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

Extensão do Padrão ODMG para Suportar Tempo e Versões

por

PAÔLA CRISTINA GELATTI

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

Prof. Clesio Saraiva do Santos
Orientador

Profa. Nina Edelweiss
Co-orientadora

Porto Alegre, agosto de 2002.

CIP - CATALOGAÇÃO NA PUBLICAÇÃO

Gelatti, Paôla Cristina
Extensão do Padrão ODMG para Suportar Tempo e Versões / por Paôla Cristina Gelatti. – Porto Alegre: PPGC da UFRGS, 2002.
133 f.: il.

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

1. Banco de Dados Orientado a Objetos. 2. Modelo de Versões. 3. Padrão ODMG. 4. Modelo Temporal. I. Santos, Clesio Saraiva dos. II. Edelweiss, Nina. III. Título.

Agradecimentos

A realização deste trabalho contou com a participação de várias pessoas que acreditaram na minha capacidade para realizar o mesmo e me apoiaram durante todo o tempo.

Primeiramente, eu gostaria de agradecer ao Prof. Clesio, meu orientador, pela tranqüilidade transmitida e por me conduzir pelo caminho certo sempre que eu chegava em sua sala com uma lista enorme de dúvidas, como também a Profa. Nina pela excelente co-orientação.

Agradeço a todos os meus colegas do grupo de pesquisa pelo auxílio, agradecendo em especial a minha colega Mirella, a quem eu freqüentemente procurava para discutir alguma dúvida.

Agradeço ao PPGC pela oportunidade de realizar esse curso de mestrado.

Quero agradecer a minha mãe, a minha irmã e ao meu irmão por representarem o meu porto seguro, tendo sido isso fundamental para superar os momentos difíceis. Também por entenderem as várias vezes em que eu ficava "muito" irritada por motivos pequenos, quase inexistentes.

E por fim, agradeço aos meus amigos e amigas pelos momentos de descontração que me faziam esquecer as preocupações com a dissertação, ajudando-me a repor as energias para seguir em frente.

Sumário

Lista de Abreviaturas

BD	Banco de Dados
CAD	*Computer Aided Design*
CASE	*Computer Aided Software Engineering*
DDL	*Data Definition Language*
LPOO	Linguagens de Programação Orientada a Objetos
ODBMS	*Object Database Management System*
ODL	*Object Definition Language*
ODMG	*Object Data Management Group*
ODMS	*Object Data Management System*
OID	*Object Identificator*
OIF	*Object Interchange Format*
OMG	*Object Management Group*
OML	*Object Manipulation Language*
OQL	*Object Query Language*
SGBD	Sistema Gerenciador de Banco de Dados
SQL	*Structured Query Language*
TTF	Tempo de Transação Final
TTI	Tempo de Transação Inicial
TVF	Tempo de Validade Final
TVI	Tempo de Validade Inicial
TVM	*Temporal Version Model*
TV_ODL	*Temporal Version Object Definition Language*
TV_ODMG	*Temporal Version extension to Object Data Management Group standart*
TV_OM	*Temporal Version Object Model*
TV_OQL	*Temporal Version Object Query Language*
UML	*Unified Modeling Language*

Lista de Figuras

Lista de Tabelas

Resumo

Este trabalho apresenta uma extensão do padrão ODMG para o suporte ao versionamento de objetos e características temporais. Essa extensão, denominada TV_ODMG, é baseada no Modelo Temporal de Versões (TVM), que é um modelo de dados orientado a objetos desenvolvido para armazenar as versões do objeto e, para cada versão, o histórico dos valores dos atributos e dos relacionamentos dinâmicos. O TVM difere de outros modelos de dados temporais por apresentar duas diferentes ordens de tempo, ramificado para o objeto e linear para cada versão. O usuário pode também especificar, durante a modelagem, classes normais (sem tempo e versões), o que permite a integração desse modelo com outras modelagens existentes.

Neste trabalho, os seguintes componentes da arquitetura do padrão ODMG foram estendidos: o Modelo de Objetos, a ODL (*Object Definition Language*) e a OQL (*Object Query Language*). Adicionalmente, foi desenvolvido um conjunto de regras para o mapeamento do TV_ODMG para o ODMG a fim de permitir o uso de qualquer ODBMS para suportar a extensão proposta.

Palavras-chave: ODMG, Banco de Dados Orientado a Objetos, Modelo Temporal, Modelo de Versões.

TITLE: “ODMG STANDARD EXTENSION TO SUPORT TIME AND VERSIONS”

Abstract

This work presents an extension to the ODMG standard for supporting object versioning and temporal features. This extension, called TV_ODMG, is based on the Temporal Versions Model (TVM), which is an Object Oriented Data Model developed to store the object versions and, for each version, the history of its dynamic attributes and relationships values. TVM differs from other temporal data models by presenting two different time orders, branched time for the objects and linear time for each version. The user can also specify normal classes (without time and version) during the modeling, which allows the integration with existing specifications.

In this work, the following components of ODMG architecture have been extended: the Object Model, the ODL (Object Definition Language), and the OQL (Object Query Language). Additionally, it was developed a set of rules for mapping TV_ODMG to ODMG in order to allow the use of any ODBMS to support the proposed extension.

Keywords: ODMG, Object-Oriented Databases, Temporal Model, Versions Model.

1 Introdução

Atualmente, muitas aplicações como, por exemplo, CASE, CAD, automação de escritórios, bancos de dados históricos e hipermídia necessitam que sejam armazenados não só um estado de uma entidade, mas diversos, representando diferentes estágios da entidade em tempos distintos ou sob pontos de vista diferentes. Dentro deste contexto, há dois conceitos que se tornam importantes: dimensão temporal e versão.

Os bancos de dados temporais [SNO 2000, TAN 93] fornecem uma história completa de todas as mudanças ocorridas na base de dados e associam aos dados o tempo em que ocorrem essas alterações. Isso permite que se tenha o estado corrente da base de dados, o estado no passado e o estado que poderá ter no futuro. Para isso, podem ser definidas duas dimensões temporais: o tempo de transação (o momento em que uma informação é registrada no banco de dados) e o tempo de validade (tempo em que o fato é válido na realidade modelada).

Em relação ao conceito de versão, há diversas pesquisas sobre o versionamento de esquema [GAN 94, DOU 96, DOU 97, FRA 2000, BEL 96, OGA 2001, GAN 99, OGA 99, GAL 99] e versionamento de objetos [GOL 95, CEL 90, SAC 95, BER 97]. Em especial, Golendziner em [GOL 95] define um modelo de versões que propõe a incorporação de conceitos e mecanismos que suportam a definição e a manipulação de objetos, versões de objetos e configurações em um modelo de dados orientado a objetos. A característica que diferencia essa proposta das demais é a presença de versões em diferentes níveis da hierarquia de tipos/classes, pois normalmente é possível ter versões apenas no nível mais especializado da hierarquia. Essa peculiaridade torna o modelo mais rico e permite representações mais naturais da realidade. Moro [MOR 2001a] estende essa proposta no Modelo Temporal de Versões (TVM - *Temporal Version Model*). Esse trabalho adiciona características temporais ao modelo de versões, permitindo que seja armazenado o histórico e a evolução dos dados dos objetos ou versões. O TVM difere de outros modelos de dados temporais por apresentar duas diferentes ordens de tempo, ramificado para o objeto e linear para cada versão.

Há cerca de uma década atrás, havia carência em relação a padronização de bancos de dados orientados a objetos, quando então, surgiu o padrão ODMG para suprir essa necessidade. Esse padrão desenvolveu um conjunto de especificações com o propósito de possibilitar que os desenvolvedores escrevam aplicações portáveis. O padrão define os seguintes componentes: um modelo de objetos, uma linguagem de definição de objetos (ODL), uma linguagem de consulta (OQL) e *bindings* com linguagens de programação orientadas a objetos [CAT 2000]. Há várias pesquisas envolvendo esse padrão [TOR 2001, BUS 94a, BUS 94b, GAR 97, TES 97, CHA 97, CLA 98a, CLA 98b, CLA 99, DUM 98, DUM 99c, SAM 2000], como também há vários bancos de dados que o suportam, entre eles: Jasmine, JYD Object Database, Titanium, ObjectStore, BSF, Objectivity, Poet, JavaBlend, EyeDB, Orient, Versant, FastObjects, Matisse e Ozone [BAR 2001, ODM 2001]. Como, atualmente, muitas aplicações necessitam usar os conceitos de versão e tempo, então seria importante que esses estivessem definidos dentro do padrão ODMG.

Na literatura, há trabalhos que estendem o padrão com versões de esquema [GRA 2002, GRA 99, GRA 2000, RIE 99] ou com tempo [TOO 99, KAK 96a, KAK 96b, KAK 96c, KAK 96d, BER 98, DUM 99a, DUM 99b, DUM 2001, FEG 98]. A maioria das extensões temporais suporta as duas dimensões, tempo de transação e de validade, associando o tempo tanto ao nível de classes quanto ao de propriedades. Essas propostas

estendem um ou mais componentes da arquitetura ODMG. Porém, não foi encontrada uma extensão que suportasse tanto versões de objetos quanto as dimensões temporais, de tempo de transação e de validade.

Este trabalho define uma extensão para o padrão ODMG, denominada TV_ODMG (*Temporal Version extension to Object Data Management Group standart)*, que suporta as características de versões e tempo definidas no TVM. O Modelo de Objetos, a ODL e a OQL são estendidos. Adicionalmente, para não ser obrigatório o uso de um ODBMS específico para suportar a extensão proposta, foi desenvolvido um conjunto de regras para o mapeamento do TV_ODMG para o ODMG.

A organização deste trabalho apresenta-se da seguinte forma: a seção 2 demonstra como o Modelo Temporal de Versões trata o tempo e as versões; a seção 3 resume os conceitos do padrão ODMG; a seção 4, apresenta o TV_ODMG, que é a extensão proposta para que o padrão suporte tempo e versões; a seção 5 mostra as regras para o mapeamento do TV_ODMG para o ODMG; a seção 6 ilustra o estudo de caso; e a seção 7 apresenta as conclusões. Além disso, há um anexo que apresenta a descrição completa das classes da hieraquia de classes básicas do TV_ODMG e outro anexo que possui a descrição dos métodos dessas classes.

2 Modelo Temporal de Versões

O conceito de versão possibilita que se tenha mais de uma versão (alternativa) de um mesmo objeto. Esse conceito pode ser usado, por exemplo, em aplicações de engenharia para armazenar as alternativas de um projeto [KHO 94]. O Modelo Temporal de Versões proposto em [MOR 2001a] estende o Modelo de Versões de [GOL 95], permitindo representar toda a história dos dados da aplicação.

Um objeto que possui versões é denominado objeto versionado. Entre as suas versões, há sempre uma versão corrente que, se não for determinada pelo usuário, é a mais recente (a última a ser criada), utilizada sempre que o usuário mandar uma mensagem para o objeto versionado sem especificar uma versão.

