TRIGGER PARA INSERÇÃO DO OID -->
      CREATE TRIGGER iClass
```

```
   NO CASCADE BEFORE INSERT ON Class
   REFERENCING NEW AS N
   FOR EACH ROW MODE DB2SQL
   BEGIN ATOMIC
      DECLARE COD INTEGER;
      SET COD = NEXTVAL FOR seqClass;
      SET N.OID=Class(COD);
      SET N.ID=COD;
   END

<!-- CRIA SEQUENCIAS PARA OID -->
   CREATE SEQUENCE seqRelationship
   START WITH 1
   INCREMENT BY 1
   NO MAXVALUE
   NO CYCLE;

   <!-- CRIA TRIGGER PARA INSERÇÃO DO OID -->
   CREATE TRIGGER iRelationship
   NO CASCADE BEFORE INSERT ON Relationship
   REFERENCING NEW AS N
   FOR EACH ROW MODE DB2SQL
   BEGIN ATOMIC
      DECLARE COD INTEGER;
      SET COD = NEXTVAL FOR seqRelationship;
      SET N.OID=Relationship(COD);
   END

<!-- CRIA SEQUENCIAS PARA OID -->
   CREATE SEQUENCE seqAssociation
   START WITH 1
   INCREMENT BY 1
   NO MAXVALUE
   NO CYCLE;

   <!-- CRIA TRIGGER PARA INSERÇÃO DO OID -->
   CREATE TRIGGER iAssociation
   NO CASCADE BEFORE INSERT ON Association
   REFERENCING NEW AS N
   FOR EACH ROW MODE DB2SQL
   BEGIN ATOMIC
      DECLARE COD INTEGER;
      SET COD = NEXTVAL FOR seqAssociation;
      SET N.OID=Association(COD);
   END
<!-- CRIA SEQUENCIAS PARA OID -->
   CREATE SEQUENCE seqAggregation
   START WITH 1
   INCREMENT BY 1
   NO MAXVALUE
   NO CYCLE;

   <!-- CRIA TRIGGER PARA INSERÇÃO DO OID -->
   CREATE TRIGGER iAggregation
   NO CASCADE BEFORE INSERT ON Aggregation
   REFERENCING NEW AS N
   FOR EACH ROW MODE DB2SQL
   BEGIN ATOMIC
      DECLARE COD INTEGER;
      SET COD = NEXTVAL FOR seqAggregation;
      SET N.OID=Aggregation(COD);
   END

<!-- CRIA SEQUENCIAS PARA OID -->
   CREATE SEQUENCE seqInheritance
   START WITH 1
   INCREMENT BY 1
   NO MAXVALUE
   NO CYCLE;

   <!-- CRIA TRIGGER PARA INSERÇÃO DO OID -->
   CREATE TRIGGER iInheritance
   NO CASCADE BEFORE INSERT ON Inheritance
```

```
   REFERENCING NEW AS N
   FOR EACH ROW MODE DB2SQL
   BEGIN ATOMIC
      DECLARE COD INTEGER;
      SET COD = NEXTVAL FOR seqInheritance;
      SET N.OID=Inheritance(COD);
   END

<!-- CRIA SEQUENCIAS PARA OID -->
   CREATE SEQUENCE seqExtension
   START WITH 1
   INCREMENT BY 1
   NO MAXVALUE
   NO CYCLE;

   <!-- CRIA TRIGGER PARA INSERÇÃO DO OID -->
   CREATE TRIGGER iExtension
   NO CASCADE BEFORE INSERT ON Extension
   REFERENCING NEW AS N
   FOR EACH ROW MODE DB2SQL
   BEGIN ATOMIC
      DECLARE COD INTEGER;
      SET COD = NEXTVAL FOR seqExtension;
      SET N.OID=Extension(COD);
   END

<!-- CRIA SEQUENCIAS PARA OID -->
   CREATE SEQUENCE seqAttribute
   START WITH 1
   INCREMENT BY 1
   NO MAXVALUE
   NO CYCLE;

   <!-- CRIA TRIGGER PARA INSERÇÃO DO OID -->
   CREATE TRIGGER iAttribute
   NO CASCADE BEFORE INSERT ON Attribute
   REFERENCING NEW AS N
   FOR EACH ROW MODE DB2SQL
   BEGIN ATOMIC
      DECLARE COD INTEGER;
      SET COD = NEXTVAL FOR seqAttribute;
      SET N.OID=Attribute(COD);
   END

<!-- CRIA SEQUENCIAS PARA OID -->
   CREATE SEQUENCE seqOperation
   START WITH 1
   INCREMENT BY 1
   NO MAXVALUE
   NO CYCLE;

   <!-- CRIA TRIGGER PARA INSERÇÃO DO OID -->
   CREATE TRIGGER iOperation
   NO CASCADE BEFORE INSERT ON Operation
   REFERENCING NEW AS N
   FOR EACH ROW MODE DB2SQL
   BEGIN ATOMIC
      DECLARE COD INTEGER;
      SET COD = NEXTVAL FOR seqOperation;
      SET N.OID=Operation(COD);
   END

<!-- CRIA SEQUENCIAS PARA OID -->
   CREATE SEQUENCE seqParameter
   START WITH 1
   INCREMENT BY 1
   NO MAXVALUE
   NO CYCLE;

   <!-- CRIA TRIGGER PARA INSERÇÃO DO OID -->
   CREATE TRIGGER iParameter
   NO CASCADE BEFORE INSERT ON Parameter
   REFERENCING NEW AS N
```

```
   FOR EACH ROW MODE DB2SQL
   BEGIN ATOMIC
      DECLARE COD INTEGER;
      SET COD = NEXTVAL FOR seqParameter;
      SET N.OID=Parameter(COD);
   END

<!-- CRIA SEQUENCIAS PARA OID -->
   CREATE SEQUENCE seqName
   START WITH 1
   INCREMENT BY 1
   NO MAXVALUE
   NO CYCLE;

   <!-- CRIA TRIGGER PARA INSERÇÃO DO OID -->
   CREATE TRIGGER iName
   NO CASCADE BEFORE INSERT ON Name
   REFERENCING NEW AS N
   FOR EACH ROW MODE DB2SQL
   BEGIN ATOMIC
      DECLARE COD INTEGER;
      SET COD = NEXTVAL FOR seqName;
      SET N.OID=Name(COD);
   END

<!-- CRIA SEQUENCIAS PARA OID -->
   CREATE SEQUENCE seqInstantAttribute
   START WITH 1
   INCREMENT BY 1
   NO MAXVALUE
   NO CYCLE;

   <!-- CRIA TRIGGER PARA INSERÇÃO DO OID -->
   CREATE TRIGGER iInstantAttribute
   NO CASCADE BEFORE INSERT ON InstantAttribute
   REFERENCING NEW AS N
   FOR EACH ROW MODE DB2SQL
   BEGIN ATOMIC
      DECLARE COD INTEGER;
      SET COD = NEXTVAL FOR seqInstantAttribute;
      SET N.OID=InstantAttribute(COD);
   END

<!-- CRIA SEQUENCIAS PARA OID -->
   CREATE SEQUENCE seqInstantRelationship
   START WITH 1
   INCREMENT BY 1
   NO MAXVALUE
   NO CYCLE;

   <!-- CRIA TRIGGER PARA INSERÇÃO DO OID -->
   CREATE TRIGGER iInstantRelationship
   NO CASCADE BEFORE INSERT ON InstantRelationship
   REFERENCING NEW AS N
   FOR EACH ROW MODE DB2SQL
   BEGIN ATOMIC
      DECLARE COD INTEGER;
      SET COD = NEXTVAL FOR seqInstantRelationship;
      SET N.OID=InstantRelationship(COD);
   END

<!-- CRIA SEQUENCIAS PARA OID -->
   CREATE SEQUENCE seqTemporalAttribute
   START WITH 1
   INCREMENT BY 1
   NO MAXVALUE
   NO CYCLE;

   <!-- CRIA TRIGGER PARA INSERÇÃO DO OID -->
   CREATE TRIGGER iTemporalAttribute
   NO CASCADE BEFORE INSERT ON TemporalAttribute
   REFERENCING NEW AS N
   FOR EACH ROW MODE DB2SQL
```

```
   BEGIN ATOMIC
      DECLARE COD INTEGER;
      SET COD = NEXTVAL FOR seqTemporalAttribute;
      SET N.OID=TemporalAttribute(COD);
   END

<!-- CRIA SEQUENCIAS PARA OID -->
   CREATE SEQUENCE seqTemporalRelationship
   START WITH 1
   INCREMENT BY 1
   NO MAXVALUE
   NO CYCLE;

   <!-- CRIA TRIGGER PARA INSERÇÃO DO OID -->
   CREATE TRIGGER iTemporalRelationship
   NO CASCADE BEFORE INSERT ON TemporalRelationship
   REFERENCING NEW AS N
   FOR EACH ROW MODE DB2SQL
   BEGIN ATOMIC
      DECLARE COD INTEGER;
      SET COD = NEXTVAL FOR seqTemporalRelationship;
      SET N.OID=TemporalRelationship(COD);
   END

<!-- CRIA SEQUENCIAS PARA OID -->
   CREATE SEQUENCE seqVersionedObjectControl
   START WITH 1
   INCREMENT BY 1
   NO MAXVALUE
   NO CYCLE;

   <!-- CRIA TRIGGER PARA INSERÇÃO DO OID -->
   CREATE TRIGGER iVersionedObjectControl
   NO CASCADE BEFORE INSERT ON VersionedObjectControl
   REFERENCING NEW AS N
   FOR EACH ROW MODE DB2SQL
   BEGIN ATOMIC
      DECLARE COD INTEGER;
      SET COD = NEXTVAL FOR seqVersionedObjectControl;
      SET N.OID=VersionedObjectControl(COD);
   END

<!-- INSERIR DADOS DO MODELO NOS METADADOS -->
INSERT INTO Schema (name) VALUES (nome);

<!-- METADADOS CLASS -->
   INSERT INTO Class (name, tempVers, specializ, visibility, root, refSchema)
      VALUES ('OIDt', 'F', 'abstract', 'public', 'F',
                      (SELECT Shema(id) FROM SCHEMA
                       WHERE name='TVM'));

<!-- METADADOS ATTRIBUTE -->
INSERT INTO Attribute (name, temp, visibility, static, type, default, refClass)
   VALUES ('value', 'F', 'public', 'False', 'VARCHAR ( 15 )', ' ',
                     (SELECT Class(id) FROM CLASS
                      WHEREname='OIDt'));

<!-- METADADOS RELATIONSHIP -->
<!-- METADADOS OPERATION -->
INSERT INTO Operation (name, specializ, visibility, static, refClass)
   VALUES ('OIDt', null, 'public', null,
                    (SELECT Class(id) FROM CLASS
                     WHEREname='OIDt'));

INSERT INTO Parameter (name, datatype, default, refOperationin, refOperationout)
   VALUES ('E', 'integer', null,
                    (SELECT Operation (OID) FROM OPERATION
                     WHEREname='OIDt'),
                    null);

INSERT INTO Parameter (name, datatype, default, refOperationin, refOperationout)
   VALUES ('C', 'integer', null,
                    (SELECT Operation (OID) FROM OPERATION
```

```
                    WHERE name='OIDt'),
                   null);

INSERT INTO Parameter (name, datatype, default, refOperationin, refOperationout)
   VALUES ('V', 'integer', null,
                   (SELECT Operation (OID) FROM OPERATION
                    WHERE name='OIDt'),
                   null);

INSERT INTO Operation (name, specializ, visibility, static, refClass)
   VALUES ('getClassNr', null, 'public', null,
                   (SELECT Class(id) FROM CLASS
                     WHERE name='OIDt'));

INSERT INTO Parameter (name, datatype, default, refOperationin, refOperationout)
   VALUES ('', 'integer', null, null,
                   (SELECT Operation (OID) FROM OPERATION
                    WHERE name='getClassNr'));

INSERT INTO Operation (name, specializ, visibility, static, refClass)
   VALUES ('getEntityNr', null, 'public', null,
                   (SELECT Class(id) FROM CLASS
                    WHERE name='OIDt'));

INSERT INTO Parameter (name, datatype, default, refOperationin, refOperationout)
   VALUES ('', 'integer', null, null,
                   (SELECT Operation (OID) FROM OPERATION
                    WHERE name='getEntityNr'));

INSERT INTO Operation (name, specializ, visibility, static, refClass)
   VALUES ('getOID', null, 'public', null,
                   (SELECT Class(id) FROM CLASS
                    WHERE name='OIDt'));

INSERT INTO Parameter (name, datatype, default, refOperationin, refOperationout)
   VALUES ('', 'OIDt', null,  null,
                   (SELECT Operation (OID) FROM OPERATION
                    WHERE name='getOID'));

INSERT INTO Operation (name, specializ, visibility, static, refClass)
   VALUES ('getVersionNr', null, 'public', null,
                   (SELECT Class(id) FROM CLASS
                    WHERE name='OIDt'));

INSERT INTO Parameter (name, datatype, default, refOperationin, refOperationout)
   VALUES ('', 'integer', null,  null,
                   (SELECT Operation (OID) FROM OPERATION
                    WHEREname='getVersionNr'));

   INSERT INTO Class (name, tempVers, specializ, visibility, root, refSchema)
      VALUES ('Name', 'F', 'none', 'public', 'F',
                      (SELECT Shema(id) FROM SCHEMA
                       WHERE name='TVM'));

   <!-- METADADOS ATTRIBUTE -->
INSERT INTO Attribute (name, temp, visibility, static, type, default, refClass)
   VALUES ('value', 'F', 'public', 'False', 'VARCHAR ( 20 )','',
                   (SELECT Class(id) FROM CLASS
                    WHERE name='Name'));

<!-- METADADOS RELATIONSHIP -->
<!-- RELACIONAMENTO DE ASSOCIACAO -->
INSERT INTO Association (name, relatedClass, inverse, temp, cardinality, refClass)
      VALUES ('nickNameOf', 'Object', 'nickname',  'F', '1:1',
                      (SELECT Class(id) FROM CLASS
                       WHERE name='Name'));

<!-- METADADOS OPERATION -->
INSERT INTO Operation (name, specializ, visibility, static, refClass)
   VALUES ('Name', null, 'public', null,
                   (SELECT Class(id) FROM CLASS
                    WHERE name='Name'));
```

```
   INSERT INTO Parameter (name, datatype, default, refOperationin, refOperationout)
      VALUES ('nam', 'VARCHAR ( 20 )', null,
                     (SELECT Operation (OID) FROM OPERATION
                      WHERE name='Name'),
                     null);

   INSERT INTO Operation (name, specializ, visibility, static, refClass)
      VALUES ('getClassName', null, 'public', null,
              (SELECT Class(id) FROM CLASS WHERE
               name='Name'));

   INSERT INTO Parameter (name, datatype, default, refOperationin, refOperationout)
      VALUES ('', 'VARCHAR ( 30 )', null,  null,
                     (SELECT Operation (OID) FROM OPERATION
                      WHERE name='getClassName'));

   INSERT INTO Operation (name, specializ, visibility, static, refClass)
      VALUES ('getName', null, 'public', null,
                     (SELECT Class(id) FROM CLASS
                      WHERE name='Name'));

   INSERT INTO Parameter (name, datatype, default, refOperationin, refOperationout)
      VALUES ('', 'VARCHAR ( 30 )', null, null,
                     (SELECT Operation (OID) FROM OPERATION
                      WHERE name='getName'));

      INSERT INTO Class (name, tempVers, specializ, visibility, root, refSchema)
         VALUES ('TemporalLabel', 'F', 'abstract', 'public', 'F',
                        (SELECT Shema(id) FROM SCHEMA
                         WHERE name='TVM'));

   <!-- METADADOS ATTRIBUTE -->
   INSERT INTO Attribute (name, temp, visibility, static, type, default, refClass)
      VALUES ('tTimei', 'F', 'public', 'False', 'TIMESTAMP','',
                     (SELECT Class(id) FROM CLASS
                      WHERE name='TemporalLabel'));

   INSERT INTO Attribute (name, temp, visibility, static, type, default, refClass)
      VALUES ('tTimef', 'F', 'public', 'False', 'TIMESTAMP','',
                     (SELECT Class(id) FROM CLASS
                      WHERE name='TemporalLabel'));

   INSERT INTO Attribute (name, temp, visibility, static, type, default, refClass)
      VALUES ('vTimei', 'F', 'public', 'False', 'TIMESTAMP','',
                     (SELECT Class(id) FROM CLASS
                      WHERE name='TemporalLabel'));

   INSERT INTO Attribute (name, temp, visibility, static, type, default, refClass)
      VALUES ('vTimef', 'F', 'public', 'False', 'TIMESTAMP','',
                     (SELECT Class(id) FROM CLASS
                      WHERE name='TemporalLabel'));

<!-- METADADOS RELATIONSHIP --><!-- METADADOS OPERATION -->
   INSERT INTO Operation (name, specializ, visibility, static, refClass)
      VALUES ('TemporalLabel', null, 'public', null,
                     (SELECT Class(id) FROM CLASS
                      WHERE name='TemporalLabel'));

   INSERT INTO Parameter (name, datatype, default, refOperationin, refOperationout)
      VALUES ('iValidTime', 'TIMESTAMP', null,
                     (SELECT Operation (OID) FROM OPERATION
                      WHERE name='TemporalLabel'),
                     null);

   INSERT INTO Parameter (name, datatype, default, refOperationin, refOperationout)
      VALUES ('fValidTime', 'TIMESTAMP', null,
                     (SELECT Operation (OID) FROM OPERATION
                      WHERE name='TemporalLabel'),
                     null);

   INSERT INTO Parameter (name, datatype, default, refOperationin, refOperationout)
      VALUES ('iTransTime', 'TIMESTAMP', null,
                     (SELECT Operation (OID) FROM OPERATION
```

```
                  WHERE name='TemporalLabel'),
                 null);

INSERT INTO Parameter (name, datatype, default, refOperationin, refOperationout)
   VALUES ('fTransTime', 'TIMESTAMP', null,
                 (SELECT Operation (OID) FROM OPERATION
                  WHERE name='TemporalLabel'),
                  null);

INSERT INTO Operation (name, specializ, visibility, static, refClass)
   VALUES ('getTTimei', null, 'public', null,
           (SELECT Class(id) FROM CLASS WHERE
            name='TemporalLabel'));

INSERT INTO Parameter (name, datatype, default, refOperationin, refOperationout)
   VALUES ('', 'TIMESTAMP', null, null,
                 (SELECT Operation (OID) FROM OPERATION
                  WHERE name='getTTimei'));

INSERT INTO Operation (name, specializ, visibility, static, refClass)
   VALUES ('getTTimef', null, 'public', null,
                 (SELECT Class(id) FROM CLASS
                  WHERE name='TemporalLabel'));

INSERT INTO Parameter (name, datatype, default, refOperationin, refOperationout)
   VALUES ('', 'TIMESTAMP', null, null,
                 (SELECT Operation (OID) FROM OPERATION
                  WHERE name='getTTimef'));

INSERT INTO Operation (name, specializ, visibility, static, refClass)
   VALUES ('getVTimei', null, 'public', null,
                 (SELECT Class(id) FROM CLASS
                  WHERE name='TemporalLabel'));