As versões de um mesmo objeto versionado estão ligadas por um relacionamento de derivação, constituindo uma hierarquia de derivação. Essa hierarquia possui o formato de um grafo acíclico dirigido. Uma versão pode ser criada a partir da derivação de outra existente ou de um objeto temporal versionável sem versões, sendo que nesse último caso, o objeto que deu origem à derivação torna-se a primeira versão, e o objeto temporal versionável, gerado pela derivação - a segunda versão; além disso, é também gerado um objeto versionado, ao qual as duas versões pertencerão. As próximas versões são criadas com novas derivações a partir de uma ou mais dessas versões. Cada uma das versões pode estar associada a apenas um objeto versionado. Assim como os objetos sem versões, as versões e os objetos versionados também possuem OID.

Com a adição da dimensão temporal, o histórico e a evolução dos dados dos objetos ou versões são armazenados. Nesse caso, cabe ao usuário definir quais dados terão suas informações temporais armazenadas.

2.1 Representação Temporal no TVM

O TVM possibilita a associação do tempo a objetos, versões, objetos versionados e propriedades (atributos e relacionamentos). Entre as características do modelo em relação ao tempo estão:

- Bitemporalidade - o modelo suporta o tempo de transação e o tempo de validade. Esses tempos são armazenados nos rótulos pré-definidos: *vTimei*, *vTimef*, *tTimei* e *tTimef*, os quais correspondem respectivamente aos tempos de validade inicial e final e aos de transação inicial e final. Esses rótulos são implícitos.
- Variação temporal discreta.
- Duas ordens no tempo - tempo ramificado para um objeto, devido às diferentes linhas de tempo de cada versão que são geradas da linha do objeto, pois muitas versões do mesmo objeto podem coexistir; e tempo linear para cada versão, devido ao armazenamento do histórico dos dados.

As propriedades de uma classe podem ser definidas como estáticas (não possuem o histórico da variação de seus valores armazenado) ou temporalizadas (possuem o histórico armazenado). A definição de uma propriedade como temporal ou não é responsabilidade do usuário.

Todo objeto possui o atributo temporalizado pré-definido *alive* associado, o qual recebe o valor *true* no instante da criação do objeto, sendo associado a ele seu tempo de validade inicial que é igual ao tempo de transação inicial, e os tempos finais que ficam, inicialmente, em aberto. No momento da exclusão lógica, é associado ao valor *true* o tempo de validade final que corresponde ao instante da exclusão, e o atributo *alive* recebe *false*, cujo valor também tem seus respectivos tempos associados. O TVM não permite ao usuário alterar diretamente o tempo de validade inicial ou final de um objeto. Esses tempos serão alterados pelo sistema quando o objeto for criado, excluído ou reativado pelo usuário.

2.1.1 Regras de Integridade Temporal

Algumas regras foram definidas para manter a integridade em relação aos tempos de vida dos objetos, versões e objetos versionados. De acordo com essas regras, o rótulo de tempo associado às versões deve estar contido no tempo de vida do objeto versionado ao qual pertencem, e o rótulo de tempo associado às variações das propriedades temporalizadas de uma versão deve estar contido no tempo de vida da versão.

Foram definidas também algumas regras em relação à inserção e atualização de valores de propriedades temporalizadas, e à exclusão de objetos.

Toda informação que for inserida (primeira definição de um valor para uma propriedade) em uma base de dados bitemporal deve receber seus tempos de transação e de validade. O sistema é responsável pelo tempo de transação. O tempo de transação inicial é fornecido quando a informação é registrada na base de dados. O tempo de transação final, dessa informação, é inicialmente *null*. Esse tempo apenas receberá algum valor quando for necessária uma alteração no tempo de validade do valor da propriedade que está registrado (por exemplo, em uma exclusão lógica para finalizar os tempos de validade e corrigi-los ou em uma inserção de novo valor para uma propriedade que necessite corrigir os tempos anteriores). O usuário pode fornecer o tempo de validade inicial e o final. Quando esse último não for informado, ele receberá *null*, sendo esse valor válido até que outra informação seja definida ou o objeto seja excluído.

Nunca há perda de informações na atualização de dados. De acordo com as regras estabelecidas para atualizações, a tabela 2.1 mostra um exemplo de uma atualização na base de dados:

- Inicialmente foi registrada, em 01/12/2000, uma informação com valor A, tendo para validade inicial 01/01/2001 e validade final em aberto, ou seja, com valor *null*. A transação final também recebe *null*.
- Em 10/02/2001, foi atualizado o valor da informação para B, e foi fornecido um valor para sua validade final em 30/04/2001. Assim, o valor da informação atual é copiado (A) e seu TVF recebe o TVI do novo valor (B) menos 1. O TTF do primeiro valor A recebe o momento da transação atual menos 1, pois aquelas informações sobre os tempos do valor (A) estão corretas apenas até este instante. Quando TTF é igual a *null*, indica que aquilo está correto para sempre; dessa forma, tem-se que até este instante (09/02/2001) se acreditava que o A valeria para sempre.

- Em 10/02/2001, foi também fornecido o valor C para ter validade a partir de 01/05/2001.
- Em 15/03/2001, notou-se que deveria ser feita uma correção no valor válido entre 10/02/2001 e 02/03/2001, sendo que o valor correto para este período é D e não B. Assim, foi finalizado o TTF de B, pois essa informação deixou de estar correta nessa data; foi inserido o valor D com seus respectivos tempos de validade; e foi copiado o valor B corrigindo seus tempos de validade.

TABELA 2.1 - Exemplo de atualização

Informação	TVI	TVF	TTI	TTF
A	01/01/2001	null	01/12/2000	**09/02/2001**
A	01/01/2001	09/02/2001	10/02/2001	null
B	10/02/2001	30/04/2001	10/02/2001	**14/03/2001**
C	01/05/2001	null	10/02/2001	null
D	10/02/2001	02/03/2001	15/03/2001	null
B	03/03/2001	30/04/2001	15/03/2001	null

A exclusão de objetos pode ser feita logicamente ou fisicamente. No primeiro caso, a versão passa para o *status deactivated*, tem seu atributo *alive* atualizado para *false* e seu tempo de vida finalizado. Se o objeto possuir propriedades temporais, os tempos finais de validade em aberto devem receber o mesmo valor de tempo final definido para o objeto (é feita uma correção nos tempos de validade por meio da geração de nova instância do histórico), com exceção do tempo do valor do atributo *status* (com valor "D") e do *alive* (com valor "*false*"). O segundo caso é utilizado quando se deseja remover fisicamente uma informação. Porém, o TVM não define as regras para esse tipo de exclusão, assumindo que todos os dados temporais são excluídos logicamente.

Um maior detalhamento dessas regras pode ser encontrado em [MOR 2001a].

2.1.2 Hierarquia de Tipos Temporais

O TVM define um conjunto de tipos que modelam o histórico e os rótulos temporais para atributos e relacionamentos (fig. 2.1). Todos os diagramas deste trabalho são apresentados em UML.

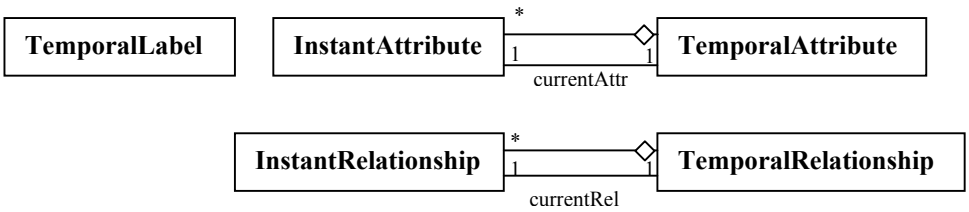

FIGURA 2.1 - Hierarquia de Tipos Temporais

A tabela 2.2 ilustra a representação gráfica utilizada na descrição das classes. Foram adicionados alguns elementos à UML.

TABELA 2.2 - Representação gráfica para os elementos do modelo

Notação	Significado
T_V	Classe temporal e versionável
<<final>>	Classe final, não pode ser especializada
itálico	Classe abstrata
<<temporal>>	Relacionamento temporal
<<T>>	Atributo temporal
t	
<<extension>>	Relacionamento de herança por extensão
negrito	Operação final(*final*) não pode ser redefinida
$	Atributo ou operação de escopo de classe (*static*)
+	Atributo ou operação pública (*public*)
-	Atributo ou operação (*private*)
#	Atributo ou operação (*protected*)

A classe *TemporalLabel* serve para armazenar o rótulo temporal com o tempo de transação inicial e final, assim como o tempo de validade inicial e final. Possui o construtor e operações para ler os atributos e para alterar os tempos finais de transação e validade (fig. 2.1).

TemporalLabel
vTimei: instant vTimef: instant tTimei: instant tTimef: instant
TemporalLabel (iValidTime:instant, fValidTime:instant, iTransTime:instant, fTransTime:instant) getTTimei ( ):instant getTTimef ( ):instant getVTimei ( ):instant getVTimef ( ):instant setTTimef (i:instant) setVTimef (i:instant)

FIGURA 2.2 - Classe *TemporalLabel*

A classe *InstantAttribute* (fig. 2.3) armazena o valor do atributo com seu rótulo temporal e a *InstantRelationship* (fig. 2.4) armazena os identificadores dos objetos que pertencem ao relacionamento e seu rótulo temporal. Ambas possuem um construtor, que recebe apenas o valor, e outro, que recebe o valor e os tempos de validade inicial e final. Também possuem as operações *getTempLabel* e *getTempLabelOf* que servem, respectivamente, para obter o rótulo temporal do valor corrente e os rótulos temporais de um valor passado por parâmetro.

InstantAttribute
value: Object tempLabel: TemporalLabel
InstantAttribute (val: Object) InstantAttribute (val : Object, iValidTime: instant, fValidTime: instant) getValue ( ): Object getTempLabel ( ): TemporalLabel getTempLabelOf (val : Object): set (TemporalLabel) setFTransactionTime (i: instant) setFValidTime (i: instant)

FIGURA 2.3 - Classe *InstantAttribute*

InstantRelationship
obj1: OIDt obj2: OIDt tempLabel: TemporalLabel
InstantRelationship (o1: OIDt, o2: OIDt) InstantRelationship (o1: OIDt, o2: OIDt, iValidTime: instant, fValidTime: instant) getObj1 ( ): OIDt getObj2 ( ): OIDt getTempLabel ( ): TemporalLabel getTempLabelOf (o1: OIDt, o2: OIDt): set (TemporalLabel) setFTransactionTime (i: instant) setFValidTime (i: instant)

FIGURA 2.4 - Classe *InstantRelationship*

As classes *TemporalAttribute* e *TemporalRelationship* (fig. 2.5 e 2.6) representam uma agregação do conjunto de valores (*InstantAttribute* e *InstantRelationship*) que os atributos e relacionamentos possuem durante a vida do objeto. A *TemporalAttribute* possui o atributo *type*, que guarda o tipo do atributo. Ambas possuem um relacionamento 1:1 com *InstantAttribute* e *InstantRelationship*, respectivamente, que indica o valor corrente do atributo e do relacionamento. Algumas das operações dessas classes são:

- *getHistory* - retorna todos os valores com os rótulos de tempo;
- *getHistoryOfValue* - retorna o histórico do valor passado por parâmetro;
- *getValueAt* - retorna os valores em um instante de tempo específico passado por parâmetro.