INSERT INTO Parameter (name, datatype, default, refOperationin, refOperationout)
   VALUES ('', 'TIMESTAMP', null, null,
                 (SELECT Operation (OID) FROM OPERATION
                  WHERE name='getVTimei'));

INSERT INTO Operation (name, specializ, visibility, static, refClass)
   VALUES ('getVTimef', null, 'public', null,
                 (SELECT Class(id) FROM CLASS
                  WHERE name='TemporalLabel'));

INSERT INTO Parameter (name, datatype, default, refOperationin, refOperationout)
   VALUES ('', 'TIMESTAMP', null, null,
                 (SELECT Operation (OID) FROM OPERATION
                  WHERE name='getVTimef'));

INSERT INTO Operation (name, specializ, visibility, static, refClass)
   VALUES ('setTTimef', null, 'public', null,
                 (SELECT Class(id) FROM CLASS
                  WHERE name='TemporalLabel'));

INSERT INTO Parameter (name, datatype, default, refOperationin, refOperationout)
   VALUES ('i', 'TIMESTAMP', null,
                 (SELECT Operation (OID) FROM OPERATION
                  WHERE name='setTTimef'),
                 null);

INSERT INTO Operation (name, specializ, visibility, static, refClass)
   VALUES ('setVTimef', null, 'public', null,
                 (SELECT Class(id) FROM CLASS
                  WHERE name='TemporalLabel'));

INSERT INTO Parameter (name, datatype, default, refOperationin, refOperationout)
   VALUES ('i', 'TIMESTAMP', null,
                 (SELECT Operation (OID) FROM OPERATION
                  WHERE name='setVTimef'),
                 null);

INSERT INTO Class (name, tempVers, specializ, visibility, root, refSchema)
      VALUES ('InstantAttribute', 'F', 'none', 'public', 'F',
```

```
                      (SELECT Shema(id) FROM SCHEMA
                       WHERE name='TVM'));

 <!-- METADADOS ATTRIBUTE -->
 INSERT INTO Attribute (name, temp, visibility, static, type, default, refClass)
    VALUES ('value', 'F', 'public', 'False', 'VARCHAR (10)','',
                     (SELECT Class(id) FROM CLASS
                      WHERE name='InstantAttribute'));

 INSERT INTO Attribute (name, temp, visibility, static, type, default, refClass)
    VALUES ('tempLabel', 'F', 'public', 'False', 'TemporalLabel','',
                     (SELECT Class(id) FROM CLASS
                      WHERE name='InstantAttribute'));

<!-- METADADOS RELATIONSHIP -->
<!-- METADADOS OPERATION -->
 INSERT INTO Operation (name, specializ, visibility, static, refClass)
    VALUES ('InstantAttribute', null, 'public', null,
                     (SELECT Class(id) FROM CLASS
                      WHERE name='InstantAttribute'));

 INSERT INTO Parameter (name, datatype, default, refOperationin,
                        refOperationout)
    VALUES ('val', 'VARCHAR (10)', null,
                     (SELECT Operation (OID) FROM OPERATION
                      WHERE name='InstantAttribute'),
                      null);

 INSERT INTO Operation (name, specializ, visibility, static, refClass)
    VALUES ('InstantAttribute', null, 'public', null,
                     (SELECT Class(id) FROM CLASS
                      WHERE name='InstantAttribute'));

 INSERT INTO Parameter (name, datatype, default, refOperationin,
                        refOperationout)
    VALUES ('val','VARCHAR (10)', null,
                     (SELECT Operation (OID) FROM OPERATION
                      WHERE name='InstantAttribute'),
                     null);

 INSERT INTO Parameter (name, datatype, default, refOperationin,
                        refOperationout)
    VALUES ('iValidTime', 'TIMESTAMP', null,
                     (SELECT Operation (OID) FROM OPERATION
                      WHERE name='InstantAttribute'),
                     null);

 INSERT INTO Parameter (name, datatype, default, refOperationin,
                        refOperationout)
    VALUES ('fValidTime', 'TIMESTAMP', null,
                     (SELECT Operation (OID) FROM OPERATION
                      WHERE name='InstantAttribute'),
                     null);

 INSERT INTO Operation (name, specializ, visibility, static, refClass)
    VALUES ('getValue', null, 'public', null,
                     (SELECT Class(id) FROM CLASS
                      WHERE name='InstantAttribute'));

 INSERT INTO Parameter (name, datatype, default, refOperationin,
                        refOperationout)
    VALUES ('', 'VARCHAR (10)', null, null,
                     (SELECT Operation (OID) FROM OPERATION
                      WHERE name='getValue'));

 INSERT INTO Operation (name, specializ, visibility, static, refClass)
    VALUES ('getTempLabel', null, 'public', null,
                      (SELECT Class(id) FROM CLASS
                       WHERE name='InstantAttribute'));

 INSERT INTO Parameter (name, datatype, default, refOperationin,
                        refOperationout)
    VALUES ('', 'TemporalLabel', null, null,
```

```
                    (SELECT Operation (OID) FROM OPERATION
                     WHERE name='getTempLabel'));

 INSERT INTO Operation (name, specializ, visibility, static, refClass)
    VALUES ('getTempLabelOf', null, 'public', null,
                    (SELECT Class(id) FROM CLASS
                     WHERE name='InstantAttribute'));

 INSERT INTO Parameter (name, datatype, default, refOperationin,
                        refOperationout)
    VALUES ('val', 'VARCHAR (10)', null,
                    (SELECT Operation (OID) FROM OPERATION
                     WHERE name='getTempLabelOf'),
                    null);

 INSERT INTO Parameter (name, datatype, default, refOperationin,
                        refOperationout)
    VALUES ('', 'TemporalLabel', null, null,
                    (SELECT Operation (OID) FROM OPERATION
                     WHERE name='getTempLabelOf'));

 INSERT INTO Operation (name, specializ, visibility, static, refClass)
    VALUES ('setFTransactionTime', null,  'public', null,
                    (SELECT Class(id) FROM CLASS
                     WHERE name='InstantAttribute'));

 INSERT INTO Parameter (name, datatype, default, refOperationin,
                        refOperationout)
    VALUES ('i', 'TIMESTAMP', null,
                    (SELECT Operation (OID) FROM OPERATION
                     WHERE name='setFTransactionTime'),
                    null);

 INSERT INTO Operation (name, specializ, visibility, static, refClass)
    VALUES ('setFValidTime', null, 'public', null,
                    (SELECT Class(id) FROM CLASS
                     WHERE name='InstantAttribute'));

 INSERT INTO Parameter (name, datatype, default, refOperationin,
                        refOperationout)
    VALUES ('i', 'TIMESTAMP', null,
                    (SELECT Operation (OID) FROM OPERATION
                     WHERE name='setFValidTime'),
                    null);

    INSERT INTO Class (name, tempVers, specializ, visibility,
                       root, refSchema)
       VALUES ('InstantRelationship', 'F', 'none', 'public', 'F',
                        (SELECT Shema(id) FROM SCHEMA
                         WHERE name='TVM'));

 <!-- METADADOS ATTRIBUTE -->
 INSERT INTO Attribute (name, temp, visibility, static, type, default, refClass)
    VALUES ('obj1', 'F', 'public', 'False', 'OIDt','',
                    (SELECT Class(id) FROM CLASS
                     WHERE name='InstantRelationship'));

 INSERT INTO Attribute (name, temp, visibility, static, type, default, refClass)
    VALUES ('obj2', 'F', 'public', 'False', 'OIDt','',
                   (SELECT Class(id) FROM CLASS
                    WHERE name='InstantRelationship'));

 INSERT INTO Attribute (name, temp, visibility, static, type, default, refClass)
    VALUES ('tempLabel', 'F', 'public', 'False', 'TemporalLabel','',
                    (SELECT Class(id) FROM CLASS
                     WHERE name='InstantRelationship'));

<!-- METADADOS RELATIONSHIP -->
<!-- METADADOS OPERATION -->
 INSERT INTO Operation (name, specializ, visibility, static, refClass)
    VALUES ('InstantRelationship', null, 'public', null,
                   (SELECT Class(id) FROM CLASS
                    WHERE name='InstantRelationship'));
```

```
INSERT INTO Parameter (name, datatype, default, refOperationin,
                       refOperationout)
   VALUES ('o1', 'OIDt', null,
                   (SELECT Operation (OID) FROM OPERATION
                    WHERE name='InstantRelationship'),
                   null);

INSERT INTO Parameter (name, datatype, default, refOperationin,
                       refOperationout)
   VALUES ('o2', 'OIDt', null,
                   (SELECT Operation (OID) FROM OPERATION
                    WHERE name='InstantRelationship'),
                   null);

INSERT INTO Operation (name, specializ, visibility, static, refClass)
   VALUES ('InstantRelationship', null, 'public', null,
                  (SELECT Class(id) FROM CLASS
                   WHERE name='InstantRelationship'));

INSERT INTO Parameter (name, datatype, default, refOperationin,
                       refOperationout)
   VALUES ('o1', 'OIDt', null,
                   (SELECT Operation (OID) FROM OPERATION
                    WHERE name='InstantRelationship'),
                   null);

INSERT INTO Parameter (name, datatype, default, refOperationin,
                       refOperationout)
   VALUES ('o2', 'OIDt', null,
                    (SELECT Operation (OID) FROM OPERATION
                     WHERE name='InstantRelationship'),
                    null);

INSERT INTO Parameter (name, datatype, default, refOperationin,
                       refOperationout)
   VALUES ('iValidTime', 'TIMESTAMP', null,
                   (SELECT Operation (OID) FROM OPERATION
                    WHERE name='InstantRelationship'),
                   null);

INSERT INTO Parameter (name, datatype, default, refOperationin,
                       refOperationout)
   VALUES ('fValidTime', 'TIMESTAMP', null,
                   (SELECT Operation (OID) FROM OPERATION
                    WHERE name='InstantRelationship'),
                   null);

INSERT INTO Operation (name, specializ, visibility, static, refClass)
   VALUES ('getObj1', null, 'public', null,
                   (SELECT Class(id) FROM CLASS
                    WHERE name='InstantRelationship'));

INSERT INTO Parameter (name, datatype, default, refOperationin,
                       refOperationout)
   VALUES ('', 'OIDt', null, null,
                    (SELECT Operation (OID) FROM OPERATION
                     WHERE name='getObj1'));

INSERT INTO Operation (name, specializ, visibility, static, refClass)
   VALUES ('getObj2', null, 'public', null,
                   (SELECT Class(id) FROM CLASS
                    WHERE name='InstantRelationship'));

INSERT INTO Parameter (name, datatype, default, refOperationin,
                       refOperationout)
   VALUES ('', 'OIDt', null,  null,
                    (SELECT Operation (OID) FROM OPERATION
                      WHERE name='getObj2'));

INSERT INTO Operation (name, specializ, visibility, static, refClass)
   VALUES ('getTempLabel', null, 'public', null,
                   (SELECT Class(id) FROM CLASS
```

```
                    WHERE name='InstantRelationship'));

INSERT INTO Parameter (name, datatype, default, refOperationin,
                       refOperationout)
   VALUES ('', 'TemporalLabel', null,  null,
                   (SELECT Operation (OID) FROM OPERATION
                    WHERE name='getTempLabel'));

INSERT INTO Operation (name, specializ, visibility, static, refClass)
   VALUES ('getTempLabelOf', null, 'public', null,
                   (SELECT Class(id) FROM CLASS
                    WHERE name='InstantRelationship'));

INSERT INTO Parameter (name, datatype, default, refOperationin,
                       refOperationout)
   VALUES ('o1', 'OIDt', null,
                  (SELECT Operation (OID) FROM OPERATION
                   WHERE name='getTempLabelOf'),
                  null);

INSERT INTO Parameter (name, datatype, default, refOperationin,
                       refOperationout)
   VALUES ('o2', 'OIDt', null,
                   (SELECT Operation (OID) FROM OPERATION
                    WHERE name='getTempLabelOf'),
                   null);

INSERT INTO Parameter (name, datatype, default, refOperationin,
                       refOperationout)
   VALUES ('', 'TemporalLabel', null, null,
                   (SELECT Operation (OID) FROM OPERATION
                    WHERE name='getTempLabelOf'));

INSERT INTO Operation (name, specializ, visibility, static, refClass)
   VALUES ('setFTransactionTime', null, 'public', null,
                   (SELECT Class(id) FROM CLASS
                    WHERE name='InstantRelationship'));

INSERT INTO Parameter (name, datatype, default, refOperationin,
                       refOperationout)
   VALUES ('i', 'TIMESTAMP', null,
                   (SELECT Operation (OID) FROM OPERATION
                    WHERE name='setFTransactionTime'),
                   null);

INSERT INTO Operation (name, specializ, visibility, static, refClass)
   VALUES ('setFValidTime', null, 'public', null,
                   (SELECT Class(id) FROM CLASS
                    WHERE name='InstantRelationship'));

INSERT INTO Parameter (name, datatype, default, refOperationin,
                       refOperationout)
   VALUES ('i', 'TIMESTAMP', null,
                   (SELECT Operation (OID) FROM OPERATION
                    WHERE name='setFValidTime'),
                   null);

 INSERT INTO Class (name, tempVers, specializ, visibility,
                      root, refSchema)
      VALUES ('TemporalAttribute', 'F', 'none', 'public', 'F',
                       (SELECT Shema(id) FROM SCHEMA
                        WHERE name='TVM'));

<!-- METADADOS ATTRIBUTE -->
INSERT INTO Attribute (name, temp, visibility, static, type, default, refClass)
   VALUES ('type', 'F', 'public', 'F', 'VARCHAR (10)','',
                   (SELECT Class(id) FROM CLASS
                    WHERE name='TemporalAttribute'));

  <!-- METADADOS RELATIONSHIP -->
  <!-- RELACIONAMENTO DE ASSOCIACAO -->
      INSERT INTO Association (name, relatedClass, inverse, temp,
                               cardinality, refClass)
```

```
        VALUES ('currentAttr', 'instantAttribute', null, 'F', '1:1',
                        (SELECT Class(id) FROM CLASS
                         WHERE name='TemporalAttribute'));

 <!-- RELACIONAMENTO DE AGREGACAO -->
      INSERT INTO Aggregation (name, relatedClass, temp, cardinality,
                               refClass)
        VALUES (null, 'instantAttribute', 'F', 'n',
                        (SELECT Class(id) FROM CLASS
                         WHERE name='TemporalAttribute'));

 <!-- METADADOS OPERATION -->
INSERT INTO Operation (name, specializ, visibility, static, refClass)
   VALUES ('TemporalAttribute', null, 'public', null,
                  (SELECT Class(id) FROM CLASS
                   WHERE name='TemporalAttribute'));

INSERT INTO Parameter (name, datatype, default, refOperationin,
                       refOperationout)
   VALUES ('typ', 'VARCHAR (10)', null,
                  (SELECT Operation (OID) FROM OPERATION
                   WHERE name='TemporalAttribute'),
                 null);

INSERT INTO Parameter (name, datatype, default, refOperationin,
                       refOperationout)
   VALUES ('val', 'VARCHAR (10)', null,
                  (SELECT Operation (OID) FROM OPERATION
                   WHERE name='TemporalAttribute'),
                 null);

INSERT INTO Operation (name, specializ, visibility, static, refClass)
   VALUES ('TemporalAttribute', null, 'public', null,
                   (SELECT Class(id) FROM CLASS
                    WHERE name='TemporalAttribute'));

INSERT INTO Parameter (name, datatype, default, refOperationin,
                       refOperationout)
   VALUES ('typ', 'VARCHAR (10)', null,
                   (SELECT Operation (OID) FROM OPERATION
                    WHERE name='TemporalAttribute'),
                   null);

INSERT INTO Parameter (name, datatype, default, refOperationin,
                       refOperationout)
   VALUES ('val', 'VARCHAR (10)', null,
                   (SELECT Operation (OID) FROM OPERATION
                    WHERE name='TemporalAttribute'),
                   null);

INSERT INTO Parameter (name, datatype, default, refOperationin,
                       refOperationout)
   VALUES ('iValidTime', 'TIMESTAMP', null,
                   (SELECT Operation (OID) FROM OPERATION
                    WHERE name='TemporalAttribute'),
                   null);

INSERT INTO Parameter (name, datatype, default, refOperationin,
                       refOperationout)
   VALUES ('fValidTime', 'TIMESTAMP', null,
                   (SELECT Operation (OID) FROM OPERATION
                    WHERE name='TemporalAttribute'),
                   null);

INSERT INTO Operation (name, specializ, visibility, static, refClass)
   VALUES ('getCurrent', null, 'public', null,
                   (SELECT Class(id) FROM CLASS
                    WHERE name='TemporalAttribute'));

INSERT INTO Parameter (name, datatype, default, refOperationin,
                       refOperationout)
   VALUES ('', 'InstantAttribute', null,  null,
                   (SELECT Operation (OID) FROM OPERATION
```

```
                    WHERE name='getCurrent'));

INSERT INTO Operation (name, specializ, visibility, static, refClass)
   VALUES ('getHistory', null, 'public', null,
                   (SELECT Class(id) FROM CLASS
                    WHERE name='TemporalAttribute'));

INSERT INTO Parameter (name, datatype, default, refOperationin,
                       refOperationout)
   VALUES ('', 'InstantAttribute', null, null,
                   (SELECT Operation (OID) FROM OPERATION
                    WHERE name='getHistory'));

INSERT INTO Operation (name, specializ, visibility, static, refClass)
   VALUES ('getHistoryOfValue', null, 'public', null,
                   (SELECT Class(id) FROM CLASS
                    WHERE name='TemporalAttribute'));

INSERT INTO Parameter (name, datatype, default, refOperationin,
                       refOperationout)
   VALUES ('val', 'VARCHAR (10)', null,
                   (SELECT Operation (OID) FROM OPERATION
                    WHERE name='getHistoryOfValue'),
                   null);

INSERT INTO Parameter (name, datatype, default, refOperationin,
                       refOperationout)
   VALUES ('', 'TemporalLabel', null, null,
                   (SELECT Operation (OID) FROM OPERATION
                    WHERE name='getHistoryOfValue'));

INSERT INTO Operation (name, specializ, visibility, static, refClass)
   VALUES ('getValueAt', null, 'public', null,
                   (SELECT Class(id) FROM CLASS
                    WHERE name='TemporalAttribute'));

INSERT INTO Parameter (name, datatype, default, refOperationin,
                       refOperationout)
   VALUES ('at', 'TIMESTAMP', null,
                   (SELECT Operation (OID) FROM OPERATION
                    WHERE name='getValueAt'),
                   null);

INSERT INTO Parameter (name, datatype, default, refOperationin,
                       refOperationout)
   VALUES ('', 'instantAttribute', null, null,
                   (SELECT Operation (OID) FROM OPERATION
                    WHERE name='getValueAt'));


INSERT INTO Operation (name, specializ, visibility, static, refClass)
   VALUES ('setFTransactionTime', null, 'private', null,
                   (SELECT Class(id) FROM CLASS
                    WHERE name='TemporalAttribute'));

INSERT INTO Parameter (name, datatype, default, refOperationin,
                       refOperationout)
   VALUES ('validAt', 'TIMESTAMP', null,
                   (SELECT Operation (OID) FROM OPERATION
                    WHERE name='setFTransactionTime'),
                   null);

INSERT INTO Parameter (name, datatype, default, refOperationin,
                       refOperationout)
   VALUES ('fTransTime', 'TIMESTAMP', null,
                   (SELECT Operation (OID) FROM OPERATION
                    WHERE name='setFTransactionTime'),
                   null);

INSERT INTO Operation (name, specializ, visibility, static, refClass)
   VALUES ('setFValidTime', null, 'public', null,
                   (SELECT Class(id) FROM CLASS
                    WHERE name='TemporalAttribute'));
```

```
 INSERT INTO Parameter (name, datatype, default, refOperationin,
                        refOperationout)
    VALUES (’validAt’, ’TIMESTAMP’, null,
                     (SELECT Operation (OID) FROM OPERATION
                      WHERE name=’setFValidTime’),
                     null);

 INSERT INTO Parameter (name, datatype, default, refOperationin,
                        refOperationout)
    VALUES (’fvalidTime’, ’TIMESTAMP’, null,
                     (SELECT Operation (OID) FROM OPERATION
                      WHERE name=’setFValidTime’),
                     null);

 INSERT INTO Operation (name, specializ, visibility, static, refClass)
    VALUES (’updateValue’, null, ’public’, null,
                     (SELECT Class(id) FROM CLASS
                      WHERE name=’TemporalAttribute’));

 INSERT INTO Parameter (name, datatype, default, refOperationin,
                        refOperationout)
    VALUES (’at’, ’TIMESTAMP’, null,
                     (SELECT Operation (OID) FROM OPERATION
                      WHERE name=’updateValue’),
                     null);

 INSERT INTO Parameter (name, datatype, default, refOperationin,
                        refOperationout)
    VALUES (’newVal’, ’VARCHAR (10)’, null,
                     (SELECT Operation (OID) FROM OPERATION
                      WHERE name=’updateValue’),
                     null);

 INSERT INTO Parameter (name, datatype, default, refOperationin,
                        refOperationout)
    VALUES (’ivalidTime’, ’TIMESTAMP’, null,
                     (SELECT Operation (OID) FROM OPERATION
                      WHERE name=’updateValue’),
                     null);

 INSERT INTO Parameter (name, datatype, default, refOperationin,
                        refOperationout)
    VALUES (’fvalidTime’, ’TIMESTAMP’, null,
                     (SELECT Operation (OID) FROM OPERATION
                      WHERE name=’updateValue’),
                     null);
INSERT INTO Class (name, tempVers, specializ, visibility,
                       root, refSchema)
    VALUES (’TemporalRelationship’, ’F’, ’none’, ’public’, ’F’,
                        (SELECT Shema(id) FROM SCHEMA
                         WHERE name=’TVM’));

<!-- METADADOS ATTRIBUTE -->
 <!-- METADADOS RELATIONSHIP -->
<!-- RELACIONAMENTO DE ASSOCIACAO -->
INSERT INTO Association (name, relatedClass, inverse, temp,
                                cardinality, refClass)
   VALUES (’currentRel’, ’InstantRelationship’, null, ’F’, ’1:1’,
                    (SELECT Class(id) FROM CLASS
                     WHERE name=’TemporalRelationship’));

<!-- RELACIONAMENTO DE AGREGACAO -->
INSERT INTO Aggregation (name, relatedClass, temp, cardinality,
                                refClass)
   VALUES (null, ’InstantRelationship’, ’F’, ’n’,
                   (SELECT Class(id) FROM CLASS
                    WHERE name=’TemporalRelationship’));

 <!-- METADADOS OPERATION -->
 INSERT INTO Operation (name, specializ, visibility, static, refClass)
    VALUES (’TemporalRelationship’, null, ’public’, null,
                     (SELECT Class(id) FROM CLASS
```

```
                    WHERE name='TemporalRelationship'));

INSERT INTO Parameter (name, datatype, default, refOperationin,
                       refOperationout)
   VALUES ('o1', 'OIDt', null,
                   (SELECT Operation (OID) FROM OPERATION
                    WHERE name='TemporalRelationship'),
                   null);

INSERT INTO Parameter (name, datatype, default, refOperationin,
                       refOperationout)
   VALUES ('o2', 'OIDt', null,
                   (SELECT Operation (OID) FROM OPERATION
                    WHERE name='TemporalRelationship'),
                   null);

INSERT INTO Operation (name, specializ, visibility, static, refClass)
   VALUES ('TemporalRelationship', null, 'public', null,
                   (SELECT Class(id) FROM CLASS
                    WHERE name='TemporalRelationship'));

INSERT INTO Parameter (name, datatype, default, refOperationin,
                       refOperationout)
   VALUES ('o1', 'OIDt', null,
                   (SELECT Operation (OID) FROM OPERATION
                    WHERE name='TemporalRelationship'),
                   null);

INSERT INTO Parameter (name, datatype, default, refOperationin,
                       refOperationout)
   VALUES ('o2', 'OIDt', null,
                   (SELECT Operation (OID) FROM OPERATION
                    WHERE name='TemporalRelationship'),
                   null);

INSERT INTO Parameter (name, datatype, default, refOperationin,
                       refOperationout)
   VALUES ('iValidTime', 'TIMESTAMP', null,
                   (SELECT Operation (OID) FROM OPERATION
                    WHERE name='TemporalRelationship'),
                   null);

INSERT INTO Parameter (name, datatype, default, refOperationin,
                       refOperationout)
   VALUES ('fValidTime', 'TIMESTAMP', null,
                   (SELECT Operation (OID) FROM OPERATION
                    WHERE name='TemporalRelationship'),
                   null);

INSERT INTO Operation (name, specializ, visibility, static, refClass)
   VALUES ('getCurrent', null, 'public', null,
                   (SELECT Class(id) FROM CLASS
                    WHERE name='TemporalRelationship'));

INSERT INTO Parameter (name, datatype, default, refOperationin,
                       refOperationout)
   VALUES ('', 'InstantRelationship', null, null,
                   (SELECT Operation (OID) FROM OPERATION
                    WHERE name='getCurrent'));

INSERT INTO Operation (name, specializ, visibility, static, refClass)
   VALUES ('getHistory', null, 'public', null,
                   (SELECT Class(id) FROM CLASS
                    WHERE name='TemporalRelationship'));

INSERT INTO Parameter (name, datatype, default, refOperationin,
                       refOperationout)
   VALUES ('', 'TnstantRelationship', null, null,
                   (SELECT Operation (OID) FROM OPERATION
                    WHERE name='getHistory'));

INSERT INTO Operation (name, specializ, visibility, static, refClass)
   VALUES ('getHistoryOfValue', null, 'public', null,
```

```
                    (SELECT Class(id) FROM CLASS
                     WHERE name='TemporalRelationship'));

INSERT INTO Parameter (name, datatype, default, refOperationin,
                       refOperationout)
   VALUES ('o1', 'OIDt', null,
                    (SELECT Operation (OID) FROM OPERATION
                     WHERE name='getHistoryOfValue'),
                    null);

INSERT INTO Parameter (name, datatype, default, refOperationin,
                       refOperationout)
   VALUES ('o2', 'OIDt', null,
                    (SELECT Operation (OID) FROM OPERATION
                     WHERE name='getHistoryOfValue'),
                    null);

INSERT INTO Parameter (name, datatype, default, refOperationin,
                       refOperationout)
   VALUES ('', 'TemporalLabel', null, null,
                    (SELECT Operation (OID) FROM OPERATION
                     WHERE name='getHistoryOfValue'));

INSERT INTO Operation (name, specializ, visibility, static, refClass)
   VALUES ('getValueAt', null, 'public', null,
                    (SELECT Class(id) FROM CLASS
                     WHERE name='TemporalRelationship'));

INSERT INTO Parameter (name, datatype, default, refOperationin,
                       refOperationout)
   VALUES ('at', 'TIMESTAMP', null,
                    (SELECT Operation (OID) FROM OPERATION
                     WHERE name='getValueAt'),
                    null);

INSERT INTO Parameter (name, datatype, default, refOperationin,
                       refOperationout)
   VALUES ('', 'InstantRelationship', null, null,
                    (SELECT Operation (OID) FROM OPERATION
                     WHERE name='getValueAt'));

INSERT INTO Operation (name, specializ, visibility, static, refClass)
   VALUES ('setFTransactionTime', null, 'private', null,
                    (SELECT Class(id) FROM CLASS
                     WHERE name='TemporalRelationship'));

INSERT INTO Parameter (name, datatype, default, refOperationin,
                       refOperationout)
   VALUES ('validAt', 'TIMESTAMP', null,
                    (SELECT Operation (OID) FROM OPERATION
                     WHERE name='setFTransactionTime'),
                    null);

INSERT INTO Parameter (name, datatype, default, refOperationin,
                       refOperationout)
   VALUES ('fTransTime', 'TIMESTAMP', null,
                    (SELECT Operation (OID) FROM OPERATION
                     WHERE name='setFTransactionTime'),
                    null);

INSERT INTO Operation (name, specializ, visibility, static, refClass)
   VALUES ('setFValidTime', null, 'private', null,
                    (SELECT Class(id) FROM CLASS
                     WHERE name='TemporalRelationship'));

INSERT INTO Parameter (name, datatype, default, refOperationin,
                       refOperationout)
   VALUES ('validAt', 'TIMESTAMP', null,
                    (SELECT Operation (OID) FROM OPERATION
                     WHERE name='setFValidTime'),
                    null);

INSERT INTO Parameter (name, datatype, default, refOperationin,
```

```
                          refOperationout)
      VALUES ('fValidTime', 'TIMESTAMP', null,
                         (SELECT Operation (OID) FROM OPERATION
                          WHERE name='setFValidTime'),
                         null);

   INSERT INTO Operation (name, specializ, visibility, static, refClass)
      VALUES ('updateValue', null, 'public', null,
                         (SELECT Class(id) FROM CLASS
                          WHERE name='TemporalRelationship'));

   INSERT INTO Parameter (name, datatype, default, refOperationin,
                          refOperationout)
      VALUES ('at', 'TIMESTAMP', null,
                         (SELECT Operation (OID) FROM OPERATION
                          WHERE name='updateValue'),
                         null);

   INSERT INTO Parameter (name, datatype, default, refOperationin,
                          refOperationout)
      VALUES ('newO1', 'OIDt', null,
                         (SELECT Operation (OID) FROM OPERATION
                          WHERE name='updateValue'),
                         null);

   INSERT INTO Parameter (name, datatype, default, refOperationin,
                          refOperationout)
      VALUES ('NewO2', 'OIDt', null,
                         (SELECT Operation (OID) FROM OPERATION
                          WHERE name='updateValue'),
                         null);

   INSERT INTO Parameter (name, datatype, default, refOperationin,
                          refOperationout)
      VALUES ('iValidTime', 'TIMESTAMP', null,
                         (SELECT Operation (OID) FROM OPERATION
                          WHERE name='updateValue'),
                         null);

   INSERT INTO Parameter (name, datatype, default, refOperationin,
                          refOperationout)
      VALUES ('fValidTime', 'TIMESTAMP', null,
                         (SELECT Operation (OID) FROM OPERATION
                          WHERE name='updateValue'),
                         null);

INSERT INTO Class (name, tempVers, specializ, visibility,
                          root, refSchema)
         VALUES ('Object', 'F', 'abstract', 'public', 'T',
                           (SELECT Shema(id) FROM SCHEMA
                            WHERE name='TVM'));

   <!-- METADADOS ATTRIBUTE -->
   INSERT INTO Attribute (name, temp, visibility, static, type, default, refClass)
      VALUES ('OID', 'F', 'public', 'F', 'OIDt','',
                         (SELECT Class(id) FROM CLASS
                          WHERE name='Object'));

 <!-- METADADOS RELATIONSHIP -->
 <!-- RELACIONAMENTO DE ASSOCIACAO -->
  INSERT INTO Association (name, relatedClass, inverse, temp,
                                 cardinality, refClass)
      VALUES ('nickname', 'Name', 'nicknameOf', 'F', '0:n',
                         (SELECT Class(id) FROM CLASS
                           WHERE name='Object'));

   <!-- METADADOS OPERATION -->
   INSERT INTO Operation (name, specializ, visibility, static, refClass)
      VALUES ('Object', null, 'public', null,
                         (SELECT Class(id) FROM CLASS
                          WHERE name='Object'));

   INSERT INTO Operation (name, specializ, visibility, static, refClass)
```

```
   VALUES ('Object', null, 'public', null,
                   (SELECT Class(id) FROM CLASS
                    WHERE name='Object'));

INSERT INTO Parameter (name, datatype, default, refOperationin,
                       refOperationout)
   VALUES ('ascendId', 'OIDt', null,
                   (SELECT Operation (OID) FROM OPERATION
                    WHERE name='Object'),
                   null);

INSERT INTO Operation (name, specializ, visibility, static, refClass)
   VALUES ('Object', null, 'public', null,
                   (SELECT Class(id) FROM CLASS
                    WHERE name='Object'));

INSERT INTO Parameter (name, datatype, default, refOperationin,
                       refOperationout)
   VALUES ('entityName', 'VARCHAR (30)', null,
                   (SELECT Operation (OID) FROM OPERATION
                    WHERE name='Object'),
                   null);

INSERT INTO Operation (name, specializ, visibility, static, refClass)
   VALUES ('Object', null, 'public', null,
                   (SELECT Class(id) FROM CLASS
                    WHERE name='Object'));

INSERT INTO Parameter (name, datatype, default, refOperationin,
                       refOperationout)
   VALUES ('entityId', 'integer', null,
                   (SELECT Operation (OID) FROM OPERATION
                    WHERE name='Object'),
                   null);

INSERT INTO Parameter (name, datatype, default, refOperationin,
                       refOperationout)
   VALUES ('classId', 'integer', null,
                   (SELECT Operation (OID) FROM OPERATION
                    WHERE name='Object'),
                   null);

INSERT INTO Parameter (name, datatype, default, refOperationin,
                       refOperationout)
   VALUES ('versionId', 'integer', null,
                   (SELECT Operation (OID) FROM OPERATION
                    WHERE name='Object'),
                   null);

INSERT INTO Operation (name, specializ, visibility, static, refClass)
   VALUES ('delete', null, 'public', null,
                   (SELECT Class(id) FROM CLASS
                    WHERE name='Object'));

INSERT INTO Parameter (name, datatype, default, refOperationin,
                       refOperationout)
   VALUES ('allReferences', 'CHAR', null,
                   (SELECT Operation (OID) FROM OPERATION
                    WHERE name='delete'),
                   null);

INSERT INTO Operation (name, specializ, visibility, static, refClass)
   VALUES ('deleteObjectTree', null, 'public', null,
                   (SELECT Class(id) FROM CLASS
                    WHERE name='Object'));

INSERT INTO Parameter (name, datatype, default, refOperationin,
                       refOperationout)
   VALUES ('allReferences', 'CHAR', null,
                   (SELECT Operation (OID) FROM OPERATION
                    WHERE name='deleteObjectTree'),
                   null);
```

```
INSERT INTO Operation (name, specializ, visibility, static, refClass)
   VALUES ('findVersion', null, 'private', null,
                   (SELECT Class(id) FROM CLASS
                    WHERE name='Object'));

INSERT INTO Parameter (name, datatype, default, refOperationin,
                       refOperationout)
   VALUES ('entityId', 'integer', null,
                   (SELECT Operation (OID) FROM OPERATION
                    WHERE name='findVersion'),
                   null);

INSERT INTO Parameter (name, datatype, default, refOperationin,
                       refOperationout)
   VALUES ('classId', 'integer', null,
                   (SELECT Operation (OID) FROM OPERATION
                    WHERE name='findVersion'),
                   null);

INSERT INTO Operation (name, specializ, visibility, static, refClass)
   VALUES ('getAscendant', null, 'public', null,
                   (SELECT Class(id) FROM CLASS
                    WHERE name='Object'));

INSERT INTO Parameter (name, datatype, default, refOperationin,
                       refOperationout)
   VALUES ('', 'OIDt', null, null,
                   (SELECT Operation (OID) FROM OPERATION
                    WHERE name='getAscendant'));

INSERT INTO Operation (name, specializ, visibility, static, refClass)
   VALUES ('getAscendant', null, 'public', null,
                   (SELECT Class(id) FROM CLASS
                    WHERE name='Object'));
INSERT INTO Parameter (name, datatype, default, refOperationin,
                       refOperationout)
   VALUES ('className', 'VARCHAR (30)', null,
                   (SELECT Operation (OID) FROM OPERATION
                    WHERE name='getAscendant'),
                   null);

INSERT INTO Parameter (name, datatype, default, refOperationin,
                       refOperationout)
   VALUES ('', 'OIDt', null, null,
                   (SELECT Operation (OID) FROM OPERATION
                    WHERE name='getAscendant'));

INSERT INTO Operation (name, specializ, visibility, static, refClass)
   VALUES ('getClassId', null, 'private', null,
                   (SELECT Class(id) FROM CLASS
                    WHERE name='Object'));

INSERT INTO Parameter (name, datatype, default, refOperationin,
                       refOperationout)
   VALUES ('', 'integer', null, null,
                   (SELECT Operation (OID) FROM OPERATION
                    WHERE name='getClassId'));

INSERT INTO Operation (name, specializ, visibility, static, refClass)
   VALUES ('getClassName', null, 'public', null,
                   (SELECT Class(id) FROM CLASS
                    WHERE name='Object'));

INSERT INTO Parameter (name, datatype, default, refOperationin,
                       refOperationout)
   VALUES ('', 'VARCHAR (30)', null, null,
                   (SELECT Operation (OID) FROM OPERATION
                    WHERE name='getClassName'));

INSERT INTO Operation (name, specializ, visibility, static, refClass)
   VALUES ('getCompleteObject', null, 'public', null,
                   (SELECT Class(id) FROM CLASS
                    WHERE name='Object'));
```

```
INSERT INTO Parameter (name, datatype, default, refOperationin,
                       refOperationout)
   VALUES (’’, ’object’, null, null,
                    (SELECT Operation (OID) FROM OPERATION
                     WHERE name=’getCompleteObject’));

INSERT INTO Operation (name, specializ, visibility, static, refClass)
   VALUES (’getCorrespondence’, null, ’public’, null,
                    (SELECT Class(id) FROM CLASS
                     WHERE name=’Object’));

INSERT INTO Parameter (name, datatype, default, refOperationin,
                       refOperationout)
   VALUES (’ascendClassName’, ’VARCHAR (30)’, null,
                    (SELECT Operation (OID) FROM OPERATION
                     WHERE name=’getCorrespondence’),
                    null);

INSERT INTO Parameter (name, datatype, default, refOperationin,
                       refOperationout)
   VALUES (’’, ’VARCHAR (30)’, null, null,
                    (SELECT Operation (OID) FROM OPERATION
                     WHERE name=’getCorrespondence’));

INSERT INTO Operation (name, specializ, visibility, static, refClass)
   VALUES (’getCorrespondenceAsc’, null, ’public’, null,
                    (SELECT Class(id) FROM CLASS
                     WHERE name=’Object’));