Cada classe possui dois tipos de construtores, assim como a *InstantAttribute* e a *InstantRelationship*. Apenas as operações *updateValue* e *close* podem alterar os valores dos tempos de transação e validade finais.

TemporalAttribute
type: string
TemporalAttribute (typ: string, val: Object) TemporalAttribute (typ: string, val: Object, iValidTime: instant, fValidTime: instant) close ( ) getCurrent ( ): InstantAttribute getHistory ( ): set (InstantAttribute) getHistoryOfValue (val: Object): set (TemporalLabel) getValueAt (at: instant): InstantAttribute getType ( ): string -setFTransactionTime (validAt: instant, fTransTime: instant) -setFValidTime (validAt: instant, fValidTime: instant) updateValue (at: instant, val: Object, iValidTime: instant, fValidTime: instant)

FIGURA 2.5 - Classe *TemporalAttibute*

TemporalRelationship
TemporalRelationship (o1: OIDt, o2: OIDt) TemporalRelationship (o1: OIDt, o2: OIDt, iValidTime: instant, fValidTime: instant) close ( ) getCurrent ( ): InstantRelationship getHistory ( ): set (InstantRelationship) getHistoryOfValue (o1: OIDt, o2: OIDt): set (TemporalLabel) getValueAt (at: instant): InstantRelationship -setFTransactionTime (validAt: instant, fTransTime: instant) -setFValidTime (validAt: instant, fValidTime: instant) updateValue (at: instant, o1: OIDt, o2: OIDt, iValidTime: instant, fValidTime: instant)

FIGURA 2.6 - Classe *TemporalRelationship*

2.2 Hierarquia de Classes

A hierarquia de classes básicas proposta pelo TVM é ilustrada na figura 2.7.

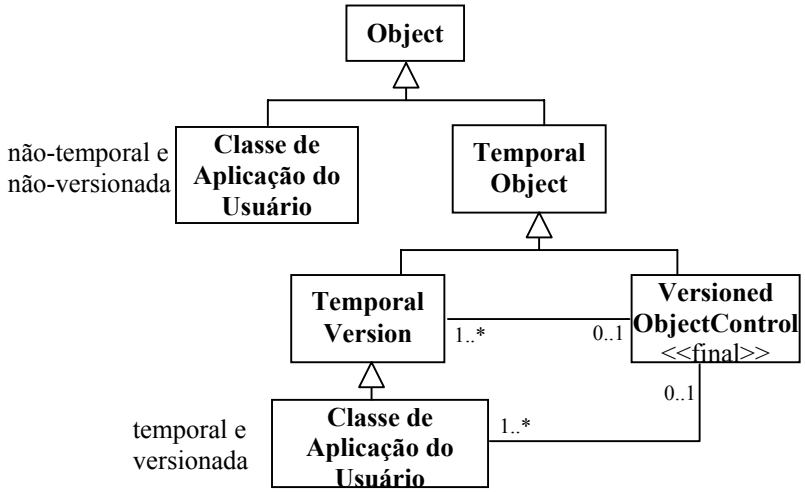

FIGURA 2.7 - Hierarquia de classes do TVM

Essa hierarquia mostra que é possível definir classes da aplicação normais (sem suporte a tempo e versões) e classes temporais versionadas. Em classes normais apenas os valores atuais são armazenados. Em classes temporais versionadas as instâncias podem ser objetos sem versões (mas com capacidade para terem versões), objetos versionados ou as próprias versões, sendo que todas possuem um rótulo temporal associado (atributo *alive* herdado de *TemporalObject*). As propriedades dessas classes podem ser definidas como estáticas ou temporalizadas.

Verificando essa hierarquia, a possibilidade de criar classes temporais diretamente de *TemporalObject* poderia ser cogitada. Essa hipótese foi descartada porque o objetivo principal é modelar e proporcionar ambas características de tempo e versões nos objetos. O Modelo permite somente os dois tipos de classe descritos. Além disso, a derivação de versões é realizada explicitamente pelo usuário e é o usuário quem define quais atributos e relacionamentos terão seus históricos armazenados. Em outras palavras, o Modelo permite que o usuário utilize somente tempo e/ou somente versões, como também ambos conceitos simultaneamente.

Nessa hierarquia, não é permitido que uma instância passe de não versionada para versionada dinamicamente; a capacidade de versionamento deve ser designada em tempo de projeto. Porém, mesmo que o objeto possua essa capacidade ele não é obrigado a apresentar versões.

As classes *TemporalObject* e *TemporalVersion* são abstratas, não podendo ser diretamente instanciadas. As classes *TemporalObject* e *VersionedObjectControl* são controladas pelo sistema gerenciador e ficam transparentes ao usuário. A *VersionedObjectControl* pode apenas ser instanciada pelo sistema.

2.3 Estados de um Versão

Os estados de uma versão identificam quais operações são permitidas a uma versão ao longo de sua vida. Os possíveis estados de uma versão e as operações aplicáveis a eles são:

- *working* - pode ser promovida para s*table*, servir como base para uma derivação (sendo promovida automaticamente para s*table*), ser alterada, consultada ou excluída;
- *stable* - pode servir como base para uma derivação, ser promovida para *consolidated*, ser consultada, compartilhada por outros usuários ou excluída (se não possuir versão sucessora);
- *consolidated* - pode servir como base para uma derivação, ser consultada e compartilhada por outros usuários;
- *deactivated* - pode ser somente consultada e restaurada (voltando ao estado w*orking* ou *stable*).

2.4 Relacionamento de Herança por Extensão

Uma das características mais importantes do Modelo Temporal de Versões é a definição de herança por extensão (ou relacionamento de herança por extensão), na qual as versões são admitidas nos vários níveis da hierarquia de herança. Com esse recurso, um objeto pode ser desenvolvido em um nível de abstração e posteriormente detalhado nos níveis inferiores da hierarquia. Isso possibilita que a modelagem de entidades do mundo real seja feita em vários níveis, projetando ou modificando características de um objeto em uma camada de cada vez. Essa hierarquia formada pelas classes que participam da herança por extensão é chamada hierarquia de extensão.

Deve-se evitar confusões com a herança adotada nas linguagens e modelos orientados a objetos, denominada de herança por refinamento. A primeira diferença entre essas heranças está na finalidade da modelagem. Na especificação de sistemas ou na modelagem de dados, essas heranças são definidas em tempo de análise ou projeto. A herança por refinamento traz as vantagens de extensibilidade, pela redefinição de estado e comportamento nas subclasses durante o projeto, e de reuso de código na implementação. A herança por extensão é definida quando é detectada a necessidade de instanciar a entidade em diferentes níveis de detalhamento (em tempo de execução), como explicado anteriormente. Além disso, a herança por extensão permite a modelagem dinâmica das aplicações. Durante a fase de projeto o esquema raramente

apresenta-se estável ou completamente definido, sendo possível realizar modificações em cada nível separadamente sem afetar as instâncias existentes.

A segunda diferença está no aspecto de funcionamento das hierarquias. Quando um tipo Y refina um tipo X, uma instância de Y mantém seu próprio armazenamento de todos os valores definidos por Y e X. Quando um tipo Y estende X, a seguinte afirmativa é verdadeira: para toda instância y de Y há uma instância x em X cujos valores dos atributos são herdados (compartilhados) por y. Nesse caso, acrescenta-se um ponteiro em y que referencia a respectiva instância x, conforme apresentado na figura 2.8. Esse conceito é apresentado por Biliris em [BIL 90].

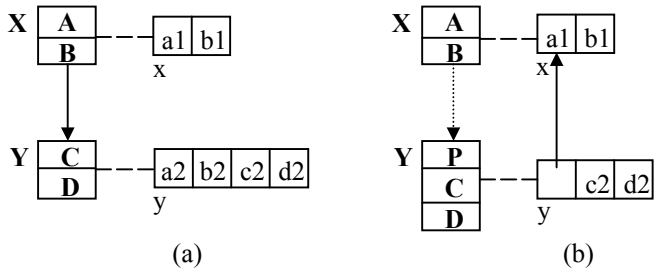

FIGURA 2.8 - (a) Refinamento: instâncias de Y mantêm seu próprio armazenamento para os atributos definidos em Y e X (b) Extensão: todos os atributos de x são compartilhados por y

Dessa forma, na hierarquia de extensão, cada versão deve estar necessariamente associada a pelo menos um ascendente (objeto versionado, versão ou objeto temporal versionável que ainda não possui versões) na superclasse, caso não pertença a uma classe raiz na hierarquia. Esse ascendente (ou grupo de ascendentes) tem que pertencer a mesma entidade da versão em questão, como será visto adiante. Assim, no momento da criação de uma versão, deve ser feita a ligação dessa com um ascendente. Entretanto, versões na superclasse podem existir sem estarem relacionadas ainda com versões nas subclasses. Essa obrigatoriedade de um ascendente também é imposta aos objetos versionados e não versionados.

A referência a um ascendente pode ser feita de forma estática, quando é indicada uma versão específica, ou dinâmica, quando a referência é ao objeto versionado e não a uma determinada versão dele. Na referência dinâmica, a versão corrente será utilizada quando o ascendente for solicitado.

É possível que uma versão apresente mais de um ascendente. Na Figura 2.9, é apresentada a correspondência entre versões em diferentes níveis. Considerando que existe a classe Veículo e a classe Automóvel, a qual é uma especialização da anterior, a entidade Palio aparece no nível de Veículo, onde apresenta os atributos motor e combustível, e no nível de Automóvel, com atributos número de volumes e acessórios. Tomando como exemplo a versão ELX no nível Automóvel, têm-se como ascendentes em Veículo, as versões v3 e v4, o que significa que é possível ter um Palio ELX 1.6 a álcool ou à gasolina.

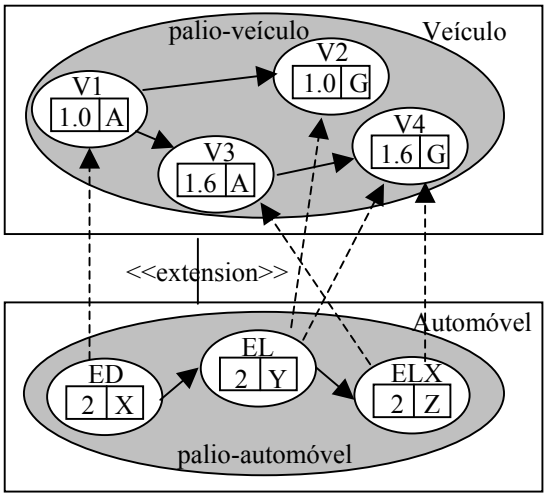

FIGURA 2.9 - Correspondências de versões entre diferentes níveis de hierarquia

Podem ser estabelecidas restrições de cardinalidade (mapeamento) entre versões de um objeto em uma classe e as versões de seus ascendentes na superclasse, podendo ser 1:1, 1:n, n:1 ou n:m. É importante lembrar que, independente da cardinalidade, os ascendentes de cada versão devem pertencer a mesma entidade da versão.

2.5 Configuração

A definição de uma configuração para uma determinada versão é dada pela escolha de uma única versão (pois pode haver vários ascendentes) para cada nível ascendente em que a entidade está representada na hierarquia de extensão e uma única versão para cada componente na hierarquia de agregação. Se o componente também possui componentes, o mesmo processo é feito de forma recursiva para todos estes na hierarquia de agregação.

É possível criar diferentes configurações para o mesmo objeto mediante a escolha de versões para componentes e/ou ascendentes. Dessa forma, uma configuração pode ser dita uma versão especial de um objeto, chamada versão configurada.

2.6 Identificador de Objeto

Todos os objetos possuem um *tvOID*, que é um identificador com a estrutura "identificador da entidade, identificador da classe, número da versão".

A entidade representa um objeto do mundo real que pode estar modelado de forma distribuída em diversas classes (diferentes níveis da hierarquia por extensão). Com o *tvOID* o objeto carrega consigo a entidade à qual pertence, o nível em que está, isto é, a classe à qual pertence, e onde está situado dentro da hierarquia de derivação de versões. No exemplo anterior, Palio representa uma entidade modelada em duas classes (Veículo e Automóvel).

Os números das versões são números inteiros gerados seqüencialmente. O número da versão é sempre nulo para objetos de classes normais, 1 para a primeira instância (objeto) de classes temporais versionadas (objeto que ainda não possui versões ou primeira versão do objeto) e 0 para objetos versionados. Para as demais versões, a

atribuição deste número deve ser de acordo com o valor do atributo *nextVersionNumber* definido para o controle do respectivo objeto versionado.

2.7 Relações entre Classes Normais e Temporais Versionáveis

Os tipos de relações permitidas entre classes normais (sem tempo e versões) e classes temporais versionáveis são descritos a seguir:

- associação - se for temporal, é possível apenas entre duas classes temporais versionadas; caso contrário, é possível entre quaisquer tipos de classes;
- herança por refinamento - é permitido apenas entre duas classes normais;
- herança por extensão - é possível entre duas classes normais, duas temporais versionadas e entre uma normal como superclasse e uma temporal versionada como subclasse (apenas com correspondência 1:1 ou n:1);
- agregação - é permitido entre duas classes normais, duas temporais versionadas e entre temporal versionada e normal, apenas se a classe normal for componente.

2.8 Descrição das Classes da Hierarquia

Esta seção apresenta a descrição das classes da hierarquia base do TVM juntamente com seus atributos e operações.

Algumas operações das classes detalhadas envolvem acesso a metadados, os quais devem armazenar informações tais como: se a classe é temporal versionada; se o relacionamento é herança por extensão, armazenando o nome de sua classe ascendente se não é classe raiz; os atributos e relacionamentos que são temporais; entre outras. Este trabalho pressupõe que as classes de metadados já estão definidas na base de dados.

2.8.1 Classe *Object*

Esta classe contém os atributos e operações comuns a todos os objetos (fig. 2.10). Possui um único atributo chamado *tvOID* que é modelado por uma classe chamada *OIDt* para possuir a estrutura "identificador da entidade, identificador da classe, número da versão".

Object
tvOID: OIDt
Object ( ) Object (ascendID: OIDt) Object (entityName: string) Object (entityId: integer, classId: integer, versionId: integer) delete (allReferences: boolean) deleteObjectTree (allReferences: boolean) -findVersion (entityId: integer, classId: integer): integer getAscendant ( ): set(OIDt) getAscendant (className: string): OIDt getClassId ( ): integer getClassName ( ): string getCompleteObject ( ): set(Object) getCorrespondence (className: string): string

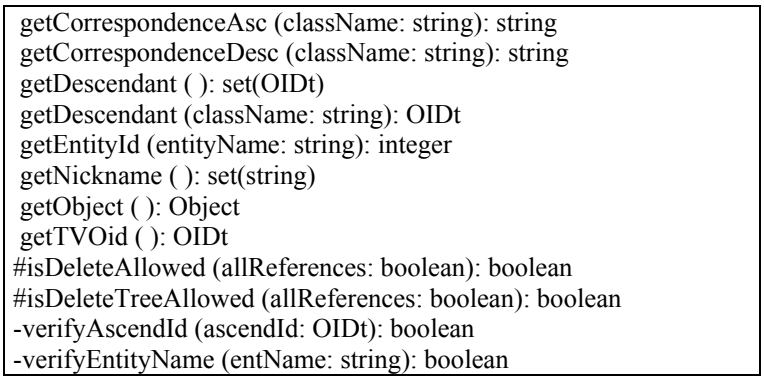

FIGURA 2.10 - Classe *Object*

O construtor que recebe um *tvOID* por parâmetro chama a operação *verifyAscendId* para verificar se o *tvOID* é válido para a classe ascendente. Quando o construtor recebe um identificador de entidade, deve conferir nos metadados se é um valor válido. E quando recebe entidade, classe e versão, deve verificar se os valores são válidos e se o objeto já não existe.

Entre as operações dessa classe estão:

- *findVersion* - retorna o número da versão a ser criada;
- *getClassName* - consulta os metadados para buscar o nome da classe à qual o objeto pertence;
- *getClassId* - consulta os metadados e retorna o identificador da classe;
- *getCorrespondence* - consulta os metadados e retorna a correspondência da classe do objeto com a classe recebida por parâmetro;
- *getCorrespondenceAsc* - consulta os metadados e retorna apenas a cardinalidade da ascendente na correspondência entre a classe do objeto e sua ascendente recebida por parâmetro;
- *getCorrespondenceDesc* - consulta os metadados e retorna apenas a cardinalidade da descendente na correspondência entre a classe do objeto e sua descendente recebida por parâmetro;
- *getEntityId* - consulta os metadados e retorna o identificador da entidade à qual foi passado o nome como parâmetro;
- *getObject* - retorna os valores de atributos de um objeto;
- *isDeleteAllowed* - retorna se é possível excluir o objeto;
- *isDeleteTreeAllowed* - retorna se é possível excluir o objeto e todos seus descendentes;
- *verifyAscendantId* - retorna se o ascendente recebido por parâmetro é válido;
- *verifyEntityName* - retorna se a entidade recebida por parâmetro é válida;
- *getAscendant* - consultando a estrutura do atributo *tvOID*, retorna o ascendente da classe imediatamente ascendente se não for informada uma classe específica; caso contrário, retorna o ascendente na classe passada por parâmetro;

- *getDescendant* - consultando a estrutura do atributo *tvOID*, retorna o descendente da classe imediatamente descendente se não for informada uma classe específica; caso contrário, retorna o descendente na classe passada por parâmetro;
- *getCompleteObject* - retorna os ascendentes de um objeto na hierarquia de herança e nas classes agregadas.

Esta classe possui o relacionamento *nickname* com a classe *Name* (fig. 2.11), o qual possibilita manter apelidos para os objetos.

Name
value: string
Name (nam: string) getClassName ( ): string getName ( ): string

FIGURA 2.11 - Classe *Name*

A classe *OIDt* (fig. 2.12) que modela o *tvOID* de um objeto, possui as operações *getClassNr*, *getEntityNr* e *getVersionNr* as quais a partir do atributo *value*, retornam, respectivamente, o número da classe, da entidade e da versão.

OIDt
value: string
OIDt (E: integer, C: integer, V: integer) getClassNr ( ): integer getEntityNr ( ): integer getOID ( ): OIDt getVersionNr ( ): integer

FIGURA 2.12 - Classe *OIDt*

2.8.2 Classe *TemporalObject*

A classe *TemporalObject* representa os aspectos temporais do modelo (fig. 2.13). Possui o atributo *alive*, que define se um objeto está ativo ou não (se foi excluído logicamente), sendo modelado como temporal porque os objetos temporais versionados podem ser restaurados.

TemporalObject
t alive: boolean default true
TemporalObject ( ) TemporalObject (ascendID: OIDt) TemporalObject (entityName: string) TemporalObject (entityId: integer, classId: integer, versionId: integer) closeTemporalLabels ( ) delete ( ) getAlive ( ): boolean **getAttributeHistory** (atribName: string): set (InstantAttribute) **getAttributeValueAt**(atribName: string, i: instant): InstantAttribute getLifetimeI ( ): instant getLifetimeF ( ): instant **getObjectHistory** ( ): set (TemporalAttribute) **getRelationshipHistory** (relatedObjId: OIDt, relatName: string): set (InstantRelationship) **getRelationshipValueAt** (relatedObjId: OIDt, relatName: string, i: instant): InstantRelationship **setTemporalAttribute** (attribName: string, newValue: Object) **setTemporalRelationship** (relatName: string, newO1: OIDt, newO2: OIDt)

FIGURA 2.13 - Classe *TemporalObject*

Esta classe contém as seguintes operações:

- *closeTemporalLabels* - fecha os tempos finais de transação e/ou validade em aberto das propriedades temporais do objeto;
- *delete* - exclui o objeto logicamente;
- *getAlive* - retorna se o objeto está ativo ou não;
- *getAttributeHistory* - retorna o conjunto de valores do histórico do atributo passado por parâmetro e seus respectivos rótulos temporais, se esta propriedade for temporal; caso contrário, retorna o valor atual com o rótulo temporal contendo *null*;
- *getAttributeValueAt* - retorna o valor do atributo no instante passado por parâmetro e seu rótulo de tempo, se a propriedade for temporal; caso contrário, retorna o valor atual com o rótulo temporal contendo *null*;
- *getLifetimeI* - retorna o tempo de vida inicial (tempo de validade inicial) do objeto ativo no momento;
- *getLifetimeF* - retorna o tempo de vida final do objeto (tempo de validade final) do objeto. Este tempo é *null* se o objeto estiver ativo, senão é o valor mais recente que aparece em tempo de validade final, pois pode ter mais de um se o objeto tiver sido excluído e restaurado;
- *getObjectHistory* - retorna os valores do atributo *alive* com seu conjunto de valores iniciais e finais;
- *getRelationshipHistory* - retorna o conjunto de valores do histórico do relacionamento passado por parâmetro e seus respectivos rótulos temporais, se esta propriedade for temporal; caso contrário, retorna o valor atual com o rótulo temporal contendo *null*;
- *getRelationshipValueAt* - retorna os valores dos *tvOIDs* do relacionamento no instante passado por parâmetro e seu rótulo de tempo, se a propriedade for temporal; caso contrário, retorna o valor atual com o rótulo temporal contendo *null*;
- *setTemporalAttribute* - atualiza o valor do atributo temporal e armazena os valores correspondentes no rótulo temporal;
- *setTemporalRelationship* - atualiza o valor do relacionamento temporal e armazena os valores correspondentes no rótulo temporal.