INSERT INTO Parameter (name, datatype, default, refOperationin,
                       refOperationout)
   VALUES (’ascendClassName’, ’VARCHAR (30)’, null,
                    (SELECT Operation (OID) FROM OPERATION
                     WHERE name=’getCorrespondenceAsc’),
                    null);

INSERT INTO Parameter (name, datatype, default, refOperationin,
                       refOperationout)
   VALUES (’’, ’VARCHAR (30)’, null, null,
                    (SELECT Operation (OID) FROM OPERATION
                     WHERE name=’getCorrespondenceAsc’));

INSERT INTO Operation (name, specializ, visibility, static, refClass)
   VALUES (’getCorrespondenceDesc’, null, ’public’, null,
                    (SELECT Class(id) FROM CLASS
                     WHERE name=’Object’));


INSERT INTO Parameter (name, datatype, default, refOperationin,
                       refOperationout)
   VALUES (’ascendClassName’, ’VARCHAR (30)’, null,
                    (SELECT Operation (OID) FROM OPERATION
                     WHERE name=’getCorrespondenceDesc’),
                    null);

INSERT INTO Parameter (name, datatype, default, refOperationin,
                       refOperationout)
   VALUES (’’, ’VARCHAR (30)’, null, null,
                    (SELECT Operation (OID) FROM OPERATION
                     WHERE name=’getCorrespondenceDesc’));

INSERT INTO Operation (name, specializ, visibility, static, refClass)
   VALUES (’getDescendant’, null, ’public’, null,
                    (SELECT Class(id) FROM CLASS
                     WHERE name=’Object’));

INSERT INTO Parameter (name, datatype, default, refOperationin,
                       refOperationout)
   VALUES (’’, ’OIDt’, null, null,
                    (SELECT Operation (OID) FROM OPERATION
                     WHERE name=’getDescendant’));
```

```
INSERT INTO Operation (name, specializ, visibility, static, refClass)
   VALUES ('getDescendant', null, 'public', null,
                   (SELECT Class(id) FROM CLASS
                    WHERE name='Object'));

INSERT INTO Parameter (name, datatype, default, refOperationin,
                       refOperationout)
   VALUES ('className', 'VARCHAR (30)', null,
                   (SELECT Operation (OID) FROM OPERATION
                    WHERE name='getDescendant'),
                   null);

INSERT INTO Parameter (name, datatype, default, refOperationin,
                       refOperationout)
   VALUES ('', 'OIDt', null, null,
                   (SELECT Operation (OID) FROM OPERATION
                    WHERE name='getDescendant'));

INSERT INTO Operation (name, specializ, visibility, static, refClass)
   VALUES ('getEntityId', null, 'public', null,
                   (SELECT Class(id) FROM CLASS
                    WHERE name='Object'));

INSERT INTO Parameter (name, datatype, default, refOperationin,
                       refOperationout)
   VALUES ('entityName', 'VARCHAR (30)', null,
                   (SELECT Operation (OID) FROM OPERATION
                    WHERE name='getEntityId'),
                   null);

INSERT INTO Parameter (name, datatype, default, refOperationin,
                       refOperationout)
   VALUES ('', 'integer', null, null,
                   (SELECT Operation (OID) FROM OPERATION
                    WHERE name='getEntityId'));

INSERT INTO Operation (name, specializ, visibility, static, refClass)
   VALUES ('getNickName', null, 'public', null,
                   (SELECT Class(id) FROM CLASS
                    WHERE name='Object'));

INSERT INTO Parameter (name, datatype, default, refOperationin,
                       refOperationout)
   VALUES ('', 'string', null,  null,
                   (SELECT Operation (OID) FROM OPERATION
                    WHERE name='getNickName'));

INSERT INTO Operation (name, specializ, visibility, static, refClass)
   VALUES ('getObject', null, 'public', null,
                   (SELECT Class(id) FROM CLASS
                    WHEREname='Object'));

INSERT INTO Parameter (name, datatype, default, refOperationin,
                       refOperationout)
   VALUES ('', 'object', null, null,
                   (SELECT Operation (OID) FROM OPERATION
                    WHERE name='getObject'));

INSERT INTO Operation (name, specializ, visibility, static, refClass)
   VALUES ('getOID', null, 'public', null,
                   (SELECT Class(id) FROM CLASS
                    WHERE name='Object'));

INSERT INTO Parameter (name, datatype, default, refOperationin,
                       refOperationout)
   VALUES ('', 'OIDt', null,  null,
                   (SELECT Operation (OID) FROM OPERATION
                    WHERE name='getOID'));

INSERT INTO Operation (name, specializ, visibility, static, refClass)
   VALUES ('isDeleteAllowed', null, 'protected', null,
                   (SELECT Class(id) FROM CLASS
                    WHERE name='Object'));
```

```
INSERT INTO Parameter (name, datatype, default, refOperationin,
                       refOperationout)
   VALUES ('allReferences', 'CHAR', null,
                   (SELECT Operation (OID) FROM OPERATION
                    WHERE name='isDeleteAllowed'),
                   null);

INSERT INTO Parameter (name, datatype, default, refOperationin,
                       refOperationout)
   VALUES ('', 'CHAR', null, null,
                   (SELECT Operation (OID) FROM OPERATION
                    WHERE name='isDeleteAllowed'));

INSERT INTO Operation (name, specializ, visibility, static, refClass)
   VALUES ('isDeleteTreeAllowed', null, 'protected',  null,
                   (SELECT Class(id) FROM CLASS
                    WHERE name='Object'));

INSERT INTO Parameter (name, datatype, default, refOperationin,
                       refOperationout)
   VALUES ('allReferences', 'CHAR', null,
                   (SELECT Operation (OID) FROM OPERATION
                    WHERE name='isDeleteTreeAllowed'),
                   null);

INSERT INTO Parameter (name, datatype, default, refOperationin,
                       refOperationout)
   VALUES ('', 'CHAR', null, null,
                   (SELECT Operation (OID) FROM OPERATION
                    WHERE name='isDeleteTreeAllowed'));

INSERT INTO Operation (name, specializ, visibility, static, refClass)
   VALUES ('verifyAscendId', null, 'private', null,
                   (SELECT Class(id) FROM CLASS
                    WHERE name='Object'));

INSERT INTO Parameter (name, datatype, default, refOperationin,
                       refOperationout)
   VALUES ('ascendId', 'OIDt', null,
                   (SELECT Operation (OID) FROM OPERATION
                    WHERE name='verifyAscendId'),
                   null);

INSERT INTO Parameter (name, datatype, default, refOperationin,
                       refOperationout)
   VALUES ('', 'CHAR', null, null,
                   (SELECT Operation (OID) FROM OPERATION
                    WHERE name='verifyAscendId'));

INSERT INTO Operation (name, specializ, visibility, static, refClass)
   VALUES ('verifyEntityName', null, 'private', null,
                   (SELECT Class(id) FROM CLASS
                    WHERE name='Object'));

INSERT INTO Parameter (name, datatype, default, refOperationin,
                       refOperationout)
   VALUES ('entName', 'VARCHAR (30)', null,
                   (SELECT Operation (OID) FROM OPERATION
                    WHERE name='verifyEntityName'),
                   null);

INSERT INTO Parameter (name, datatype, default, refOperationin,
                       refOperationout)
   VALUES ('', 'CHAR', null, null,
                   (SELECT Operation (OID) FROM OPERATION
                    WHERE name='verifyEntityName'));

   INSERT INTO Class (name, tempVers, specializ, visibility,
                      root, refSchema)
      VALUES ('TemporalObject', 'F', 'abstract', 'public', 'F',
                       (SELECT Shema(id) FROM SCHEMA
                        WHERE name='TVM'));
```

```
<!-- METADADOS RELATIONSHIP INHERITANCE -->
 INSERT INTO Inheritance (name, relatedClass, single, refClass)
     VALUES (null, 'Object', 'F',
                (SELECT Class(id) FROM Class
                WHERE name='TemporalObject'));

<!-- METADADOS ATTRIBUTE -->
INSERT INTO Attribute (name, temp, visibility, static, type, default, refClass)
   VALUES ('alive', 'T', 'public', 'F', 'CHAR', 'T',
                    (SELECT Class(id) FROM CLASS
                     WHERE name='TemporalObject'));

<!-- METADADOS RELATIONSHIP -->
<!-- METADADOS OPERATION -->
INSERT INTO Operation (name, specializ, visibility, static, refClass)
   VALUES ('TemporalObject', null, 'public', null,
                    (SELECT Class(id) FROM CLASS
                     WHERE name='TemporalObject'));

INSERT INTO Operation (name, specializ, visibility, static, refClass)
   VALUES ('TemporalObject', null, 'public', null,
                    (SELECT Class(id) FROM CLASS
                     WHERE name='TemporalObject'));

INSERT INTO Parameter (name, datatype, default, refOperationin,
                       refOperationout)
   VALUES ('ascendId', 'OIDt', null,
                    (SELECT Operation (OID) FROM OPERATION
                     WHERE name='TemporalObject'),
                    null);

INSERT INTO Operation (name, specializ, visibility, static, refClass)
   VALUES ('TemporalObject', null, 'public', null,
                    (SELECT Class(id) FROM CLASS
                     WHERE name='TemporalObject'));

INSERT INTO Parameter (name, datatype, default, refOperationin,
                       refOperationout)
   VALUES ('entityName', 'VARCHAR (30)', null,
                    (SELECT Operation (OID) FROM OPERATION
                     WHERE name='TemporalObject'),
                    null);

INSERT INTO Operation (name, specializ, visibility, static, refClass)
   VALUES ('TemporalObject', null, 'public', null,
                    (SELECT Class(id) FROM CLASS
                     WHERE name='TemporalObject'));

INSERT INTO Parameter (name, datatype, default, refOperationin,
                       refOperationout)
   VALUES ('entityId', 'integer', null,
                    (SELECT Operation (OID) FROM OPERATION
                     WHERE name='TemporalObject'),
                    null);

INSERT INTO Parameter (name, datatype, default, refOperationin,
                       refOperationout)
   VALUES ('classId', 'integer', null,
                    (SELECT Operation (OID) FROM OPERATION
                     WHERE name='TemporalObject'),
                    null);

INSERT INTO Parameter (name, datatype, default, refOperationin,
                       refOperationout)
   VALUES ('versionId', 'integer', null,
                    (SELECT Operation (OID) FROM OPERATION
                     WHERE name='TemporalObject'),
                    null);

INSERT INTO Operation (name, specializ, visibility, static, refClass)
   VALUES ('classTempLabel', null, 'public', null,
                    (SELECT Class(id) FROM CLASS
```

```
                    WHERE name='TemporalObject'));

INSERT INTO Operation (name, specializ, visibility, static, refClass)
   VALUES ('delete', null, 'public', null,
                   (SELECT Class(id) FROM CLASS
                    WHERE name='TemporalObject'));

INSERT INTO Operation (name, specializ, visibility, static, refClass)
   VALUES ('getAlive', null, 'public', null,
                   (SELECT Class(id) FROM CLASS
                    WHERE name='TemporalObject'));

INSERT INTO Parameter (name, datatype, default, refOperationin,
                       refOperationout)
   VALUES ('', 'CHAR', null, null,
                   (SELECT Operation (OID) FROM OPERATION
                    WHERE name='getAlive'));

INSERT INTO Operation (name, specializ, visibility, static, refClass)
   VALUES ('getAttributeHistory', null, 'public', null,
                   (SELECT Class(id) FROM CLASS
                    WHERE  name='TemporalObject'));

INSERT INTO Parameter (name, datatype, default, refOperationin,
                       refOperationout)
   VALUES ('atribName', 'VARCHAR (30)', null,
                   (SELECT Operation (OID) FROM OPERATION
                    WHERE name='getAttributeHistory'),
                   null);

INSERT INTO Parameter (name, datatype, default, refOperationin,
                       refOperationout)
   VALUES ('', 'InstantAttribute', null, null,
                   (SELECT Operation (OID) FROM OPERATION
                    WHERE name='getAttributeHistory'));

INSERT INTO Operation (name, specializ, visibility, static, refClass)
   VALUES ('getAttributeValueAt', null, 'public', null,
                   (SELECT Class(id) FROM CLASS
                    WHERE name='TemporalObject'));

INSERT INTO Parameter (name, datatype, default, refOperationin,
                       refOperationout)
   VALUES ('atribName', 'VARCHAR (30)', null,
                   (SELECT Operation (OID) FROM OPERATION
                    WHERE name='getAttributeValueAt'),
                   null);

INSERT INTO Parameter (name, datatype, default, refOperationin,
                       refOperationout)
   VALUES ('i', 'TIMESTAMP', null,
                   (SELECT Operation (OID) FROM OPERATION
                    WHERE name='getAttributeValueAt'),
                   null);

INSERT INTO Parameter (name, datatype, default, refOperationin,
                       refOperationout)
   VALUES ('', 'InstantAttribute', null, null,
                   (SELECT Operation (OID) FROM OPERATION
                    WHERE name='getAttributeValueAt'));

INSERT INTO Operation (name, specializ, visibility, static, refClass)
   VALUES ('getLifetimeI', null, 'public', null,
                   (SELECT Class(id) FROM CLASS
                    WHERE name='TemporalObject'));

INSERT INTO Parameter (name, datatype, default, refOperationin,
                       refOperationout)
   VALUES ('', 'TIMESTAMP', null, null,
                   (SELECT Operation (OID) FROM OPERATION
                    WHERE name='getLifetimeI'));

INSERT INTO Operation (name, specializ, visibility, static, refClass)
```

```
   VALUES ('getLifetimeF', null, 'public', null,
                  (SELECT Class(id) FROM CLASS
                   WHERE name='TemporalObject'));

INSERT INTO Parameter (name, datatype, default, refOperationin,
                       refOperationout)
   VALUES ('', 'TIMESTAMP', null,  null,
                  (SELECT Operation (OID) FROM OPERATION
                   WHERE name='getLifetimeF'));

INSERT INTO Operation (name, specializ, visibility, static, refClass)
   VALUES ('getObjectHistory', null, 'public', null,
                  (SELECT Class(id) FROM CLASS
                   WHERE name='TemporalObject'));

INSERT INTO Parameter (name, datatype, default, refOperationin,
                       refOperationout)
   VALUES ('', 'instantAttribute', null, null,
                  (SELECT Operation (OID) FROM OPERATION
                   WHERE name='getObjectHistory'));

INSERT INTO Operation (name, specializ, visibility, static, refClass)
   VALUES ('getRelationshipHistory', null, 'public', null,
                  (SELECT Class(id) FROM CLASS
                   WHERE  name='TemporalObject'));

INSERT INTO Parameter (name, datatype, default, refOperationin,
                       refOperationout)
   VALUES ('relatedObjId', 'OIDt', null,
                  (SELECT Operation (OID) FROM OPERATION
                   WHERE name='getRelationshipHistory'),
                  null);

INSERT INTO Parameter (name, datatype, default, refOperationin,
                       refOperationout)
   VALUES ('relatedName', 'VARCHAR  (30)', null,
                  (SELECT Operation (OID) FROM OPERATION
                   WHERE name='getRelationshipHistory'),
                  null);

INSERT INTO Parameter (name, datatype, default, refOperationin,
                       refOperationout)
   VALUES ('', 'InstantRelationship', null, null,
                  (SELECT Operation (OID) FROM OPERATION
                   WHERE name='getRelationshipHistory'));

INSERT INTO Operation (name, specializ, visibility, static, refClass)
   VALUES ('getRelationshipHistoryAt', null, 'public', null,
                  (SELECT Class(id) FROM CLASS
                   WHERE name='TemporalObject'));

INSERT INTO Parameter (name, datatype, default, refOperationin,
                       refOperationout)
   VALUES ('relatedObjId', 'OIDt', null,
                  (SELECT Operation (OID) FROM OPERATION
                   WHERE name='getRelationshipHistoryAt'),
                  null);

INSERT INTO Parameter (name, datatype, default, refOperationin,
                       refOperationout)
   VALUES ('relatedName', 'VARCHAR (30)', null,
                  (SELECT Operation (OID) FROM OPERATION
                   WHERE name='getRelationshipHistoryAt'),
                  null);

INSERT INTO Parameter (name, datatype, default, refOperationin,
                       refOperationout)
   VALUES ('i', 'TIMESTAMP', null,
                  (SELECT Operation (OID) FROM OPERATION
                   WHERE name='getRelationshipHistoryAt'),
                  null);

INSERT INTO Parameter (name, datatype, default, refOperationin,
```

```
                        refOperationout)
   VALUES (’’, ’InstantRelationship’, null,  null,
                    (SELECT Operation (OID) FROM OPERATION
                     WHERE name=’getRelationshipHistoryAt’));

INSERT INTO Operation (name, specializ, visibility, static, refClass)
   VALUES (’setTemporalAttribute’, null, ’public’, null,
                    (SELECT Class(id) FROM CLASS
                     WHERE name=’TemporalObject’));

INSERT INTO Parameter (name, datatype, default, refOperationin,
                        refOperationout)
   VALUES (’attribName’, ’VARCHAR (30)’, null,
                    (SELECT Operation (OID) FROM OPERATION
                     WHERE name=’setTemporalAttribute’),
                    null);

INSERT INTO Parameter (name, datatype, default, refOperationin,
                        refOperationout)
   VALUES (’newValue’, ’VARCHAR (10)’, null,
                    (SELECT Operation (OID) FROM OPERATION
                     WHERE name=’setTemporalAttribute’),
                    null);

INSERT INTO Operation (name, specializ, visibility, static, refClass)
   VALUES (’setTemporalRelationship’, null, ’public’, null,
                    (SELECT Class(id) FROM CLASS
                     WHERE name=’TemporalObject’));

INSERT INTO Parameter (name, datatype, default, refOperationin,
                        refOperationout)
   VALUES (’relatedName’, ’VARCHAR (30)’, null,
                    (SELECT Operation (OID) FROM OPERATION
                     WHERE name=’setTemporalRelationship’),
                    null);

INSERT INTO Parameter (name, datatype, default, refOperationin,
                        refOperationout)
   VALUES (’newO1’, ’OIDt’, null,
                    (SELECT Operation (OID) FROM OPERATION
                     WHERE name=’setTemporalRelationship’),
                    null);

INSERT INTO Parameter (name, datatype, default, refOperationin,
                        refOperationout)
   VALUES (’newO2’, ’OIDt’, null,
                    (SELECT Operation (OID) FROM OPERATION
                     WHERE name=’setTemporalRelationship’),
                    null);

   INSERT INTO Class (name, tempVers, specializ, visibility,
                      root, refSchema)
      VALUES (’TemporalVersion’, ’F’, ’abstract’, ’public’, ’F’,
                       (SELECT Shema(id) FROM SCHEMA
                        WHERE name=’TVM’));

<!-- METADADOS RELATIONSHIP INHERITANCE -->
 INSERT INTO Inheritance (name, relatedClass, single, refClass)
     VALUES (null, ’TemporalObject’, ’F’,
                      (SELECT Class(id) FROM Class
                       WHERE name=’TemporalVersion’));