2.8.3 Classe *VersionedObjectControl*

Esta classe define atributos para fazer o controle dos objetos versionados em relação as suas versões (fig. 2.14).

VersionedObjectControl
t configurationCount: integer default 0 **t** currentVersion: OIDt **t** firstVersion: OIDt **t** lastVersion: OIDt nextVersionNumber: integer default 3 **t** userCurrentFlag: boolean default false

t versionCount: integer default 2
VersionedObjectControl (entityId: integer, classId: integer, configCount: integer, currentV: OIDt, firstV: OIDt, lastV: OIDt, nextVNumber: integer, userCurrentFlag: boolean, vCount: integer) VersionedObjectControl (entityId: integer, classId: integer, currentV: OIDt, firstV: OIDt, lastV: OIDt) delete (entityId: integer, classId: integer) getConfigurationCount ( ): integer getCurrentVersion ( ): OIDt getFirstVersion ( ): OIDt getLastVersion ( ): OIDt getNextVersionNumber ( ): integer getUserCurrentFlag ( ): boolean getVersionCount ( ): integer **restore** ( ) setCurrentVersion ( ) setCurrentVersion (newVersionId: OIDt) setFirstVersion (first: OIDt) setLastVersion (last: OIDt) setUserCurrentFlag (flag: boolean) updateConfigurationCount ( ) updateVersionCount ( ) updateNextVersionNumber ( )

FIGURA 2.14 - Classe *VersionedObjectControl*

Alguns atributos são inicializados com valores *default*, pois um controle para o objeto versionado é criado apenas na primeira derivação de versão. Portanto, quando esta primeira derivação ocorre, o objeto fica com a primeira versão, que era a única até o momento, e a segunda, que corresponde à nova derivada. Assim, o atributo *versionCount* inicia com 2 porque, obrigatoriamente, têm-se duas versões quando se cria uma instância desta classe; e o *nextVersionNumber* recebe 3, pois representa o número da próxima versão dentro do objeto versionado. O *configurationCount* recebe 0 porque inicialmente não existem configurações no objeto versionado.

A operação *setCurrentVersion* serve para o usuário trocar a versão corrente que, em princípio, é mantida pelo sistema como a mais recente (última a ser criada), podendo ser uma versão configurada. Quando nesta operação, o usuário passar por parâmetro o *tvOID* de uma versão do objeto versionado, esta passa a ser a corrente, e o atributo *userCurrentFlag* é alterado para *true*, indicando que a versão corrente foi definida pelo usuário. Se a operação for invocada sem passagem de parâmetro, o sistema recupera o controle da versão corrente que passa a ser a mais recente, e o *userCurrentFlag* é alterado para *false.*

As demais operações são simples e fazem apenas a recuperação e atualização dos atributos.

2.8.4 Classe *TemporalVersion*

A classe *TemporalVersion* fornece a capacidade de versionamento aos objetos (fig. 2.15).

TemporalVersion
t ascendant: set (OIDt) default NULL configuration: boolean default false **t** descendant: set (OIDt) default NULL predecessor: set (OIDt) default NULL

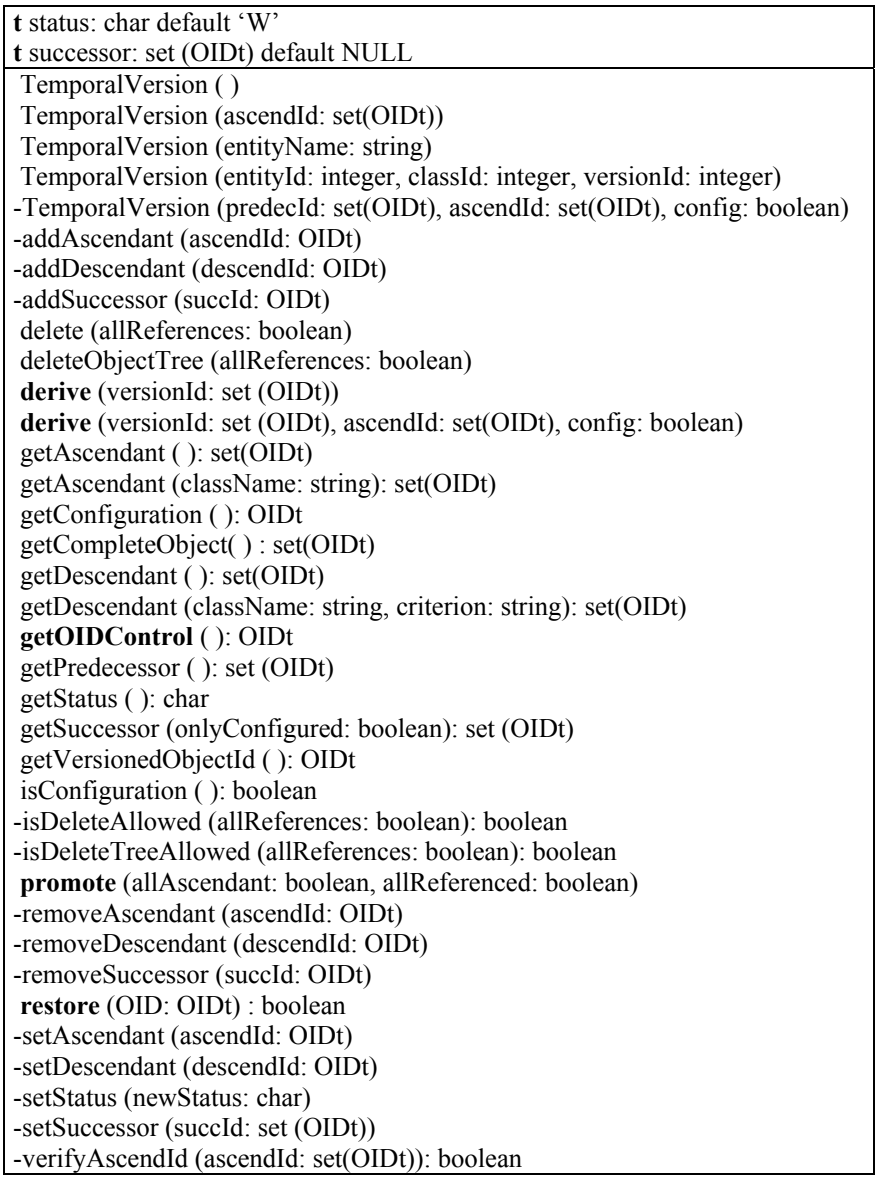

FIGURA 2.15 - Classe *TemporalVersion*

Entre as suas operações estão:

- *derive* - gera uma nova versão para um objeto, a partir de uma (um único *tvOID* é fornecido), ou mais (um conjunto de *tvOID*s é fornecido) versões ou objetos. Se for fornecido apenas um *tvOID*, este pode ser de uma versão, de um objeto versionado ou de um objeto sem versões; caso contrário, os *tvOID*s devem ser apenas de versões. A nova versão gerada possui estado *working*, e a versão que serviu de base a ela é promovida para *stable* (se seu estado era *working*).
- *getConfiguration* - cria uma configuração.
- *promote* - atualiza o *status* da versão ou do objeto sem versões. O *status* de uma versão deve ser o mesmo ou menos desenvolvido que os de seus ascendentes. Assim, se a opção *allAscendants* for verdadeira na promoção de uma versão, então seus ascendentes serão automaticamente promovidos para o mesmo *status* da versão, caso já não possuam o mesmo *status*. Se os ascendentes forem menos desenvolvidos e esta opção não for usada, retornará

um erro. Da mesma forma, funciona o *allReferences*, permitindo que todos os objetos referenciados sejam também promovidos.

- *setStatus* - atribui um determinado *status* ao objeto, sendo chamada pelo construtor e pelos métodos *derive*, *promote*, *delete* e *restore*.
- *getOIDControl* - retorna o *tvOID* da instância correspondente ao objeto versionado em *VersionedObjectControl*.
- *delete* - verifica as regras de exclusão de versões e objetos, chamando a operação *isDeleteAllowed* para isto, e armazena os dados para a exclusão lógica. Se *allReferences* for verdadeiro, então as referências para o objeto são excluídas também.
- *isDeleteAllowed* - retorna verdadeiro se uma versão ou objeto versionado pode ser excluído de acordo com as condições de exclusão.
- *isDeleteTreeAllowed* - retorna verdadeiro se todos os objetos, versões e objetos versionados descendentes podem ser excluídos de acordo com as condições de exclusão.
- *deleteObjectTree* - verifica através da operação *isDeleteTreeAllowed* se todos os descendentes podem ser excluídos, e então exclui o objeto e toda sua árvore descendente.
- *restore* - restaura objetos que foram excluídos logicamente de acordo com várias regras que foram definidas. Podem ser restaurados uma versão folha que possui predecessora, uma versão sem predecessora (é o próprio objeto sem versões), uma versão cuja predecessora foi excluída (a predecessora deve ser restaurada também), todas as versões do objeto (se não houver outra seqüência de derivação) e um objeto versionado com versões (o objeto restaurado volta da forma que estava no momento da exclusão).
- *getCompleteObject* - retorna os ascendentes de um objeto na hierarquia de herança, um para cada superclasse, e um objeto por classe agregada.
- *getVersionedObjectId* - retorna o *tvOID* do objeto versionado ao qual o objeto pertence.
- *verifyAscendantId* - retorna se o conjunto de ascendentes recebido por parâmetro é válido.

2.9 Linguagens

O TVM define uma linguagem de definição de classes, na qual é possível especificar tanto classes normais quanto temporais versionadas.

Além disso, foi definida uma linguagem de consulta para o TVM em [MOR 2001b], a qual permite que sejam feitas consultas para recuperar: o histórico da vida de um objeto (os tempos em que ele está ativo), os valores atuais e o histórico de atributos e relacionamentos, os valores de propriedades em determinados instantes, os estados das versões, as versões de um objeto versionado, a hierarquia de derivação e de extensão em determinados instantes ou períodos, as configurações, entre outras informações.

2.10 Considerações Finais

Este capítulo apresentou como o Modelo Temporal de Versões trabalha com os conceitos de tempo e versões. Além de algumas características importantes, como as regras de integridade temporal a serem obedecidas e o funcionamento do relacionamento de herança por extensão, foram mostradas também as hierarquias de classes necessárias para lidar com esses conceitos, bem como uma descrição mais detalhada dessas classes para esclarecer a função de cada uma delas.