<!-- METADADOS ATTRIBUTE -->
INSERT INTO Attribute (name, temp, visibility, static, type, default, refClass)
   VALUES (’ascendant’, ’T’, ’public’, ’F’, ’OIDt’, ’NULL’,
                    (SELECT Class(id) FROM CLASS
                     WHERE name=’TemporalVersion’));

INSERT INTO Attribute (name, temp, visibility, static, type, default, refClass)
   VALUES (’descendant’, ’T’, ’public’, ’F’, ’OIDt’, ’NULL’,
                    (SELECT Class(id) FROM CLASS
                     WHERE name=’TemporalVersion’));
```

```
 INSERT INTO Attribute (name, temp, visibility, static, type, default, refClass)
    VALUES ('successor', 'T', 'public', 'F', 'OIDt', 'NULL',
                     (SELECT Class(id) FROM CLASS
                      WHERE name='TemporalVersion'));

 INSERT INTO Attribute (name, temp, visibility, static, type, default, refClass)
    VALUES ('predecessor', 'F', 'public', 'F', 'OIDt', 'NULL',
                     (SELECT Class(id) FROM CLASS
                      WHERE name='TemporalVersion'));

 INSERT INTO Attribute (name, temp, visibility, static, type, default, refClass)
    VALUES ('status', 'T', 'public', 'F', 'char', ''W'',
                     (SELECT Class(id) FROM CLASS
                      WHERE name='TemporalVersion'));

 INSERT INTO Attribute (name, temp, visibility, static, type, default, refClass)
    VALUES ('configuration', 'F', 'public', 'F', 'CHAR', 'F',
                     (SELECT Class(id) FROM CLASS
                      WHERE name='TemporalVersion'));

<!-- METADADOS RELATIONSHIP -->
<!-- METADADOS OPERATION -->
 INSERT INTO Operation (name, specializ, visibility, static, refClass)
    VALUES ('TemporalVersion', null, 'public', null,
                     (SELECT Class(id) FROM CLASS
                      WHERE name='TemporalVersion'));

 INSERT INTO Operation (name, specializ, visibility, static, refClass)
    VALUES ('TemporalVersion', null, 'public', null,
                     (SELECT Class(id) FROM CLASS
                      WHERE name='TemporalVersion'));

 INSERT INTO Parameter (name, datatype, default, refOperationin,
                        refOperationout)
    VALUES ('ascendId', 'OIDt', null,
                     (SELECT Operation (OID) FROM OPERATION
                      WHERE name='TemporalVersion'),
                     null);

 INSERT INTO Operation (name, specializ, visibility, static, refClass)
    VALUES ('TemporalVersion', null, 'public', null,
                     (SELECT Class(id) FROM CLASS
                      WHERE name='TemporalVersion'));

 INSERT INTO Parameter (name, datatype, default, refOperationin,
                        refOperationout)
    VALUES ('entityName', 'VARCHAR (30)', null,
                     (SELECT Operation (OID) FROM OPERATION
                      WHERE name='TemporalVersion'),
                     null);

 INSERT INTO Operation (name, specializ, visibility, static, refClass)
    VALUES ('TemporalVersion', null, 'public', null,
                     (SELECT Class(id) FROM CLASS
                      WHERE name='TemporalVersion'));

 INSERT INTO Parameter (name, datatype, default, refOperationin,
                        refOperationout)
    VALUES ('entityId', 'integer', null,
                     (SELECT Operation (OID) FROM OPERATION
                      WHERE name='TemporalVersion'),
                     null);

 INSERT INTO Parameter (name, datatype, default, refOperationin,
                        refOperationout)
    VALUES ('classId', 'integer', null,
                     (SELECT Operation (OID) FROM OPERATION
                      WHERE name='TemporalVersion'),
                     null);

 INSERT INTO Parameter (name, datatype, default, refOperationin,
                        refOperationout)
    VALUES ('versionId', 'integer', null,
```

```
                    (SELECT Operation (OID) FROM OPERATION
                     WHERE  name='TemporalVersion'),
                    null);

INSERT INTO Operation (name, specializ, visibility, static, refClass)
   VALUES ('TemporalVersion', null, 'private', null,
                    (SELECT Class(id) FROM CLASS
                     WHERE name='TemporalVersion'));

INSERT INTO Parameter (name, datatype, default, refOperationin,
                       refOperationout)
   VALUES ('predecId', 'OIDt', null,
                    (SELECT Operation (OID) FROM OPERATION
                     WHERE name='TemporalVersion'),
                    null);

INSERT INTO Parameter (name, datatype, default, refOperationin,
                       refOperationout)
   VALUES ('ascendId', 'OIDt', null,
                    (SELECT Operation (OID) FROM OPERATION
                     WHERE name='TemporalVersion'),
                    null);

INSERT INTO Parameter (name, datatype, default, refOperationin,
                       refOperationout)
   VALUES ('config', 'CHAR', null,
                    (SELECT Operation (OID) FROM OPERATION
                     WHERE name='TemporalVersion'),
                    null);

INSERT INTO Operation (name, specializ, visibility, static, refClass)
   VALUES ('addAscendant', null, 'private', null,
                    (SELECT Class(id) FROM CLASS
                     WHERE name='TemporalVersion'));

INSERT INTO Parameter (name, datatype, default, refOperationin,
                       refOperationout)
   VALUES ('ascendId', 'OIDt', null,
                 (SELECT Operation (OID) FROM OPERATION
                  WHERE name='addAscendant'),
                    null);

INSERT INTO Parameter (name, datatype, default, refOperationin,
                       refOperationout)
   VALUES ('', 'CHAR', null, null,
                    (SELECT Operation (OID) FROM OPERATION
                     WHERE name='addAscendant'));

INSERT INTO Operation (name, specializ, visibility, static, refClass)
   VALUES ('addDescendant', null, 'private', null,
                    (SELECT Class(id) FROM CLASS
                     WHERE name='TemporalVersion'));

INSERT INTO Parameter (name, datatype, default, refOperationin,
                       refOperationout)
   VALUES ('descendId', 'OIDt', null,
                    (SELECT Operation (OID) FROM OPERATION
                     WHERE name='addDescendant'),
                    null);

INSERT INTO Parameter (name, datatype, default, refOperationin,
                       refOperationout)
   VALUES ('', 'CHAR', null, null,
                    (SELECT Operation (OID) FROM OPERATION
                     WHERE name='addDescendant'));

INSERT INTO Operation (name, specializ, visibility, static, refClass)
   VALUES ('addSuccessor', null, 'private', null,
                    (SELECT Class(id) FROM CLASS
                     WHERE name='TemporalVersion'));

INSERT INTO Parameter (name, datatype, default, refOperationin,
                       refOperationout)
```

```
   VALUES ('succID', 'OIDt', null,
                   (SELECT Operation (OID) FROM OPERATION
                    WHERE name='addSuccessor'),
                   null);

INSERT INTO Operation (name, specializ, visibility, static, refClass)
   VALUES ('delete', null, 'public', null,
                   (SELECT Class(id) FROM CLASS
                    WHERE name='TemporalVersion'));

INSERT INTO Parameter (name, datatype, default, refOperationin,
                       refOperationout)
   VALUES ('allReferences', 'CHAR', null,
                   (SELECT Operation (OID) FROM OPERATION
                    WHERE name='delete'),
                   null);

INSERT INTO Operation (name, specializ, visibility, static, refClass)
   VALUES ('deleteObjectTree', null, 'public', null,
                   (SELECT Class(id) FROM CLASS
                    WHERE name='TemporalVersion'));

INSERT INTO Parameter (name, datatype, default, refOperationin,
                       refOperationout)
   VALUES ('allReferences', 'CHAR', null,
                   (SELECT Operation (OID) FROM OPERATION
                    WHERE name='deleteObjectTree'),
                   null);

INSERT INTO Operation (name, specializ, visibility, static, refClass)
   VALUES ('derive', null, 'public', null,
                   (SELECT Class(id) FROM CLASS
                    WHERE name='TemporalVersion'));

INSERT INTO Parameter (name, datatype, default, refOperationin,
                       refOperationout)
   VALUES ('versionId', 'OIDt', null,
                   (SELECT Operation (OID) FROM OPERATION
                    WHERE name='derive'),
                   null);

INSERT INTO Operation (name, specializ, visibility, static, refClass)
   VALUES ('derive', null, 'public', null,
                   (SELECT Class(id) FROM CLASS
                    WHERE name='TemporalVersion'));

INSERT INTO Parameter (name, datatype, default, refOperationin,
                       refOperationout)
   VALUES ('versionId', 'OIDt', null,
                   (SELECT Operation (OID) FROM OPERATION
                    WHERE name='derive'),
                   null);

INSERT INTO Parameter (name, datatype, default, refOperationin,
                       refOperationout)
   VALUES ('ascendId', 'OIDt', null,
                   (SELECT Operation (OID) FROM OPERATION
                    WHERE name='derive'),
                   null);

INSERT INTO Parameter (name, datatype, default, refOperationin,
                       refOperationout)
   VALUES ('config', 'CHAR', null,
                   (SELECT Operation (OID) FROM OPERATION
                    WHERE name='derive'),
                   null);

INSERT INTO Operation (name, specializ, visibility, static, refClass)
   VALUES ('getAscendant', null, 'public', null,
                   (SELECT Class(id) FROM CLASS
                    WHERE name='TemporalVersion'));

INSERT INTO Parameter (name, datatype, default, refOperationin,
```

```
                      refOperationout)
   VALUES (’’, ’OIDt’, null,  null,
                    (SELECT Operation (OID) FROM OPERATION
                     WHERE name=’getAscendant’));

INSERT INTO Operation (name, specializ, visibility, static, refClass)
   VALUES (’getAscendant’, null, ’public’, null,
                    (SELECT Class(id) FROM CLASS
                     WHERE name=’TemporalVersion’));

INSERT INTO Parameter (name, datatype, default, refOperationin,
                      refOperationout)
   VALUES (’className’, ’VARCHAR (30)’, null,
                    (SELECT Operation (OID) FROM OPERATION
                     WHERE name=’getAscendant’),
                    null);

INSERT INTO Parameter (name, datatype, default, refOperationin,
                      refOperationout)
   VALUES (’’, ’OIDt’, null,  null,
                    (SELECT Operation (OID) FROM OPERATION
                     WHERE name=’getAscendant’));

INSERT INTO Operation (name, specializ, visibility, static, refClass)
   VALUES (’getConfiguration’, null, ’public’, null,
                    (SELECT Class(id) FROM CLASS
                     WHERE name=’TemporalVersion’));

INSERT INTO Parameter (name, datatype, default, refOperationin,
                      refOperationout)
   VALUES (’’, ’OIDt’, null,  null,
                    (SELECT Operation (OID) FROM OPERATION
                     WHERE name=’getConfiguration’));

INSERT INTO Operation (name, specializ, visibility, static, refClass)
   VALUES (’getCompleteObject’, null,  ’public’, null,
                    (SELECT Class(id) FROM CLASS
                     WHERE name=’TemporalVersion’));

INSERT INTO Operation (name, specializ, visibility, static, refClass)
   VALUES (’getDescendant’, null, ’public’, null,
                    (SELECT Class(id) FROM CLASS
                     WHERE name=’TemporalVersion’));

INSERT INTO Parameter (name, datatype, default, refOperationin,
                      refOperationout)
   VALUES (’’, ’OIDt’, null,  null,
                    (SELECT Operation (OID) FROM OPERATION
                     WHERE name=’getDescendant’));

INSERT INTO Operation (name, specializ, visibility, static, refClass)
   VALUES (’getDescendant’, null, ’public’, null,
                    (SELECT Class(id) FROM CLASS
                      WHERE name=’TemporalVersion’));

INSERT INTO Parameter (name, datatype, default, refOperationin,
                      refOperationout)
   VALUES (’className’, ’VARCHAR (30)’, null,
                    (SELECT Operation (OID) FROM OPERATION
                     WHERE name=’getDescendant’),
                    null);

INSERT INTO Parameter (name, datatype, default, refOperationin,
                      refOperationout)
   VALUES (’criterion’, ’VARCHAR (50)’, null,
                    (SELECT Operation (OID) FROM OPERATION
                     WHERE name=’getDescendant’),
                    null);

INSERT INTO Parameter (name, datatype, default, refOperationin,
                      refOperationout)
   VALUES (’’, ’OIDt’, null,  null,
                    (SELECT Operation (OID) FROM OPERATION
```

```
                    WHERE name='getDescendant'));

INSERT INTO Operation (name, specializ, visibility, static, refClass)
   VALUES ('getOIDControl', null,  'public', null,
                   (SELECT Class(id) FROM CLASS
                    WHERE name='TemporalVersion'));

INSERT INTO Parameter (name, datatype, default, refOperationin,
                       refOperationout)
   VALUES ('', 'OIDt', null,  null,
                   (SELECT Operation (OID) FROM OPERATION
                    WHERE name='getOIDControl'));

INSERT INTO Operation (name, specializ, visibility, static, refClass)
   VALUES ('getPredeccessor', null, 'public',  null,
                   (SELECT Class(id) FROM CLASS
                    WHERE name='TemporalVersion'));

INSERT INTO Parameter (name, datatype, default, refOperationin,
                       refOperationout)
   VALUES ('',  'OIDt', null,  null,
                   (SELECT Operation (OID) FROM OPERATION
                    WHERE name='getPredeccessor'));

INSERT INTO Operation (name, specializ, visibility, static, refClass)
   VALUES ('getStatus', null, 'public', null,
                   (SELECT Class(id) FROM CLASS
                    WHERE name='TemporalVersion'));

INSERT INTO Parameter (name, datatype, default, refOperationin,
                       refOperationout)
   VALUES ('', 'char', null,  null,
                   (SELECT Operation (OID) FROM OPERATION
                    WHERE name='getStatus'));

INSERT INTO Operation (name, specializ, visibility, static, refClass)
   VALUES ('getSuccessor', null,  'public', null,
                   (SELECT Class(id) FROM CLASS
                    WHERE name='TemporalVersion'));

INSERT INTO Parameter (name, datatype, default, refOperationin,
                       refOperationout)
   VALUES ('onlyConfigured', 'CHAR', null,
                   (SELECT Operation (OID) FROM OPERATION
                    WHERE name='getSuccessor'),
                   null);

INSERT INTO Parameter (name, datatype, default, refOperationin,
                       refOperationout)
   VALUES ('', 'OIDt', null,  null,
                   (SELECT Operation (OID) FROM OPERATION
                    WHERE name='getSuccessor'));

INSERT INTO Operation (name, specializ, visibility, static, refClass)
   VALUES ('getVersionedObjectId', null, 'public', null,
                   (SELECT Class(id) FROM CLASS
                    WHERE name='TemporalVersion'));

INSERT INTO Parameter (name, datatype, default, refOperationin,
                       refOperationout)
   VALUES ('', 'OIDt', null,  null,
                   (SELECT Operation (OID) FROM OPERATION
                    WHERE name='getVersionedObjectId'));

INSERT INTO Operation (name, specializ, visibility, static, refClass)
   VALUES ('isDeleteAllowed', null, 'private', null,
                   (SELECT Class(id) FROM CLASS
                    WHERE name='TemporalVersion'));

INSERT INTO Parameter (name, datatype, default, refOperationin,
                       refOperationout)
   VALUES ('', 'CHAR', null,  null,
                   (SELECT Operation (OID) FROM OPERATION
```

```
                    WHERE name='isDeleteAllowed'));

INSERT INTO Operation (name, specializ, visibility, static, refClass)
   VALUES ('isDeleteTreeAllowed', null, 'private', null,
                   (SELECT Class(id) FROM CLASS
                    WHERE name='TemporalVersion'));

INSERT INTO Parameter (name, datatype, default, refOperationin,
                       refOperationout)
   VALUES ('', 'CHAR', null,  null,
                   (SELECT Operation (OID) FROM OPERATION
                    WHERE name='isDeleteTreeAllowed'));

INSERT INTO Operation (name, specializ, visibility, static, refClass)
   VALUES ('promote', null, 'public', null,
                   (SELECT Class(id) FROM CLASS
                    WHERE name='TemporalVersion'));

INSERT INTO Parameter (name, datatype, default, refOperationin,
                       refOperationout)
   VALUES ('allAscendant', 'CHAR', null,
                   (SELECT Operation (OID) FROM OPERATION
                    WHERE name='promote'),
                   null);

INSERT INTO Parameter (name, datatype, default, refOperationin,
                       refOperationout)
   VALUES ('allReferenced', 'CHAR', null,
                   (SELECT Operation (OID) FROM OPERATION
                    WHERE name='promote'),
                   null);

INSERT INTO Operation (name, specializ, visibility, static, refClass)
   VALUES ('removeAscendant', null, 'private', null,
                   (SELECT Class(id) FROM CLASS
                    WHERE name='TemporalVersion'));

INSERT INTO Parameter (name, datatype, default, refOperationin,
                       refOperationout)
   VALUES ('ascendId', 'OIDt', null,
                   (SELECT Operation (OID) FROM OPERATION
                    WHERE name='removeAscendant'),
                   null);

INSERT INTO Operation (name, specializ, visibility, static, refClass)
   VALUES ('removeDescendant', null, 'private', null,
                   (SELECT Class(id) FROM CLASS
                    WHERE name='TemporalVersion'));

INSERT INTO Parameter (name, datatype, default, refOperationin,
                       refOperationout)
   VALUES ('descendId', 'OIDt', null,
                   (SELECT Operation (OID) FROM OPERATION
                    WHERE name='removeDescendant'),
                   null);

INSERT INTO Operation (name, specializ, visibility, static, refClass)
   VALUES ('removeSuccessor', null, 'private', null,
                   (SELECT Class(id) FROM CLASS
                    WHERE name='TemporalVersion'));

INSERT INTO Parameter (name, datatype, default, refOperationin,
                       refOperationout)
   VALUES ('succId', 'OIDt', null,
                   (SELECT Operation (OID) FROM OPERATION
                    WHERE name='removeSuccessor'),
                   null);

INSERT INTO Operation (name, specializ, visibility, static, refClass)
   VALUES ('restore', null, 'public', null,
                   (SELECT Class(id) FROM CLASS
                    WHERE ='TemporalVersion'));
```

```
INSERT INTO Parameter (name, datatype, default, refOperationin,
                       refOperationout)
   VALUES ('OID', 'OIDt', null,
                    (SELECT Operation (OID) FROM OPERATION
                     WHERE name='restore'),
                    null);

INSERT INTO Parameter (name, datatype, default, refOperationin,
                       refOperationout)
   VALUES ('', 'CHAR', null,  null,
                    (SELECT Operation (OID) FROM OPERATION
                     WHERE name='restore'));

INSERT INTO Operation (name, specializ, visibility, static, refClass)
   VALUES ('setAscendant', null, 'private', null,
                    (SELECT Class(id) FROM CLASS
                     WHERE name='TemporalVersion'));

INSERT INTO Parameter (name, datatype, default, refOperationin,
                       refOperationout)
   VALUES ('ascendId', 'OIDt', null,
                    (SELECT Operation (OID) FROM OPERATION
                     WHERE name='setAscendant'),
                    null);