3 Padrão ODMG

O principal objetivo do padrão ODMG (*Object Data Management Group*) é fornecer um conjunto de especificações que permita que os desenvolvedores escrevam aplicações portáveis, isto é, aplicações que rodam em mais de um produto [CAT 2000]. Este capítulo descreve os principais componentes desse padrão.

3.1 Componentes da Arquitetura ODMG

Os principais componentes que formam a arquitetura ODMG são:

- Modelo de Objetos - é constituído de extensões que foram feitas baseadas no modelo de objetos OMG.
- Linguagem de especificação de objetos – ODL (*Object Definition Language*), que é equivalente a DDL (*Data Definition Language*) dos SGBDs convencionais. Há também a OIF (*Object Interchange Language*), que serve para carregar e descarregar o estado corrente de um ODMS para ou de um arquivo ou conjunto de arquivos.
- Linguagem de consulta – OQL (*Object Query Language*), que é uma linguagem declarativa para fazer atualização e consulta ao conteúdo da base de dados, sendo baseada no SQL.
- *Bindings* com LPOO (Linguagens de Programação Orientada a Objetos) - C++ *language binding*, Smalltalk *language binding* e Java *language binding*. Apresentam as definições do padrão em uma LPOO.

Os programas são escritos em uma LPOO (C++, Java, Smalltalk), e o esquema é escrito em ODL abstrata do ODMG ou em ODL específica de uma LPOO, a qual é definida pelo *binding* de cada linguagem.

3.2 Modelo de Objetos

O Modelo de Objetos do ODMG é o modelo de referência para um banco de dados orientado a objetos, assim como o modelo relacional para um BD relacional. Esse modelo especifica construções que são suportadas por um ODBMS, tais como: características de objetos, como estes se relacionam, e como os objetos podem ser nomeados e identificados.

Algumas características do modelo em questão são:

- As primitivas do Modelo são o objeto, o qual possui um identificador único (pode ser referenciado), e o literal, que não possui identificador (não pode ser referenciado).
- O estado de um objeto é definido pelos valores de suas propriedades (atributos ou relacionamentos).
- O comportamento de um objeto é determinado pelo seu conjunto de operações (apenas as assinaturas das operações).

- Tanto objetos quanto literais podem ser de um tipo. Todos os elementos de um determinado tipo possuem um mesmo conjunto de propriedades e comportamento.
- Um modelo específico é construído usando a ODL (independente de linguagem ou não).

3.2.1 Tipos

O conceito de tipo no ODMG possui dois aspectos importantes em sua definição, os quais se referem aos níveis de abstração:

- uma especificação externa (nível de especificação) - é uma definição abstrata de operações que podem ser chamadas, de propriedades (atributos e relacionamentos) que cada objeto possui e de exceções que podem ser sinalizadas;
- uma ou mais implementações (nível de implementação) - é a implementação das operações e outros detalhes internos em uma ou mais linguagens específicas de programação.

Há três formas para especificação externa:

- definição de interface - define apenas o comportamento de um tipo objeto e não pode ser instanciada;
- definição de classe - define estado e comportamento de um tipo objeto e pode ser instanciada;
- definição de literal - define apenas estado de um tipo literal.

A implementação define uma representação da especificação em uma determinada linguagem de programação da seguinte forma:

- associa a cada propriedade abstrata uma variável de instância, de acordo com a LPOO escolhida;
- define para cada operação abstrata um método (corpo de um procedimento que implementa a operação).

Pode haver métodos que não possuem correspondência nas operações abstratas, pois a parte interna de uma implementação não é visível ao usuário. Assim, pode-se dizer que a interface é a parte pública do tipo, enquanto a implementação pode introduzir, mais tarde, propriedades e operações privadas, se necessário [COO 97].

A maneira de mapear conceitos abstratos para uma LPOO varia de linguagem para linguagem. Por exemplo, um literal *struct* é mapeado no C++ diretamente para o equivalente, mas em Java ou Smalltalk, ele é mapeado para uma classe sem métodos. No caso de um literal *float* em C++ e Java tem-se o equivalente, mas em Smalltalk é transformado para a classe *Float*.

3.2.1.1 Herança

No modelo de objetos ODMG, a herança pode ser definida de duas formas:

- *ISA* ou *is-a* - representa herança de comportamento. Neste caso, apenas interfaces podem ser herdadas, mas tanto classes quanto interfaces podem

herdá-las. Pode-se ter herança múltipla, ou seja, é possível herdar de várias interfaces. A forma de representá-la é através de dois pontos (:). Por exemplo:

```
interface Funcionario {...};
interface Professor : Funcionario {...};
class Professor_Titular : Professor {...};
```

- *Extends* - representa herança de estado e comportamento. Neste caso, a herança é dada apenas entre classes, não podendo haver herança múltipla, ou seja, é possível herdar de apenas uma classe. A forma de representá-la é através da palavra *extends*. Por exemplo:

```
class Funcionario{...};
class Professor extends Funcionario{...};
```

A mesma classe pode também ter os dois tipos de herança, por exemplo:

```
interface Pessoa {...};
class Funcionario{...};
class Professor extends Funcionario : Pessoa{...};
```

Logo, uma interface, ou uma classe, pode apresentar herança do tipo *is-a* de várias outras interfaces, e uma classe pode ter herança do tipo *extends* de, no máximo, uma outra classe.

3.2.1.2 *Extent*

A extensão de um tipo (*extent*) é formada por todas as instâncias deste tipo que existem no banco de dados. Se o projetista desejar, esta extensão é mantida automaticamente pelo ODBMS. A manutenção do *extent* envolve a inclusão e remoção de instâncias, bem como a criação e manutenção de índices para estas. Por exemplo, na classe Pessoa, pode-se declarar o seu *extent* conforme descrito abaixo:

```
class Pessoa
    (extent pessoas)
{...};
```

3.2.1.3 Chaves

O conceito de chaves no modelo de objetos ODMG é apenas uma restrição de unicidade de valor e não tem o papel de OID, nem de referência entre objetos. Portanto, chave em ODMG corresponde a um ou a mais atributos cujos valores identificam uma instância dentro de uma extensão. Essa chave deve ser definida pelo usuário, enquanto o OID é definido pelo sistema.

```
class Pessoa
       (extent pessoas key codigo)
 {attribute short codigo;
  ...};
```

3.2.2 Objetos

Há alguns aspectos que devem ser considerados em relação aos objetos:

- O OID de um objeto o diferencia dos demais objetos dentro do BD, sendo gerado pelo ODBMS. Ele se mantém o mesmo durante toda a vida do objeto.

A estrutura do identificador não é definida pelo modelo, sendo esta definida na implementação.

- Dois objetos são iguais quando possuem os mesmos valores para os atributos.
- Dois objetos são o mesmo quando se referenciam ao mesmo OID.
- O método *same_as()* testa se dois objetos possuem o mesmo OID.
- O método *copy()* faz uma cópia de determinado objeto, o que resulta em dois objetos iguais.
- O método *delete()* remove o objeto da memória e do banco de dados.
- Os métodos *same_as()*, *copy()*, *delete()*, entre outros, estão definidos na interface *Object*, que é herdada por qualquer objeto definido pelo usuário.
- Acesso, criação, alteração ou remoção de objetos persistentes devem ser feitos dentro do escopo de uma transação.
- É permitido associar um ou mais nomes a um objeto. Este nome não é armazenado dentro do objeto, ele é controlado pelo ODBMS que fornece uma função que mapea um nome de objeto para o objeto em si. É importante enfatizar que nomes de objetos não devem ser confundidos com chaves ou OIDs.
- Um objeto pode ser persistente (sobrevive à execução de um programa), ou transiente (ao terminar a execução do programa deixa de existir). Os dois podem ser manipulados pelas mesmas operações. Esses dois conceitos são independentes de tipo, pois um mesmo tipo pode possuir algumas instâncias persistentes e outras transientes.
- Um objeto pode ser do tipo atômico (definido pelo usuário), coleção ou estruturado.
- Os elementos de um objeto coleção devem ser todos de um mesmo tipo, podendo ser de um dos tipos de literais ou de um dos tipos de objetos. Os tipos de coleções são:
    - *Set<t>* (conjunto) - sem ordenação e sem duplicatas.
    - *Bag<t>* (conjunto) - sem ordenação e aceita duplicatas.
    - *List<t>* (lista) - com ordenação e aceita duplicatas.
    - *Array<t>*(arranjo) - com ordenação, aceita duplicatas e localiza elementos por posição.
    - *Dictionary<t,v>* (lista indexada) - com ordenação, aceita duplicatas e localiza elementos por chave associada a cada elemento.
- Vários métodos podem ser aplicados a coleções, tais como - *new()*, *is_empty()*, *contains_element()*, etc.
- Os objetos estruturados são: *Date*, *Time*, *Timestamp* e *Interval.*

3.2.3 Literais

Como visto anteriormente, os literais não possuem identificadores. Três tipos de literais são suportados pelo Modelo de Objetos:

- Atômico - é representado por números e caracteres, tais como: *long*, *long long*, *short*, *unsigned long*, *unsigned short*, *float*, *double*, *boolean*, *octet*, *char*, *string*, *enum* (enumeração).
- Coleção - é semelhante a um objeto coleção, porém não possui identificador de objeto. Os literais coleção existentes são: *set*<t>, *bag*<t>, *list*<t>, *array*<t>, *dictionary*<t>. Seus elementos podem ser de tipos literais ou tipos objetos.
- Estruturado - é representado por: *date*, *interval*, *time*, *timestamp* e definidos pelo usuário (*structure*<>).

3.2.4 Modelando Estado e Comportamento

No ODMG, atributos e relacionamentos modelam estado, enquanto operações definem comportamento.

O valor de um atributo é sempre um literal ou um identificador de objeto. No exemplo a seguir, os atributos nome e idade são literais, e depto é um identificador de objeto de uma instância de Departamento.

```
class Pessoa
{attribute string nome;
 attribute short idade;
 attribute Departamento depto;};
```

Os relacionamentos são propriedades definidas entre, no máximo, dois tipos, em que cada tipo participante do relacionamento tem que possuir um OID. O ODMG permite apenas relacionamentos binários, podendo ter cardinalidade um-para-um, um-para-muitos ou muitos-para-muitos.

Um relacionamento é definido explicitamente pela declaração de caminhos que permitem à aplicação usar as conexões lógicas entre os objetos envolvidos no relacionamento. Esses caminhos são declarados em pares, um para cada direção. A seguir, é mostrado um exemplo em que um professor ensina um conjunto de disciplinas, e uma disciplina é ensinada por um professor. O relacionamento declarado nas duas direções é apresentado nas duas classes Professor e Disciplina:

```
class Professor
{...
 relationship set<Disciplina> ensina
    inverse Disciplina :: é_ensinada_por;
 ...};

class Disciplina
{...
  relationship Professor é_ensinada_por
    inverse Professor :: ensina;
 ...};
```

O ODBMS é responsável por manter a integridade referencial dos relacionamentos. Se um objeto que participa de um relacionamento for removido, então qualquer caminho para aquele objeto deve ser removido também.

As operações são representadas apenas pelas suas assinaturas, que contêm o nome da operação, nome e tipo de cada argumento, tipo de valor retornado e exceções que podem ser sinalizadas pela operação.

O nome de uma operação deve ser único apenas dentro de uma definição de tipo. Assim, tipos diferentes podem ter operações definidas com o mesmo nome. Nesses casos, quando uma operação é chamada, uma operação específica deve ser selecionada para execução. Essa seleção é feita pegando-se o tipo mais específico do objeto fornecido como primeiro argumento da chamada.

Operações podem criar exceções, e essas podem passar os resultados das exceções. Quando uma exceção é criada, informações sobre a causa desta são passadas para o manipulador de exceções como propriedades de uma exceção.

3.3 ODL

A ODL é a linguagem de especificação usada para definir a especificação de tipos objeto de acordo com o Modelo de Objetos ODMG. Alguns aspectos devem ser considerados em relação a essa linguagem de especificação:

- suporta todas as construções semânticas do Modelo de Objetos ODMG;
- não é uma linguagem de programação completa, mas uma linguagem de definição para especificação de objetos;
- é independente de LPOO, o que permite que os usuários a usem para definir semânticas de esquema de uma forma independente de linguagem;
- deve ser extensível;
- define as propriedades e operações dos tipos objeto, sendo que para as operações é descrita apenas a assinatura dessas.

A figura 3.1 ilustra um gráfico de representação de esquema, seguida da definição utilizando ODL.

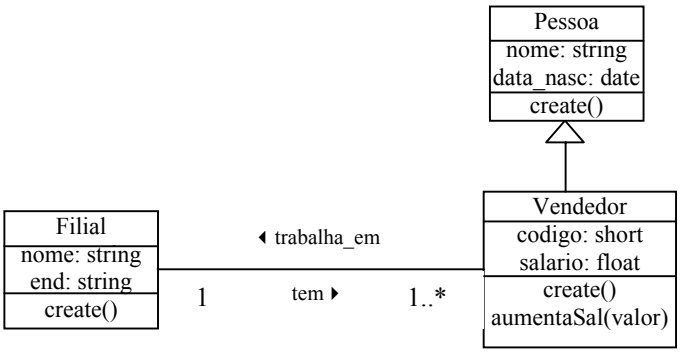

FIGURA 3.1 - Gráfico de representação de esquema

```
class Filial
  (extent filiais)
{
 attribute string nome;
 attribute string end;
 relationship set<Vendedor> tem
       inverse Vendedor::trabalha_em;
 void create();
};
class Pessoa
  (extent pessoas)
{
 attribute string nome;
 attribute date data_nasc;
 void create();
};
```

```
class Vendedor extends Pessoa
   (extent vendedores)
{
 attribute short codigo;
 attribute float salario;
 relationship Filial trabalha_em
          inverse Filial::tem;
 void create();
 void aumentaSal(in float valor);
};
```

Este padrão não especifica uma OML – *Object Manipulation Language.*

3.4 OQL

A OQL é uma linguagem declarativa para consultar os objetos do BD. Algumas de suas características são:

- Suporta as cláusulas *select*, *from*, *where*, *group by*, *having* e *order by*.
- Pode ser usada por usuários ou dentro de uma linguagem de programação.
- Fornece primitivas de alto nível para lidar com *sets, structures, lists* e *arrays*, tratando estas construções com a mesma eficiência.
- Possui uma sintaxe baseada no SQL, mas a composição é mais livre.
- Permite expressões aninhadas.
- Pode chamar métodos dos tipos envolvidos na consulta.
- Não fornece operadores para atualização, mas pode chamar operações definidas nos objetos para realizar esta tarefa. Assim, não viola a semântica do modelo de objetos, o qual, por definição, é gerenciado pelos métodos especificados no objeto.
- É possível definir um nome para uma determinada consulta, que é armazenada no BD. Para uso, a consulta é referenciada através do nome definido.
- Apresenta construtores de objetos, estruturas, *sets*, *lists*, *bags* e *arrays*.

3.4.1 Entradas e Resultados

É possível fazer consultas em cima de extensões de classes, como também nomes de objetos. Para ilustrar a primeira situação, é mostrado um exemplo em que se tem um tipo Pessoa com o *extent* Pessoas, com atributos nome, data de nascimento e sexo e com a operação idade. Deseja-se obter de Pessoas o conjunto com as diferentes idades de pessoas chamadas Carlos.

```
select distinct p.idade
from Pessoas p
where p.nome = “Carlos”
```

A possibilidade de utilizar o método idade na consulta é uma característica importante desta linguagem. Se a operação exigir parâmetros, esses podem ser passados entre parênteses. Utilizando *distinct* é retornado um *set*<*integer*>, caso contrário, seria retornado um *bag*<*integer*>. Se além do *distinct* fosse utilizado o termo *order by*, seria um *list*<*integer*>.

A seguir, é dado um exemplo que retorna um *set*<*struct*> com o par idade e sexo de cada pessoa chamada Carlos.

```
select distinct struct (id:p.idade, sx:p.sexo)
from Pessoas p
where p.nome = “Carlos”
```

Nem sempre é necessário o uso da cláusula *select-from-where*. Podem-se utilizar os *extents* e nomes de objetos diretamente para se fazer consultas. Por exemplo, para se obter o conjunto de todas as pessoas (*set*<Pessoa>), é necessário apenas o *extent* de Pessoa.

```
Pessoas
```

De mesmo modo, tendo-se o nome de objeto Presidente para uma determinada pessoa, pode-se consultá-la através desse nome, obtendo um objeto do tipo Pessoa como resposta.

```
Presidente
```

Ou ainda pode-se obter o salário dessa pessoa, tendo como retorno um literal.

```
Presidente.salario
```

Os possíveis resultados de uma extração são: um objeto com identidade, ou uma coleção desses, como também um literal ou uma coleção desses. Embora o resultado de um *select-from-where* seja sempre uma coleção, pode-se obter um objeto com identidade ou sem, aplicando, por exemplo, ao resultado de uma consulta que retorna uma coleção com apenas um elemento, a função *element*, a qual extrai um objeto ou literal dependendo de que tipo é o elemento da coleção.

3.4.2 Expressões de Caminho

As expressões de caminho servem para navegar através de relacionamentos (um-para-um) e objetos complexos, sendo representadas por “.” ou “→”. Podem existir chamadas a métodos no meio dessas expressões. A seguir, é mostrado um exemplo em que a partir de uma Pessoa p, deseja-se saber o nome da cidade onde o cônjuge dessa pessoa mora.

```
p.conjuge.endereco.cidade.nome
```

Essa consulta começa por uma pessoa p, pega seu cônjuge que é outra pessoa, vai dentro do atributo complexo do tipo endereço para pegar o objeto cidade e, então, acessa o nome da cidade.

Porém, deve-se ter cuidado quando se quiser como resultado um conjunto. Por exemplo, se o objetivo é obter o nome dos filhos de uma pessoa, deve-se fazer a consulta da seguinte forma:

```
select f.nome
from p.filhos f
```

Se for utilizado apenas "p.filhos.nome", o resultado será o nome da lista dos filhos e não o nome dos filhos. Como filhos é uma lista de referências, a consulta resultaria indefinido.

3.4.3 Valores Indefinidos

Uma propriedade não definida referencia o objeto *nil*, que, se for acessado, devolve como resultado o valor *undefined*. O *undefined* é um valor literal especial que, dentro da OQL, é tido como valor válido para qualquer tipo literal ou objeto. Algumas das regras que envolvem este valor são:

- *is_defined*(predicado) e *is_undefined*(predicado) testam se um predicado é definido ou não;
- se o predicado definido em uma cláusula *where* retorna *undefined*, isto é tratado como se fosse retornado *false*;
- o valor *undefined* é válido para a operação *count*, ou seja, é considerado da mesma forma que qualquer outro valor;
- o *undefined* é um valor válido para expressões de construção (construção de objetos, estruturas, *sets*, *lists*, *bags* e *arrays*);
- qualquer outra operação com operandos *undefined* resultam em *undefined*.

A seguir são mostrados alguns exemplos de consultas com seus respectivos resultados, considerando que o BD contém três pessoas: uma morando em Porto Alegre, outra em Caxias e a última com endereço *nil*.

```
select p
from Pessoas p
where p.endereco.cidade= "Caxias"
```

Retorna um bag com a pessoa que mora em Caxias.

```
select p.endereco.cidade
from Pessoas p
```

Retorna {"Porto Alegre", "Caxias", *undefined* }.

```
select p
from Pessoas p
where is_undefined(p.endereco.cidade)
```

Retorna um bag com a pessoa que não tem endereço definido.

3.4.4 Operadores

Esta seção apresenta quais os operadores que podem ser usados para formarem expressões com literais atômicos (tab. 3.1), expressões com coleções (tab. 3.2) e expressões com objetos e literais (tab. 3.3).

TABELA 3.1 - Operadores para literais atômicos

Forma de uso dos operadores	**Operadores**
Os operadores aritméticos podem ser definidos entre operandos que devem ser de um tipo inteiro ou ponto flutuante.	Unários: +, - , *abs*. Binários: +, -, *, /, mod.
Os operadores de comparação devem ser aplicadas sobre operandos do mesmo tipo ou tipos compatíveis.	Binários: =, !=, <, >, <=, >=.
Os operadores booleanos devem ser aplicados sobre operandos do tipo booleano.	Unário: *not*. Binários: *and*, *or*, *andthen*, *orelse.*
Os operadores de *string* devem ser aplicadas sobre operandos do tipo *string*.	Unários: *str* [*n*], *str*[*m*:*n*] (*str* representa uma *string*, *m* e *n* indicam inteiros). A primeira expressão extrai o caracter que se encontra na posição *n* dentro da *string*. A segunda extrai os caracteres da posição *m* até *n*. Binários: || (concatenação), + (concatenação), *like*, *in* (testa se um caracter pertence a uma string).

TABELA 3.2 - Operadores para coleções

Forma de uso dos operadores	**Operadores**
Operadores aplicados sobre coleções contendo qualquer tipo.	*exists*, *unique*, *element* (aplicada sobre uma coleção contendo apenas um elemento), *distinct*, *count*.
Operadores aplicados sobre coleções contendo qualquer tipo que testam se determinado elemento de tipo compatível está contido na coleção.	*in*, *for all*, *exists* (os dois últimos testam de acordo com uma condição especificada).
Operadores aplicados sobre coleções contendo qualquer tipo que testam se determinado elemento de tipo compatível é =, !=, >, <, >= ou <= a algum ou todos os elementos da coleção.	*some*, *any*, *all*.
Operadores unários aplicados sobre coleções contendo elementos de tipos numérico.	*min*, *max*, *sum*, *avg*.
Operadores binários específicos para *sets* e *bags*.	*union*, *intersect*, *except*.
Operadores binários de subconjunto e superconjunto específicos para *sets* e *bags*. Testam se um conjunto é subconjunto ou superconjunto do outro.	<, <=, >, >=.