INSERT INTO Operation (name, specializ, visibility, static, refClass)
   VALUES ('setDecendant', null,  'private', null,
                    (SELECT Class(id) FROM CLASS
                     WHERE  name='TemporalVersion'));

INSERT INTO Parameter (name, datatype, default, refOperationin,
                       refOperationout)
   VALUES ('descendId', 'OIDt', null,
                    (SELECT Operation (OID) FROM OPERATION
                     WHERE name='setDecendant'),
                    null);

INSERT INTO Operation (name, specializ, visibility, static, refClass)
   VALUES ('setSatus', null, 'private', null,
                    (SELECT Class(id) FROM CLASS
                     WHERE name='TemporalVersion'));

INSERT INTO Parameter (name, datatype, default, refOperationin,
                       refOperationout)
   VALUES ('newStatus', 'char', null,
                    (SELECT Operation (OID) FROM OPERATION
                     WHERE name='setSatus'),
                    null);

INSERT INTO Operation (name, specializ, visibility, static, refClass)
   VALUES ('setSuccessor', null, 'private', null,
                    (SELECT Class(id) FROM CLASS
                     WHERE  name='TemporalVersion'));

INSERT INTO Parameter (name, datatype, default, refOperationin,
                       refOperationout)
   VALUES ('succId', 'OIDt', null,
                    (SELECT Operation (OID) FROM OPERATION
                     WHERE name='setSuccessor'),
                    null);

INSERT INTO Parameter (name, datatype, default, refOperationin,
                       refOperationout)
   VALUES ('', 'CHAR', null,  null,
                    (SELECT Operation (OID) FROM OPERATION
                     WHERE name='setSuccessor'));

INSERT INTO Operation (name, specializ, visibility, static, refClass)
   VALUES ('verifyAscendId', null, 'private', null,
                    (SELECT Class(id) FROM CLASS
                     WHERE name='TemporalVersion'));

INSERT INTO Parameter (name, datatype, default, refOperationin,
```

```
                        refOperationout)
   VALUES ('ascendId', 'OIDt', null,
                    (SELECT Operation (OID) FROM OPERATION
                     WHERE name='verifyAscendId'),
                    null);

 INSERT INTO Parameter (name, datatype, default, refOperationin,
                        refOperationout)
   VALUES ('', 'CHAR', null, null,
                    (SELECT Operation (OID) FROM OPERATION
                     WHERE name='verifyAscendId'));

   INSERT INTO Class (name, tempVers, specializ, visibility,
                      root, refSchema)
      VALUES ('VersionedObjectControl', 'F', 'final', 'public', 'F',
                       (SELECT Shema(id) FROM SCHEMA
                        WHERE name='TVM'));

   <!-- METADADOS RELATIONSHIP INHERITANCE -->
      INSERT INTO Inheritance (name, relatedClass, single, refClass)
         VALUES (null,
                'TemporalObject',
                'F',
                (SELECT Class(id) FROM Class
                WHERE name='VersionedObjectControl'));

 <!-- METADADOS ATTRIBUTE -->
 INSERT INTO Attribute (name, temp, visibility, static, type, default, refClass)
   VALUES ('configurationCount', 'T', 'public', 'F', 'integer', '0',
                    (SELECT Class(id) FROM CLASS
                     WHERE name='VersionedObjectControl'));

 INSERT INTO Attribute (name, temp, visibility, static, type, default, refClass)
   VALUES ('currentVersion', 'T', 'public', 'F', 'OIDt','',
                    (SELECT Class(id) FROM CLASS
                     WHERE name='VersionedObjectControl'));

 INSERT INTO Attribute (name, temp, visibility, static, type, default, refClass)
   VALUES ('firstVersion', 'T', 'public', 'F', 'OIDt','',
                    (SELECT Class(id) FROM CLASS
                     WHERE name='VersionedObjectControl'));

 INSERT INTO Attribute (name, temp, visibility, static, type, default, refClass)
   VALUES ('lastVersion', 'T', 'public', 'F', 'OIDt','',
                    (SELECT Class(id) FROM CLASS
                     WHERE name='VersionedObjectControl'));

 INSERT INTO Attribute (name, temp, visibility, static, type, default, refClass)
   VALUES ('nextVersionNumber', 'F', 'public', 'F', 'integer', '3',
                    (SELECT Class(id) FROM CLASS
                     WHERE name='VersionedObjectControl'));

 INSERT INTO Attribute (name, temp, visibility, static, type, default, refClass)
   VALUES ('userCurrentVersion', 'T', 'public', 'F', 'CHAR', 'F',
                    (SELECT Class(id) FROM CLASS
                     WHERE name='VersionedObjectControl'));

 INSERT INTO Attribute (name, temp, visibility, static, type, default, refClass)
   VALUES ('versionCount', 'T', 'public', 'F', 'integer', '2',
                    (SELECT Class(id) FROM CLASS
                     WHERE name='VersionedObjectControl'));

<!-- METADADOS RELATIONSHIP -->
<!-- METADADOS OPERATION -->
 INSERT INTO Operation (name, specializ, visibility, static, refClass)
   VALUES ('VersionedObjectControl',  null, 'public', null,
                    (SELECT Class(id) FROM CLASS
                     WHERE name='VersionedObjectControl'));

 INSERT INTO Parameter (name, datatype, default, refOperationin,
                        refOperationout)
   VALUES ('entityId', 'integer', null,
                    (SELECT Operation (OID) FROM OPERATION
```

```
                    WHERE name='VersionedObjectControl'),
                  null);

INSERT INTO Parameter (name, datatype, default, refOperationin,
                       refOperationout)
   VALUES ('classId', 'integer', null,
                  (SELECT Operation (OID) FROM OPERATION
                    WHERE name='VersionedObjectControl'),
                  null);

INSERT INTO Parameter (name, datatype, default, refOperationin,
                       refOperationout)
   VALUES ('configCount', 'integer', null,
                  (SELECT Operation (OID) FROM OPERATION
                    WHERE name='VersionedObjectControl'),
                  null);

INSERT INTO Parameter (name, datatype, default, refOperationin,
                       refOperationout)
   VALUES ('currentV', 'OIDt', null,
                  (SELECT Operation (OID) FROM OPERATION
                    WHERE name='VersionedObjectControl'),
                  null);

INSERT INTO Parameter (name, datatype, default, refOperationin,
                       refOperationout)
   VALUES ('firstV', 'OIDt', null,
                  (SELECT Operation (OID) FROM OPERATION
                    WHERE name='VersionedObjectControl'),
                  null);

INSERT INTO Parameter (name, datatype, default, refOperationin,
                       refOperationout)
   VALUES ('lastV', 'OIDt', null,
                  (SELECT Operation (OID) FROM OPERATION
                    WHERE name='VersionedObjectControl'),
                  null);

INSERT INTO Parameter (name, datatype, default, refOperationin,
                       refOperationout)
   VALUES ('nextVNumber', 'integer', null,
                  (SELECT Operation (OID) FROM OPERATION
                    WHERE name='VersionedObjectControl'),
                  null);

INSERT INTO Parameter (name, datatype, default, refOperationin,
                       refOperationout)
   VALUES ('userCurrentFlag', 'CHAR', null,
                  (SELECT Operation (OID) FROM OPERATION
                    WHERE name='VersionedObjectControl'),
                  null);

INSERT INTO Parameter (name, datatype, default, refOperationin,
                       refOperationout)
   VALUES ('vCount', 'integer', null,
                  (SELECT Operation (OID) FROM OPERATION
                    WHERE name='VersionedObjectControl'),
                  null);

INSERT INTO Operation (name, specializ, visibility, static, refClass)
   VALUES ('VersionedObjectControl', null, 'public', null,
                  (SELECT Class(id) FROM CLASS
                    WHERE name='VersionedObjectControl'));

INSERT INTO Parameter (name, datatype, default, refOperationin,
                       refOperationout)
   VALUES ('entityId', 'integer', null,
                  (SELECT Operation (OID) FROM OPERATION
                    WHERE name='VersionedObjectControl'),
                  null);

INSERT INTO Parameter (name, datatype, default, refOperationin,
                       refOperationout)
```

```
   VALUES ('classId', 'integer', null,
                  (SELECT Operation (OID) FROM OPERATION
                   WHERE name='VersionedObjectControl'),
                  null);

INSERT INTO Parameter (name, datatype, default, refOperationin,
                       refOperationout)
   VALUES ('currentV', 'OIDt', null,
                  (SELECT Operation (OID) FROM OPERATION
                   WHERE name='VersionedObjectControl'),
                  null);

INSERT INTO Parameter (name, datatype, default, refOperationin,
                       refOperationout)
   VALUES ('firstV', 'OIDt', null,
                  (SELECT Operation (OID) FROM OPERATION
                   WHERE name='VersionedObjectControl'),
                  null);

INSERT INTO Parameter (name, datatype, default, refOperationin,
                       refOperationout)
   VALUES ('lastV', 'OIDt', null,
                  (SELECT Operation (OID) FROM OPERATION
                   WHERE name='VersionedObjectControl'),
                  null);

INSERT INTO Operation (name, specializ, visibility, static, refClass)
   VALUES ('delete', null,  'public', null,
                  (SELECT Class(id) FROM CLASS
                   WHERE name='VersionedObjectControl'));

INSERT INTO Parameter (name, datatype, default, refOperationin,
                       refOperationout)
   VALUES ('entityId', 'integer', null,
                  (SELECT Operation (OID) FROM OPERATION
                   WHERE name='delete'),
                  null);

INSERT INTO Parameter (name, datatype, default, refOperationin,
                       refOperationout)
   VALUES ('classId', 'integer', null,
                  (SELECT Operation (OID) FROM OPERATION
                   WHERE name='delete'),
                  null);

INSERT INTO Operation (name, specializ, visibility, static, refClass)
   VALUES ('getConfigurationCount', null,  'public',null,
                  (SELECT Class(id) FROM CLASS
                   WHERE name='VersionedObjectControl'));

INSERT INTO Parameter (name, datatype, default, refOperationin,
                       refOperationout)
   VALUES ('', 'integer', null,  null,
                  (SELECT Operation (OID) FROM OPERATION
                   WHERE name='getConfigurationCount'));

INSERT INTO Operation (name, specializ, visibility, static, refClass)
   VALUES ('getCurrentVersion', null,  'public', null,
                  (SELECT Class(id) FROM CLASS
                   WHERE name='VersionedObjectControl'));

INSERT INTO Parameter (name, datatype, default, refOperationin,
                       refOperationout)
   VALUES ('', 'OIDt', null,  null,
                  (SELECT Operation (OID) FROM OPERATION
                   WHERE  name='getCurrentVersion'));

INSERT INTO Operation (name, specializ, visibility, static, refClass)
   VALUES ('getFirstVersion', null,  'public', null,
                  (SELECT Class(id) FROM CLASS
                   WHERE  name='VersionedObjectControl'));

INSERT INTO Parameter (name, datatype, default, refOperationin,
```

```
                        refOperationout)
    VALUES (’’, ’OIDt’, null,  null,
                  (SELECT Operation (OID) FROM OPERATION
                   WHERE name=’getFirstVersion’));

  INSERT INTO Operation (name, specializ, visibility, static, refClass)
    VALUES (’getLastVersion’, null, ’public’, null,
                     (SELECT Class(id) FROM CLASS
                      WHERE name=’VersionedObjectControl’));

  INSERT INTO Parameter (name, datatype, default, refOperationin,
                        refOperationout)
    VALUES (’’, ’OIDt’, null,  null,
                     (SELECT Operation (OID) FROM OPERATION
                      WHERE name=’getLastVersion’));

  INSERT INTO Operation (name, specializ, visibility, static, refClass)
    VALUES (’getNextVersionNumber’, null, ’public’, null,
                     (SELECT Class(id) FROM CLASS
                      WHERE name=’VersionedObjectControl’));

  INSERT INTO Parameter (name, datatype, default, refOperationin,
                        refOperationout)
    VALUES (’’, ’integer’, null, null,
                     (SELECT Operation (OID) FROM OPERATION
                      WHERE name=’getNextVersionNumber’));

  INSERT INTO Operation (name, specializ, visibility, static, refClass)
    VALUES (’getUserCurrentFlag’, null, ’public’, null,
                     (SELECT Class(id) FROM CLASS
                      WHERE name=’VersionedObjectControl’));

  INSERT INTO Parameter (name, datatype, default, refOperationin,
                        refOperationout)
    VALUES (’’, ’CHAR’, null,  null,
                     (SELECT Operation (OID) FROM OPERATION
                      WHERE name=’getUserCurrentFlag’));

  INSERT INTO Operation (name, specializ, visibility, static, refClass)
    VALUES (’getVersionCount’, null, ’public’, null,
                     (SELECT Class(id) FROM CLASS
                      WHERE name=’VersionedObjectControl’));

  INSERT INTO Parameter (name, datatype, default, refOperationin,
                        refOperationout)
    VALUES (’’, ’integer’, null, null,
                     (SELECT Operation (OID) FROM OPERATION
                      WHERE name=’getVersionCount’));

INSERT INTO Operation (name, specializ, visibility, static, refClass)
    VALUES (’setCurrentVersion’, null, ’public’, null,
                     (SELECT Class(id) FROM CLASS
                      WHERE name=’VersionedObjectControl’));

  INSERT INTO Operation (name, specializ, visibility, static, refClass)
    VALUES (’restore’, null, ’private’, null,
                     (SELECT Class(id) FROM CLASS
                      WHERE name=’VersionedObjectControl’));

  INSERT INTO Operation (name, specializ, visibility, static, refClass)
    VALUES (’setCurrentVersion’, null,  ’public’, null,
                     (SELECT Class(id) FROM CLASS
                      WHERE  name=’VersionedObjectControl’));

  INSERT INTO Parameter (name, datatype, default, refOperationin,
                        refOperationout)
    VALUES (’newVersionId’, ’OIDt’, null,
                     (SELECT Operation (OID) FROM OPERATION
                      WHERE name=’setCurrentVersion’),
                     null);

  INSERT INTO Operation (name, specializ, visibility, static, refClass)
    VALUES (’setFirstVersion’, null, ’public’, null,
```

```
                    (SELECT Class(id) FROM CLASS
                     WHERE name='VersionedObjectControl'));

   INSERT INTO Parameter (name, datatype, default, refOperationin,
                          refOperationout)
      VALUES ('first', 'OIDt', null,
                    (SELECT Operation (OID) FROM OPERATION
                     WHERE name='setFirstVersion'),
                    null);

   INSERT INTO Operation (name, specializ, visibility, static, refClass)
      VALUES ('setLastVersion', null, 'public', null,
                    (SELECT Class(id) FROM CLASS
                     WHERE name='VersionedObjectControl'));

   INSERT INTO Parameter (name, datatype, default, refOperationin,
                          refOperationout)
      VALUES ('last', 'OIDt', null,
                    (SELECT Operation (OID) FROM OPERATION
                     WHERE name='setLastVersion'),
                    null);

   INSERT INTO Operation (name, specializ, visibility, static, refClass)
      VALUES ('setUserCurrentFlag', null, 'public', null,
                    (SELECT Class(id) FROM CLASS
                     WHERE name='VersionedObjectControl'));

   INSERT INTO Parameter (name, datatype, default, refOperationin,
                          refOperationout)
      VALUES ('flag', 'CHAR', null,
                    (SELECT Operation (OID) FROM OPERATION
                     WHERE name='setUserCurrentFlag'),
                    null);

   INSERT INTO Operation (name, specializ, visibility, static, refClass)
      VALUES ('updateConfigurationCount', null, 'public', null,
                    (SELECT Class(id) FROM CLASS
                     WHERE name='VersionedObjectControl'));

   INSERT INTO Operation (name, specializ, visibility, static, refClass)
      VALUES ('updateVersionCount', null, 'public', null,
                    (SELECT Class(id) FROM CLASS
                     WHERE name='VersionedObjectControl'));

   INSERT INTO Operation (name, specializ, visibility, static, refClass)
      VALUES ('updateNextersionNumber', null, 'public', null,
                    (SELECT Class(id) FROM CLASS
                     WHERE name='VersionedObjectControl'));
```

XSLT para DDL

```
<?xml version="1.0"?> <xsl:stylesheet
xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0" >
    <xsl:output method="html" indent="yes"/>

    <xsl:template match="/">
            <html>
                <xsl:apply-templates/>
            </html>
    </xsl:template>

    <xsl:template match="ddlExtended">
            <head>
                <title> DDL Estendida </title>
            </head>
            <body bgcolor="white">
                <h2 align="center"> Linguagem de Definição de Dados do Modelo
                Temporal de Versões </h2>
                <hr width="100%" />
                <p />
```

```
            <h3>  <u> Classes normais </u> </h3>
            <xsl:apply-templates>
                <xsl:with-param name="tempVers">false</xsl:with-param>
            </xsl:apply-templates>
        <center> DDL estendida para classes normais </center>
            <p/>
            <xsl:call-template name="dados_adicionais_normais" />
            <h3> <u> Classes Temporais Versionadas </u> </h3>
            <xsl:apply-templates>
                <xsl:with-param name="tempVers">true</xsl:with-param>
            </xsl:apply-templates>
        <center> DDL estendida para classes temporais versionadas </center>
            <xsl:call-template name="dados_adicionais_tv" />
            <xsl:call-template name="dados_adicionais" />
        </body>
</xsl:template>

<xsl:template match="class">
    <xsl:param name="tempVers"/>
        <p align="left"/>
    <center> <table border="1"  width="70%"> <tr> <td>
            <xsl:apply-templates select="type" />
            <b> class </b>
            <xsl:value-of select="name"/>

            <xsl:if test="versions/@temporalVersion=$tempVers">
                [ <b> <xsl:value-of select="versions" /> </b> ]
            </xsl:if>
            <xsl:apply-templates select="inherit">
                <xsl:with-param name="tempVers" select="$tempVers" />
            </xsl:apply-templates>
            <xsl:apply-templates select="aggregateOf" >
                <xsl:with-param name="tempVers" select="$tempVers" />
            </xsl:apply-templates>
            <p/>
            (
        <xsl:apply-templates select="properties" >
                <xsl:with-param name="tempVers" select="$tempVers" />
            </xsl:apply-templates>
        <xsl:apply-templates select="relationship">
                <xsl:with-param name="tempVers" select="$tempVers" />
            </xsl:apply-templates>
        <xsl:apply-templates select="operations" />);
    </td> </tr> </table> </center>
</xsl:template>

<xsl:template match="visibility">
        <xsl:if test="@optional='true'">[</xsl:if>
        <xsl:apply-templates />]
</xsl:template>

<xsl:template match="type">
        <xsl:if test="@optional='true'">[</xsl:if>
        <xsl:apply-templates/>]
</xsl:template>

<xsl:template match="choice">
        <b> <xsl:value-of select="."/> </b>
        <xsl:if test="position()!=last()">|</xsl:if>
</xsl:template>

<xsl:template match="inherit">
    <xsl:param name="tempVers" />
        [<b> inherit </b>
        <xsl:if test="inheritBy/@temporalVersion!=$tempVers">
            [<xsl:value-of select="inheritedBy" />]
        </xsl:if>
        <xsl:value-of select="name" /> ]
    <xsl:if test="correspondence/@temporalVersion=$tempVers">
            <xsl:apply-templates select="correspondence" />
        </xsl:if>
</xsl:template>
```

```
<xsl:template match="correspondence">
        <b> correspondence </b>
        <xsl:if test="@choice='true'">(</xsl:if>
        <xsl:apply-templates/>)
</xsl:template>

<xsl:template match="aggregateOf">
    <xsl:param name="tempVers" />
    <br/>
    <xsl:call-template name="aggregateOf">
        <xsl:with-param name="tempVers" select="$tempVers" />
    </xsl:call-template>

    <xsl:if test="@multipleOccurrence='true'">
            <br />
            {,
            <xsl:if test="@optionalOccurrence='true'">[</xsl:if>
            <xsl:call-template name="aggregateOf">
                <xsl:with-param name="tempVers" select="$tempVers" />
            </xsl:call-template>
            <xsl:if test="@optionalOccurrence='true'">]</xsl:if>
            }
    </xsl:if>
</xsl:template>

<xsl:template name="temporal">
        [ <b>temporal </b> ]
</xsl:template>

<xsl:template match="by">
    [<xsl:apply-templates />]
</xsl:template>

<xsl:template name="aggregateOf">
    <xsl:param name="tempVers" />
    <xsl:if test="temporal/@temporalVersion=$tempVers">
        <xsl:call-template name="temporal" />
    </xsl:if>
    <b> aggregateOf </b>
    [<xsl:value-of select="multiplicity" />]
    <xsl:value-of select="name" />
    <xsl:apply-templates select="by" />
</xsl:template>

<xsl:template match="properties" >
    <xsl:param name="tempVers"/>
    [ <b> Properties </b>:
    <br />
    {<xsl:call-template name="static" />
    <xsl:if test="temporal/@temporalVersion=$tempVers">
        <xsl:call-template name="temporal" />
    </xsl:if>
    <xsl:apply-templates />}
    <xsl:if test="@multipleOccurrence='true'"   > + </xsl:if>]
</xsl:template>

<xsl:template name="static">
        [ <b> static </b> ]
</xsl:template>

<xsl:template match="attribute">
    <xsl:value-of select="name"/> :
    <xsl:value-of select="domain"  />
    [<b> default </b>
    <xsl:value-of select="defaultValue" /> ];
</xsl:template>

<xsl:template match="relationship">
    <xsl:param name="tempVers"/>
    <br/>
    [ <b> Relationships </b>:
    <br />
    {<xsl:if test="temporal/@temporalVersion=$tempVers">
```

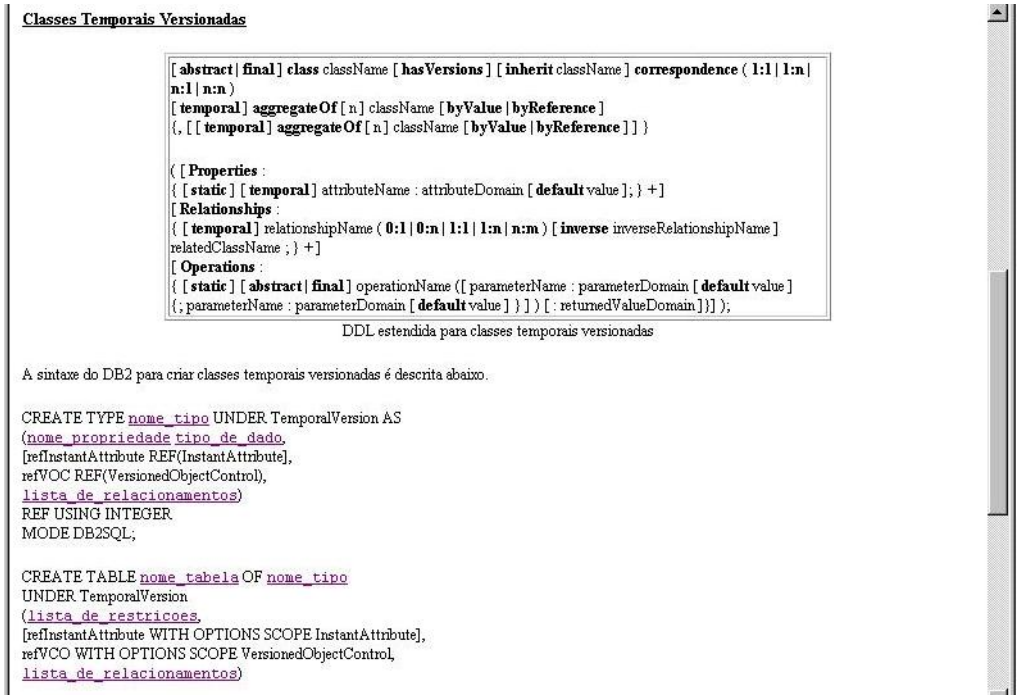
```
        <xsl:call-template name="temporal" />
    </xsl:if>
    <xsl:apply-templates />; }
    <xsl:if test="@multipleOccurrence='true'"   > + </xsl:if>]
</xsl:template>

<xsl:template match="cardinality">
    (<xsl:apply-templates/>)
</xsl:template>

<xsl:template match="inverse" >
    [ <b> inverse </b>
    <xsl:value-of select="name"/> ]
</xsl:template>

<xsl:template match="operations">
    <br />
       [ <b> Operations </b>:
       <br />
    {<xsl:call-template name="static" />
    <xsl:apply-templates select="type" />
    <xsl:value-of select="name" />
    <xsl:apply-templates select="parameter" />
    [ : <xsl:value-of select="return"/> ]}]
</xsl:template>

<xsl:template match="parameter">
    ([ <xsl:call-template name="parameter" />
    <xsl:if test="@multipleOccurrence='true'">
        <br />
        {;
        <xsl:call-template name="parameter" />
        }
    </xsl:if> ] )
</xsl:template>

<xsl:template name="parameter">
    <xsl:value-of select="name" />  :
    <xsl:value-of select="domain" />
    [<b> default </b>
    <xsl:value-of select="defaultValue" /> ]
</xsl:template>

<xsl:template name="dados_adicionais_normais">
    <p>
        Para criar as classes no DB2 é necessário utilizar as funcionalidades
        objeto-relacional do banco de dados.  Classes correspondem a tabelas tipadas,
        cuja sintaxe é apresentada abaixo.   Essa sintaxe é aplicada à classes normais,
        ou seja, aquelas que não lidam com tempo e versão.
        <p/><p/>
        CREATE TYPE <a href="#tipo"> <font face="Courier New,Courier">nome_tipo</font> </a>
            UNDER Object AS <br/>
        (<a href="#propriedade"> <font face="Courier New,Courier">nome_propriedade</font>
         </a>
         <xsl:text> </xsl:text>
         <a href="#tipo_dado"> <font face="Courier New,Courier">tipo_de_dado</font> </a>,
         <br/>
         <xsl:text> </xsl:text>
         <a href="#relacionamento"> <font face="Courier New,Courier">lista_de_relacionamentos
         </font> </a>) <br/>
        MODE DB2SQL; <br/>

        <p/>
        CREATE TABLE <a href="#tabela"> <font face="Courier New,Courier">nome_tabela</font>
                    </a>
            OF
            <xsl:text> </xsl:text>
                       <a href="#tipo"> <font face="Courier New,Courier">nome_tipo</font>
                       </a>  <br/>
            UNDER Object <br/>
             <xsl:text> </xsl:text>
        (<a href="#restricao"> <font face="Courier New,Courier">lista_de_restricoes</font>
         </a>, <br/>
```

```
         <xsl:text> </xsl:text>
         <a href="#relacionamento"> <font face="Courier New,Courier">lista_de_relacionamentos
         </font> </a>) <br/>
    </p>
</xsl:template>

    <xsl:template name="dados_adicionais_tv">
    <p>
        A sintaxe do DB2 para criar classes temporais versionadas é descrita abaixo.
        <p/><p/>
        CREATE TYPE <a href="#tipo"> <font face="Courier New,Courier">nome_tipo</font> </a>
            UNDER TemporalVersion AS <br/>
         <xsl:text> </xsl:text>
        (<a href="#propriedade"> <font face="Courier New,Courier">nome_propriedade</font>
         </a>
         <xsl:text> </xsl:text>
             <a href="#tipo_dado"> <font face="Courier New,Courier">tipo_de_dado</font>
             </a>,  <br/>
             [refInstantAttribute REF(InstantAttribute], <br/>
         refVOC REF(VersionedObjectControl), <br/>
         <xsl:text> </xsl:text>
         <a href="#relacionamento"> <font face="Courier New,Courier"> lista_de_relacionamentos
         </font> </a>) <br/>
         REF USING INTEGER <br/>
        MODE DB2SQL; <br/>

        <p/>
        CREATE TABLE <a href="#tabela"> <font face="Courier New,Courier">nome_tabela</font>
                     </a>
            OF
                  <xsl:text> </xsl:text>
            <a href="#tipo"> <font face="Courier New,Courier">nome_tipo</font> </a>  <br/>
            UNDER TemporalVersion <br/>
         <xsl:text> </xsl:text>
        (<a href="#restricao"> <font face="Courier New,Courier">lista_de_restricoes</font>
         </a>, <br/>
             [refInstantAttribute WITH OPTIONS SCOPE InstantAttribute], <br/>
         refVCO WITH OPTIONS SCOPE VersionedObjectControl, <br/>
         <xsl:text> </xsl:text>
         <a href="#relacionamento"> <font face="Courier New,Courier">lista_de_relacionamentos
         </font> </a>) <br/>
    </p>
</xsl:template>

<xsl:template name="dados_adicionais">
    <p/>
    <h3> <u> Dados Adicionais </u></h3> <p/>

    <center> <table border="1" width="100%">
        <tr bgcolor="#CCCCCC">
            <td><center> Opção </center></td>
            <td><center> Descrição </center></td>
            <td><center> Sintaxe </center></td>
        </tr>
        <tr>
            <a name="tabela"><td> nome_tabela </td></a>
            <td>nome da tabela tipada </td>
            <td> <center> - </center> </td>
        </tr>
        <tr>
            <a name="tipo"><td> nome_tipo </td></a>
            <td>nome dado ao tipo que será baseada a tabela </td>
            <td> <center> - </center> </td>
        </tr>
        <tr>
            <a name="propriedade"><td> nome_propriedade </td></a>
            <td> nome da propriedade </td>
            <td> <center> - </center> </td>
        </tr>
        <tr>
            <a name="tipo_dado"><td> tipo_dado </td></a>
            <td> tipo de dado aplicado a propriedade </td>
            <td> <center> - </center> </td>
```

```
            </tr>
            <tr>
                <a name="relacionamento"><td>lista_de_relacionamentos</td></a>
                <td> referências as colunas das tabelas relacionadas </td>
                <td>  nome_coluna REF(nome_tipo)</td>
            </tr>
            <tr>
                <a name="restricao"><td> lista_de_restricoes </td></a>
                <td> restrições impostas às colunas das tabelas  </td>
                <td>
                    <li> Valores não nulos:  <br/> <font face="Courier New,Courier"> nome_propriedade
                    </font> NOT NULL  </li>
                    <li> Valores <i>default</i>:
                        <br/> <font face="Courier New,Courier">  nome_propriedade </font>
                        WITH DEFAULT valor  </li>
                    <li> Check constraint:
                        <br/> CONSTRAINT <font face="Courier New,Courier">  nome_constraint
                        </font> CHECK
                        <font face="Courier New,Courier">  condicao </font> </li>
                </td>
            </tr>
        </table>    </center>

        <p/>
                <ul>
           <li> Após a criação da classe é necessário que os dados da mesma sejam inseridos
                nos metadados. </li>
           <li> Para atributos temporais é necessário a inclusão do relacionamento opcional
                <b>refInstantAttribute</b> na classe.  </li>
           <li> Para atributos cujos valores são conjuntos, observar as seguintes regras:</li>
                 <ul>
                       <li> se atributo temporal, incluir do relacionamento opcional
                            <b>refInstantAttribute</b> na classe;</li>
                       <li> se atributo não temporal, criar uma classe para armazenar os
                            valores do atributo e incluir relacionamento na classe origem.</li>
                  </ul>
             </ul>
        <p/>
    </xsl:template>
</xsl:stylesheet>
```

Documentação gerada para a DDL

FIGURA C.1 – Documentação gerada para a DDL



Classes Temporais Versionadas

[**abstract** | **final**] **class** className [**hasVersions**] [**inherit** className] **correspondence** (**1:1** | **1:n** | **n:1** | **n:n**)
[**temporal**] **aggregateOf** [n] className [**byValue** | **byReference**]
{, [[**temporal**] **aggregateOf** [n] className [**byValue** | **byReference**]] }

([**Properties** :
{ [**static**] [**temporal**] attributeName : attributeDomain [**default** value]; } +]
[**Relationships** :
{ [**temporal**] relationshipName (**0:1** | **0:n** | **1:1** | **1:n** | **n:m**) [**inverse** inverseRelationshipName] relatedClassName ; } +]
[**Operations** :
{ [**static**] [**abstract** | **final**] operationName ([parameterName : parameterDomain [**default** value] {; parameterName : parameterDomain [**default** value] }]) [: returnedValueDomain]}]);

DDL estendida para classes temporais versionadas

A sintaxe do DB2 para criar classes temporais versionadas é descrita abaixo.

```
CREATE TYPE nome_tipo UNDER TemporalVersion AS
(nome_propriedade tipo_de_dado,
[refInstantAttribute REF(InstantAttribute],
refVOC REF(VersionedObjectControl),
lista_de_relacionamentos)
REF USING INTEGER
MODE DB2SQL;

CREATE TABLE nome_tabela OF nome_tipo
UNDER TemporalVersion
(lista_de_restricoes,
[refInstantAttribute WITH OPTIONS SCOPE InstantAttribute],
refVCO WITH OPTIONS SCOPE VersionedObjectControl,
lista_de_relacionamentos)
```

FIGURA C.2 – Documentação gerada para a DDL

Dados Adicionais

Opção	Descrição	Sintaxe
nome_tabela	nome da tabela tipada	-
nome_tipo	nome dado ao tipo que será baseada a tabela	-
nome_propriedade	nome da propriedade	-
tipo_dado	tipo de dado aplicado a propriedade	-
lista_de_relacionamentos	referências as colunas das tabelas relacionadas	nome_coluna REF(nome_tipo)

lista_de_restricoes	restrições impostas às colunas das tabelas	• Valores não nulos:
		`nome_propriedade NOT NULL`
		• Valores *default*:
		`nome_propriedade WITH DEFAULT valor`
		• Check constraint:
		`CONSTRAINT nome_constraint CHECK condicao`

- Após a criação da classe é necessário que os dados da mesma sejam inseridos nos metadados.
- Para atributos temporais é necessário a inclusão do relacionamento opcional **refInstantAttribute** na classe.
- Para atributos cujos valores são conjuntos, observar as seguintes regras:
  - se atributo temporal, incluir do relacionamento opcional **refInstantAttribute** na classe;
  - se atributo não temporal, criar uma classe para armazenar os valores do atributo e incluir relacionamento na classe origem.

file:///D:/DDL.html#relacionamento Meu computador

FIGURA C.3 – Documentação gerada para a DDL

Anexo D Métodos implementados

Object

```
CREATE FUNCTION findVersion (entityId integer, classId integer) \
RETURNS integer \
LANGUAGE SQL \
NOT DETERMINISTIC \
EXTERNAL ACTION \
READS SQL DATA \
fnc: begin atomic \
    declare tvClass CHAR; \
    declare nomeClasse VARCHAR(30); \
    declare contador integer; \
    declare qtdEntidade integer; \
    set contador = 0; \
    set tvClass = (select tempVers from class1 where id=classId); \
    if tvClass = 'F' then \
       RETURN null; \
    else \
       set nomeClasse = getClassName (classId); \
       for row as select substr(tvOID, 1, posstr(tvOID, ',')-1) as ent, \
                         substr(substr(tvOID, posstr(tvOID, ',')+1), \
                                  posstr(substr(tvOID, posstr(tvOID, ',')-1), ',')) as clas \
                  from VersionedObjectControl do \
          if (integer(ent) = entityId) and (integer(clas) = classId) then \
             set contador = contador + 1; \
          end if; \
       end for; \
       if nomeClasse = 'VersionedObjectControl' and contador = 0 then \
          RETURN null; \
       else \
          set qtdEntidade = select count(tvOID) from nomeClasse \
             where integer(substr(tvOID, 1, posstr(tvOID, ',')-1))=entityId); \
          if qtdEntidade = 0 then \
             RETURN 1; \
          else \
             if (select refVersionedObjectControl from nomeClasse) = null then
                RETURN (select nextVersionNumber from VersionedObjectControl \
                        where tvOID = concat(string(entityId), ',',
                                     string(classId), ',', '0'); \
             else \
                if qtdEntidade = 1 then RETURN 2; \
             end if; \
       end if; \
    end if; \
end

CREATE FUNCTION getClassId (className varchar(30)) \
RETURNS integer \
LANGUAGE SQL \
NOT DETERMINISTIC \
EXTERNAL ACTION \
READS SQL DATA \
fnc: begin atomic \
    declare classId integer; \
    set classId = (select id from class where name = className); \
    RETURN classId; \
end

CREATE FUNCTION getClassName (classId integer) \
RETURNS varchar(30) \
LANGUAGE SQL \
NOT DETERMINISTIC \
EXTERNAL ACTION \
READS SQL DATA \
fnc: begin atomic \
    declare className varchar(30); \
    set className = (select name from class where id = classId); \
    RETURN className; \
```

```
end

CREATE FUNCTION getCorrespondence (classeAtual varchar(30), \
               className varchar(30)) \
RETURNS varchar(10) \
LANGUAGE SQL \
NOT DETERMINISTIC \
EXTERNAL ACTION \
READS SQL DATA \
fnc: begin atomic \
    declare corresp varchar(10); \
    set corresp = (select correspondencia \
                   from Extension e, Relationship r, Class c \
                   where e.refRelationship = r.ID and \
                         r.Name = className and \
                         r.refClass = c.ID and \
                         c.Name = classeAtual); \
    RETURN corresp; \
end

CREATE FUNCTION getCorrespondenceAsc (classeAtual varchar(30), \
               className varchar(30)) \
RETURNS varchar(10) \
LANGUAGE SQL \
NOT DETERMINISTIC \
EXTERNAL ACTION \
READS SQL DATA \
fnc: begin atomic \
    declare corresp varchar(10); \
    set corresp = (select substr(correspondencia, posstr(correspondencia, ':')+1) \
                   from Extension e, Relationship r, Class c \
                   where e.refRelationship = r.ID and \
                         r.Name = className and \
                         r.refClass = c.ID and \
                         c.Name = classeAtual); \
    RETURN corresp; \
end

CREATE FUNCTION getCorrespondenceAsc (classeAtual varchar(30), \
               className varchar(30)) \
RETURNS varchar(10) \
LANGUAGE SQL \
NOT DETERMINISTIC \
EXTERNAL ACTION \
READS SQL DATA \
fnc: begin atomic \
    declare corresp varchar(10); \
    set corresp = (select substr(correspondencia, 1, posstr(correspondencia, ':')-1) \
                   from Extension e, Relationship r, Class c \
                   where e.refRelationship = r.ID and \
                         r.Name = className and \
                         r.refClass = c.ID and \
                         c.Name = classeAtual); \
    RETURN corresp; \
end

CREATE FUNCTION getEntityId (entityName varchar(30)) \
RETURNS integer \
LANGUAGE SQL \
NOT DETERMINISTIC \
EXTERNAL ACTION \
READS SQL DATA \
fnc: begin atomic \
    declare entityId integer; \
    set entityId = (select id from entity where name = entityName); \
    RETURN entityId; \
end

CREATE FUNCTION getNickName (OID varchar(17)) \
RETURNS varchar(50) \
LANGUAGE SQL \
NOT DETERMINISTIC \
EXTERNAL ACTION \
```

```
READS SQL DATA \
fnc: begin atomic \
    declare apelido varchar(50); \
    set apelido = (select name from Name where tvOid = OID); \
    RETURN apelido; \
end

CREATE FUNCTION verifyEntityName (entName varchar(30)) \
RETURNS char \
LANGUAGE SQL \
NOT DETERMINISTIC \
EXTERNAL ACTION \
READS SQL DATA \
fnc: begin atomic \
    declare entity integer; \
    set entity = getEntityId (entName); \
    if entity is null \
       RETURN 'F'; \
    else \
       RETURN 'T'; \
    end if; \
end
```

TemporalVersion

```
CREATE FUNCTION getAscendant (classe varchar(30), OID varchar(17), \
                              onlyConfiguration char) \
RETURNS TABLE (ascendant varchar(17)) \
LANGUAGE SQL \
NOT DETERMINISTIC \
EXTERNAL ACTION \
READS SQL DATA \
fnc: begin atomic \
    if onlyConfiguration = 'F' the \
       RETURN (select value from InstantAttribute i, classe c \
               where i.property = 'ascendant' \
                     i.refInstantAttribute = c.tvOID and \
                     c.tvOID = OID); \
end

CREATE FUNCTION getDescendant (classe varchar(30), OID varchar(17), \
                               onlyConfiguration char) \
RETURNS TABLE (descendant varchar(17)) \
LANGUAGE SQL \
NOT DETERMINISTIC \
EXTERNAL ACTION \
READS SQL DATA \
fnc: begin atomic \
    if onlyConfiguration = 'F' the \
       RETURN (select value from InstantAttribute i, classe c \
               where i.property = 'descendant' \
                     i.refInstantAttribute = c.tvOID and \
                     c.tvOID = OID); \
end

CREATE FUNCTION getSuccessor (classe varchar(30), OID varchar(17), \
                              onlyConfiguration char) \
RETURNS TABLE (successor varchar(17)) \
LANGUAGE SQL \
NOT DETERMINISTIC \
EXTERNAL ACTION \
READS SQL DATA \
fnc: begin atomic \
    if onlyConfiguration = 'F' the \
       RETURN (select value from InstantAttribute i, classe c \
               where i.property = 'successor' \
                     i.refInstantAttribute = c.tvOID and \
                     c.tvOID = OID); \
    else \
       RETURN (select successor from InstantAttribute i, classe c \
```

```
               where i.property = 'successor' \
                     i.refInstantAttribute = c.tvOID and \
                     c.tvOID = OID and c.configuration = 'T'); \
end

CREATE FUNCTION isConfiguration (classe varchar(30), OID varchar(17)) \
RETURNS char \
LANGUAGE SQL \
NOT DETERMINISTIC \
EXTERNAL ACTION \
READS SQL DATA \
fnc: begin atomic \
    declare config char; \
    set config = (select configuration from classe \
                  where tvOID = OID); \
    RETURN config; \
end
```

VersionedObjectControl

```
CREATE PROCEDURE updateNextVersionNumber (OID varchar(17)) \
LANGUAGE SQL \
NOT DETERMINISTIC \
EXTERNAL ACTION \
READS SQL DATA \
fnc: begin atomic \
    update VersionedObjectControl \
    set nextVersionNumber = nextVersionNumber + 1 \
    where refVOC = OID; \
end

CREATE PROCEDURE VersionedObjectControl (entityId integer, classId integer, \
                                         configCount integer, currentV varchar(17), \
                                         firstV varchar (17), lastV varchar (17), \
                                         nextVNumber integer, userCurrentFlag char, \
                                         vCount integer) \
LANGUAGE SQL \
NOT DETERMINISTIC \
EXTERNAL ACTION \
READS SQL DATA \
fnc: begin atomic \
    declare qtdObjeto integer; \
    set qtdObjeto = (select count(*) from VersionedObjectControl \
                     where integer(substr(tvOID, 1, posstr(tvOID, ',')-1)) = entityId and \
                     integer(substr(substr(tvOID, posstr(tvOID, ',')+1),  \
                     posstr(substr(tvOID, posstr(tvOID, ',')-1), ','))) = classId); \
    if qtdObjeto = 0 then \
       insert into VersionedObjectControl (tvOID, alive, configurationCount, \
                                           currentVersion, firstVersion , \
                                           lastVersion, nextVersionNumber, \
                                           userCurrentFlag, versionCount integer) \
              values (concat(rtrim(char(entityId)), ',', rtrim(char(classId)), ','), 'null') \
                      'T', configCount, currentV, firstV , lastV, nextVNumber, \
                      userCurrentFlag, vCount); \
    end if; \
end

CREATE PROCEDURE VersionedObjectControl (entityId integer, classId integer, \
                                         currentV varchar(17), firstV varchar (17), \
                                         lastV varchar (17)) \
LANGUAGE SQL \
NOT DETERMINISTIC \
EXTERNAL ACTION \
READS SQL DATA \
fnc: begin atomic \
    declare qtdObjeto integer; \
    set qtdObjeto = (select count(*) from VersionedObjectControl \
                     where integer(substr(tvOID, 1, posstr(tvOID, ',')-1)) = entityId and \
                           integer(substr(substr(tvOID, posstr(tvOID, ',')+1), \
                           posstr(substr(tvOID, posstr(tvOID, ',')-1), ','))) = classId); \
```

```
    if qtdObjeto = 0 then \
       insert into VersionedObjectControl (tvOID, alive, configurationCount, \
                                           currentVersion, firstVersion , \
                                           lastVersion, nextVersionNumber, \
                                           userCurrentFlag, versionCount integer)
              values (concat(rtrim(char(entityId)), ',', rtrim(char(classId)), ','), 'null') \
                      'T', 0, currentV, firstV , lastV, 3, userCurrentFlag, 2); \
    end if; \
end
```

Bibliografia

[AGR 91] AGRAWAL, R.; BUROFF, S. J.; GEHANI, N. H.; SHASHA, D. Object versioning in ode. In: INTERNATIONAL CONFERENCE ON DATA ENGINEERING, 7., 1991, Tokyo. **Proceedings...** Piscataway: IEEE, 1991. p.446–455.

[ALT 2001] ALTOVA. **XML spy 4.1 software suite**. Disponível em: <http://www.xmlspy.com>. Acesso em: dez. 2001.

[ANT 97] ANTUNES, D. C.; HEUSER, C. A.; EDELWEISS, N. TempER: uma abordagem para modelagem de banco de dados. **Revista de Informática Teórica e aplicada**, Porto Alegre, v.4, n.1, p.49–85, 1997.

[BAR 2001] BARKMEYER, E. J.; LUBELL, J. **XML representation of EXPRESS models and data**. Disponível em: <http://www.mel.nist.gov/msidlibrary/doc/xse2001.pdf>. Acesso em: dez. 2001.

[BAR 99] BARU, C. K. XViews: XML views of relational schemas. In: WORKSHOP ON DATABASE AND EXPERT SYSTEMS APPLICATIONS, 10., 1999, Florence, Italy. **Proceedings...** Piscataway: IEEE Computer Society, 1999. p.700–705. Disponível em: <http://www.citeseer.nj.nec.com/252703.html>. Acesso em: dez. 2000.

[BIE 98] BIELIKOVÁ, M.; NÁVRAT, P. Modelling versioned hypertext documents. In: SYSTEM CONFIGURATION MANAGEMENT, ECOOP'98 SCM-8 SYMPOSIUM, 1998, Brussels. **Proceedings...** Berlin: Springer, 1998. p.188–197. (Lecture Notes in Computer Science, v.1439).

[BOU 2001] BOURRET, R. **Mapping DTD's to databases**. Disponível em: <http://www.xml.com/pub/a/2001/05/09/dtdtodbs.html>. Acesso em: dez. 2001.

[BUR 2000] BURKE, P. XML y SQL server 2000. **SQL Server Magazine**, Barcelona, n.1, Septiembre 2000. Disponível em: <http://www.w2000mag.com/sqlmag/atrasados/01_sep00/articulos/XML_1.htm>. Acesso em: dez. 2000.

[CAR 99] CAREY, M. J.; CHAMBERLIN, D. D.; NARAYANAN, S.; VANCE, B.; DOOLE, D.; RIELAU, S.; SWAGERMAN, R.; MATTOS, N. M. O-O, what have they done to DB2. In: INTERNATIONAL CONFERENCE ON VERY LARGE DATA BASES, VLDB, 25., 1999, Edinburgh, Scotland. **Proceedings...** San Fransisco: Morgan Kaufmann, 1999. p.542–553.

[CAR 2000] CAREY, M. J.; FLORESCU, D.; IVES, Z. G.; LU, Y.; SHANMUGASUNDARAM, J.; SHEKITA, E. J.; SUBRAMANIAN, S. N. XPERANTO: publishing object-relational data as XML. In: INTERNATIONAL WORKSHOP ON THE WEB AND DATABASES (WebDB), 3., 2000, Dallas. **Proceedings...** [S.l.: s.n.], 2000. p.105–110. Disponível em: <http://www.cs.wisc.edu/~jai/papers/XperantoOverview.pdf>. Acesso em: dez. 2000.

[CHA 98] CHA, M. **DB2 universal database**. Chicago: Wrox Press, 1998.

[CHR 2000] CHRISTOPHIDES, V.; CLUET, S.; SIMÈON, J. On wrapping query languages and efficient XML integration. In: ACM SIGMOD INTERNATIONAL CONFERENCE ON MANAGEMENT OF DATA, 2000, Dallas, Texas. **Proceedings...** New York: ACM Press, 2000. p.141–152.

[CON 98] CONRADI, R.; WESTFECHTEL, B. Version models for software configuration management. **ACM Computing Surveys**, New York, v.30, n.2, p.232–282, June 1998.

[DAT 97] DATE, C. J.; DARWEN, H. **A guide to SQL standard**. New York: Addison-Wesley, 1997.

[EDE 94] EDELWEISS, N.; OLIVEIRA, J. P. M. **Modelagem de aspectos temporais de sistemas de informação**. Recife: Universidade Federal de Pernambuco, 1994. (Livro texto da IX Escola de Computação, 24 a 31 de Julho).

[ELM 2000] ELMASRI, R.; NAVATHE, S. B. **Fundamentals of database systems.** 3th ed. New York: Addison-Wesley, 2000.

[FAN 2001] FAN, W.; KUPER, G. M.; SIMÉON, J. A unified constraint model for XML. In: INTERNATIONAL WORLD WIDE WEB CONFERENCE, 10., 2001, Hong Knong. **Proceedings...** New York: ACM Press, 2001. p.179–190. Disponível em: <http://citeseer.nj.nec.com/379774.html>. Acesso em dez. 2001.

[FAN 2000] FAN, W.; SIMÉON, J. Integrity constraints for XML. In: ACM SIGMOD-SIGACT-SIGART SYMPOSIUM ON PRINCIPLES OF DATABASE SYSTEMS, 19., 2000, Dallas, Texas. **Proceedings...** New York: ACM Press, 2000. p.23–34. Disponível em: <http://www.cis.upenn.edu/~wfan/wefein/publication.html>. Acesso em dez. 2000.

[FUH 99] FUH, Y.-C.; DESSLOCH, S.; CHEN, W.; MATTOS, N. M.; TRAN, B. T.; LINDSAY, B. G.; DEMICHEL, L.; RIELAU, S.; MANNHAUPT, D. Implementation of SQL3 structured types with inheritance and value substitutability. In: INTERNATIONAL CONFERENCE ON VERY LARGE DATA BASES,

VLDB, 25., 1999, Edinburgh, Scotland. **Proceedings...** San Fransisco: Morgan Kaufmann, 1999. p.565–574.

[GOL 95] GOLENDZINER, L. G. **Um modelo de versões para banco de dados orientados a objetos**. 1995. Tese (Doutorado em Ciência da Computação) — Instituto de Informática, UFRGS, Porto Alegre.

[GOL 95a] GOLENDZINER, L. G.; SANTOS, C. S. Versions and configurations in object-oriented database systems: a uniform treatment. In: INTERNATIONAL CONFERENCE ON MANAGEMENT OF DATA, COMAD, 7., 1995, Pune, India. **Proceedings...** New Delhi: Tata McGraw-Hill, 1995. p.18–37.

[GOL 95b] GOLENDZINER, L. G.; SANTOS, C. S. Uma abordagem multinível para suporte a versões em bancos de dados orientados a objetos. In: SEMINARIO INTEGRADO DE SOFTWARE E HARDWARE, SEMISH, 22., 1995, Canela, RS. **Anais...** Porto Alegre: Instituto de Informatica: UFRGS, 1995. v.2, p.1127–1138.

[IBM 2001] IBM. **Application development guide**: version 7. Armonk, NY: IBM, 2001. Manual de referência.

[IBM 2001a] IBM. **SQL reference**: version 7. Armonk, NY: IBM, 2001. Manual de referência.

[IBM 2001b] IBM. **DB2 release notes**: version 7.2. Armonk, NY: IBM, 2001. Manual de referência.

[KAY 2000] KAY, M. **XSLT programmer's reference**. Chicago: Wrox Press, 2000.

[KUR 2000] KURAMITSU, K.; SAKAMURA, K. Distributed object-oriented schema to share XML-based catalogs among business. In: INTERNATIONAL CONFERENCE ON WEB INFORMATION SYSTEMS ENGINEERING, WISE, 1., 2000. **Proceedings...** Piscataway: IEEE, 2000. v.1, p.87–96.

[MOE 94] MOELLER, H. Versioning structured technical documentation. In: WORKSHOP ON VERSIONING IN HYPERTEXT SYSTEMS, ACM EUROPEAN CONFERENCE ON HYPERMEDIA TECHNOLOGY, ECHT, 1994, Edinburgh. **Proceedings...** [S.l.: s.n.], 1994. Disponível em: <http://cs-people.bu.edu/dgd/workshop/moeller.html>. Acesso em: dez. 2000.

[MOR 2001] MORO, M. M. **Modelo temporal de versões**. 2001. Dissertação (Mestrado em Ciência da Computação) — Instituto de Informática, UFRGS, Porto Alegre.

[MOR 2001a] MORO, M. M.; GELATTI, P. C.; GOMES, C. H. P.; ROSSETTI, L. L. F.; ZAUPA, A. P. **Linguagem de consulta do TVM**. Porto Alegre: PPGC, UFRGS, 2001. (RP-308).

[MOR 2001b] MORO, M.; SAGGIORATO, S.; EDELWEISS, N.; SANTOS, C. Adding time to an object-oriented versions model. In: INTERNATIONAL CONFERENCE ON DATABASE AND EXPERT SYSTEMS APPLICATIONS, DEXA, 12., 2001, Munich, Germany. **Proceedings...** Berlin: Springer, 2001. p.805–814.

[MUE 2000] MUENCH, S. **Building oracle xml applications**. Cambridge, MA: O'Reilly, 2000. p.275–309. Disponível em: <http://www.oreilly.com/catalog/orxmlapp/chapter/ch07.html>. Acesso em: maio 2001.

[NAV 89] NAVATHE, S. B.; AHMED, R. A temporal relational model and a query language. **Information Sciences**, Amsterdam, v.49, n.1–3, p.147–175, Oct.-Dec. 1989.

[OAS 2001] OASIS XSLT/XPath Conformance Technical Committee. **XSLT/XPath Conformance**. Billerica, MA: OASIS, 2001. Disponível em: <http://www.oasis-open.org/committees/xslt/index.shtml#overview>. Acesso em: maio 2001.

[ORA 2001] ORACLE Technology Network. Oracle XSQL Pages and the XSQL Servlet. Redwood Shores, CA: ORACLE Coorporation, 2001. Disponível em: <http://technet.oracle.com/tech/xml/xsql_servlet/>. Acesso em: dez. 2001.

[RIV 98] RIVERS-MOORE, D. **XML and EXPRESS as schema definition languages**. Wiltshire, UK: RivCom Limited, 1998. White Paper, Disponível em: <http://www.rivcom.com/k0603.html>. Acesso em: abr. 2001.

[ROS 91] ROSE, E.; SEGEV, A. TOODM: a temporal object-oriented data model with temporal constraints. In: INTERNATIONAL CONFERENCE ON THE ENTITY-RELATIONSHIP APPROACH, ER, 10., 1991, San Mateo, CA. **Proceedings...** [S.l]: ER Institute, 1991. p.205–229.

[ROS 2001] ROSSETTI, L. L. F.; GOMES, C. H. P.; SANTOS, C. S.; EDELWEISS, N. **XTVQL - linguagem de consulta do TVM em XML**. Disponível em: <http://www.inf.ufrgs.br/~lialda>. Acesso em: dez. 2001.

[SAG 99] SAGGIORATO, S. M. **Mapeamento de esquemas orientados a objeto com versões para esquemas objeto-relacionais**. 1999. Dissertação (Mestrado em Ciência da Computação) — Instituto de Informática, UFRGS, Porto Alegre.

[SAR 90] SARDA, N. L. Extensions to SQL for historical databases. **IEEE Transactions on Knowledge and Data Engineering**, Piscataway, v.2, n.2, p.220–230, June 1990.

[SHI 99] SHIMURA, T.; YOSHIKAWA, M.; UEMURA, S. Storage and retrieval of XML documents using object-relational databases. In: INTERNATIONAL CONFERENCE ON DATABASE AND EXPERT SYSTEMS APPLICATIONS (DEXA'99), 10., 1999, Florence, Italy. **Proceedings...** Berlin: Springer, 1999. p.206–217. Disponível em: <http://www.citeseer.nj.nec.com/shimura99storage.html>. Acesso em: dez. 2000.

[SNO 99] SNODGRASS, R. **Developing time-oriented database applications in SQL**. San Fransisco: Morgan Kaufmann, 1999.

[TAN 93] TANSEL, A.; CLIFFORD, J.; GADIA, S.; JAJODIA, S.; SEGEV, A.; SNODGRASS, R. **Temporal databases**: theory, design, and implementation. San Francisco: Benjamin Cummings, 1993. (Database Systems and Applications Series).

[W3C 2000] W3C. **Extensible markup language (XML) 1.0.** 2nd ed. Cambridge, MA: W3C, 2000. W3C Recomendation.

[W3C 2001] W3C. **XSL transformations (XSLT) version 1.1**. Cambridge, MA: W3C, 2001. W3C Working Draft.

[W3C 2001a] W3C. **XML schema part 0**: primer. Cambridge, MA: W3C, 2001. W3C Recomendation.