Forma de uso dos operadores	Operadores
Operadores específicos para as coleções indexadas *list* ou *array*.	*first*, *last*, *col*[*n*], *col*[*m*:*n*] (*col* representa um *list* ou *array*, *m* e *n* indicam inteiros). Estes dois últimos operadores servem para extrair um elemento que se encontra na posição *n* dentro da coleção, ou conjunto de elementos que se encontram entre duas posições.
Operador específico para a coleção indexada *Dictionary*.	*e1*[*e2*], sendo *e1* uma coleção do tipo *Dictionary* e *e2* uma chave. Esta operação extrai o valor associado com a chave *e2* na coleção *e1*
Expressões de conversão de tipos de coleções.	*listtoset* (transforma um *list* em *set*), *flatten* (transforma uma coleção de coleções de um tipo T em uma coleção do tipo T). Uma coleção pode ser convertida para um *list* usando a cláusula *orderby*.

TABELA 3.3 - Operadores para objetos e literais

Forma de uso dos operadores	Operadores
Operadores de igualdade e de diferença podem ser aplicados na comparação entre dois objetos de tipos compatíveis, ou entre dois literais de tipos compatíveis.	= , !=.

3.4.5 Polimorfismo

O polimorfismo pode ser tratado de duas diferentes formas em uma consulta: *late binding* e indicação de classe. A figura 3.2 ilustra um exemplo para estas situações.

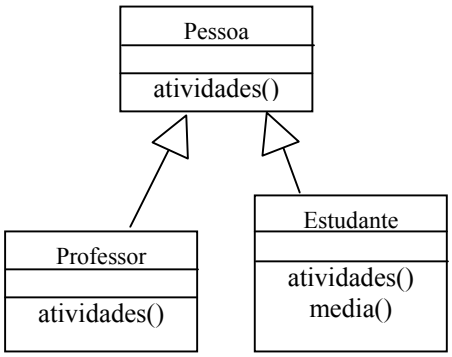

FIGURA 3.2 - Gráfico de representação de polimorfismo

No exemplo a seguir, se for utilizado *late binding*, o método a ser chamado é decidido em tempo de execução. Se *p* for um Estudante, a OQL chama a operação *atividades* definida em Estudante; caso *p* seja Professor, chama *atividades* de Professor; e se *p* for Pessoa, chama a operação *atividades* definida em Pessoa.

```
select p.atividades
from Pessoas p
```

Caso seja utilizada indicação de classe, então deve ser indicada explicitamente uma determinada classe. O avaliador analisa, em tempo de execução, se o objeto

pertence àquela classe (ou a uma de suas subclasses), para que seja chamado o método daquela classe. Por exemplo, assumindo que somente estudantes podem gastar seu tempo estudando, pode-se selecionar essas pessoas e obter suas médias. Abaixo é indicado explicitamente que as Pessoas são da classe Estudante.

```
select ((Estudante)p).média
from Pessoas p
where "estudar" in p.atividades
```

3.5 *Bindings*

Os *bindings* integram a programação da aplicação e do banco de dados em um único ambiente, ou seja, baseiam-se no uso da própria sintaxe e semântica da linguagem de programação para fornecer capacidades de banco de dados. Pode-se dizer, também, que os *bindings* ODMG são uma extensão da linguagem de programação base. Isso caracteriza uma grande facilidade no uso do padrão ODMG, pois o programador não precisa aprender uma nova linguagem para lidar com o BD [BAR 98].

Há mapeamentos para C++, Java e Smalltalk. Por exemplo, no caso do C++ a ODL abstrata definida anteriormente, que define as características lógicas dos objetos e as operações para manipulá-los, é mapeada para a ODL/C++ (tanto essa quanto a ODL/Java e ODL/Smalltalk foram desenvolvidas para se ter uma adequação suave dentro da sintaxe declarativa destas LPOO).

3.6 Considerações Finais

Neste capítulo, foram apresentados os componentes do padrão ODMG, dando ênfase ao Modelo de Objetos, à ODL e à OQL, pois esses foram os componentes estendidos para suportarem as características do TVM, conforme será mostrado no próximo capítulo.

Foram apresentadas algumas características do Modelo de Objetos como: o conceito de tipos, os tipos de herança, os tipos de objetos, os tipos de literais e a definição do estado e do comportamento de um objeto. Em relação à ODL foi mostrado brevemente como podem ser definidas classes. Sobre a OQL foi demostrado o potencial da linguagem e a facilidade de se realizar consultas sobre objetos.

4 TV_ODMG

Com o objetivo de possibilitar que o padrão ODMG suporte versionamento de objetos e características temporais, baseando-se no Modelo Temporal de Versões, foram estendidos o Modelo de Objetos, a Linguagem de Definição de Objetos (ODL) e a Linguagem de Consulta (OQL) do padrão. Essas extensões foram denominadas TV_OM, TV_ODL e TV_OQL, originando, assim, o TV_ODMG.

4.1 TV_OM

Esta extensão do Modelo de Objetos possibilita que se tenha os objetos já definidos no ODMG com suas características originais, como também os objetos temporais versionáveis.

As características gerais do TV_OM são:

- As classes definidas no TVM foram acrescentadas ao ODMG, com exceção da *Object*.
- No TVM, um instante de tempo é indicado usando o tipo *instant*, o qual representa data e hora. Nessa extensão, é usado um tipo correspondente e que já existe no ODMG, o *timestamp*.
- O tempo pode estar associado aos objetos temporais versionáveis e aos valores de suas propriedades.
- Suporta tanto o tempo de transação como o tempo de validade, ou seja, é bitemporal.
- Os objetos temporais versionáveis podem ser excluídos apenas logicamente, a fim de se manter o histórico dos mesmos.
- As regras de integridade temporal e de exclusão lógica (seção 2.1.1), definidas no TVM, são respeitadas para objetos temporais versionáveis.
- Os estados possíveis de uma versão ou de um objeto temporal versionável sem versões são: *working*, *stable*, *consolidated* e *deactivated*.
- Suporta o conceito de configuração.

A forma de associação do tempo segue o que foi especificado pelo TVM:

- A cada valor de uma propriedade temporal de um objeto temporal versionável são associados os tempos de transação inicial e final, assim como os tempos de validade inicial e final.
- Ao objeto temporal versionável é associada uma propriedade temporal que indica quando o objeto está ativo, ou não, no banco de dados, pois ele pode ser excluído logicamente e depois reativado.

4.1.1 Tipos

Para que esta extensão suporte as características do TVM foram acrescentados novos tipos objetos ao TV_ODMG: *TemporalVersion, VersionedObjectControl,*

TemporalObject, OIDt, TVEntity, TemporalLabel, InstantAttribute, InstantRelationship, *TemporalAttribute* e *TemporalRelationship*. Os quatro primeiros tipos lidam com versões, herança por extensão e tempo de vida dos objetos temporais versionáveis. O *TVEntity* trata os nomes das entidades presentes nas classes temporais versionáveis. O *TemporalLabel* lida com os tempos de transação e de validade referentes a vida dos objetos e aos valores das propriedades temporais. Os últimos quatro tipos fornecem suporte às propriedades temporais. Além desses, foram incluídos mais alguns tipos que correspondem aos tipos literais do ODMG, os quais lidam com propriedades temporais, como será explicado na próxima seção. Todos esses tipos são especificados por classes que são descritas ao longo do texto.

4.1.2 Hierarquia de Classes

As classes definidas no TVM foram acrescentadas à extensão do padrão ODMG, com exceção da *Object*, pois o ODMG já define uma interface *Object*. Isso implica a impossibilidade de se ter herança por extensão entre as classes normais (classes que não são temporais versionáveis) como no TVM, no qual são definidas determinadas características na classe *Object* para que as classes normais possam ter esse tipo de herança. Como nesta extensão é usada a *Object* do ODMG sem alterá-la, para que essa continue compatível com aplicações existentes, então é imposta essa restrição ao modelo. As características especificadas na *Object* do TVM para dar suporte à herança por extensão foram definidas na classe *TemporalVersion* do TV_ODMG. Assim, apenas as classes temporais versionáveis poderão apresentar herança por extensão, como também, de acordo com o TVM, apenas essas possuem suporte a tempo e versões.

Entretanto, a *TemporalObject* é a classe que fornece suporte para a definição de características temporais, portanto, as classes básicas do TV_ODMG são denominadas classes normais especiais ou classes temporais porque não são classes temporais versionáveis (pois não herdam de *TemporalVersion* que fornece suporte a versões) mas possuem acesso a esse suporte ao tempo.

A hierarquia de classes básicas do TV_ODMG é ilustrada na figura 4.1, na qual *ApplicationClass*(ODMG) representa possíveis classes normais, e *ApplicationTVClass* representa possíveis classes temporais versionáveis da aplicação.

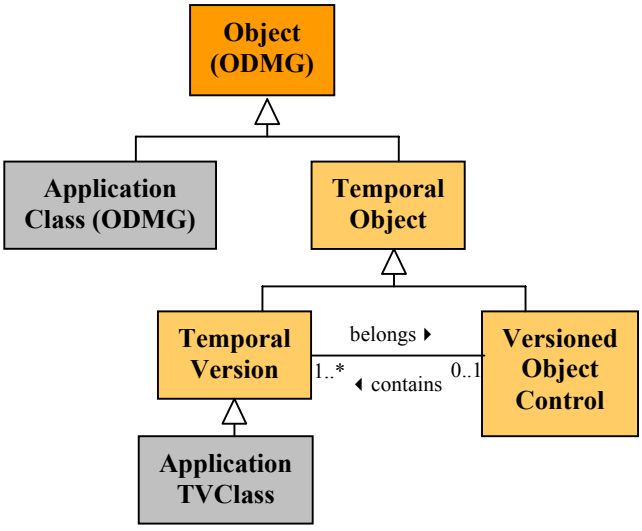

FIGURA 4.1 - Hierarquia de Classes

A hierarquia de classes temporais do TV_ODMG é ilustrada na figura 4.2. É mostrado um relacionamento entre *TemporalAttribute/TemporalRelationship* e *InstantAttribute/InstantRelationship*, que indica o valor corrente de uma propriedade temporal, e outro, que representa os vários valores de uma propriedade temporal ao longo do tempo. Uma das alterações em relação à hierarquia apresentada pelo TVM é que os relacionamentos de agregação foram substituídos por relacionamentos de associação, pois esta extensão não permite aquele tipo de relacionamento entre classes normais, como será explicado posteriormente.

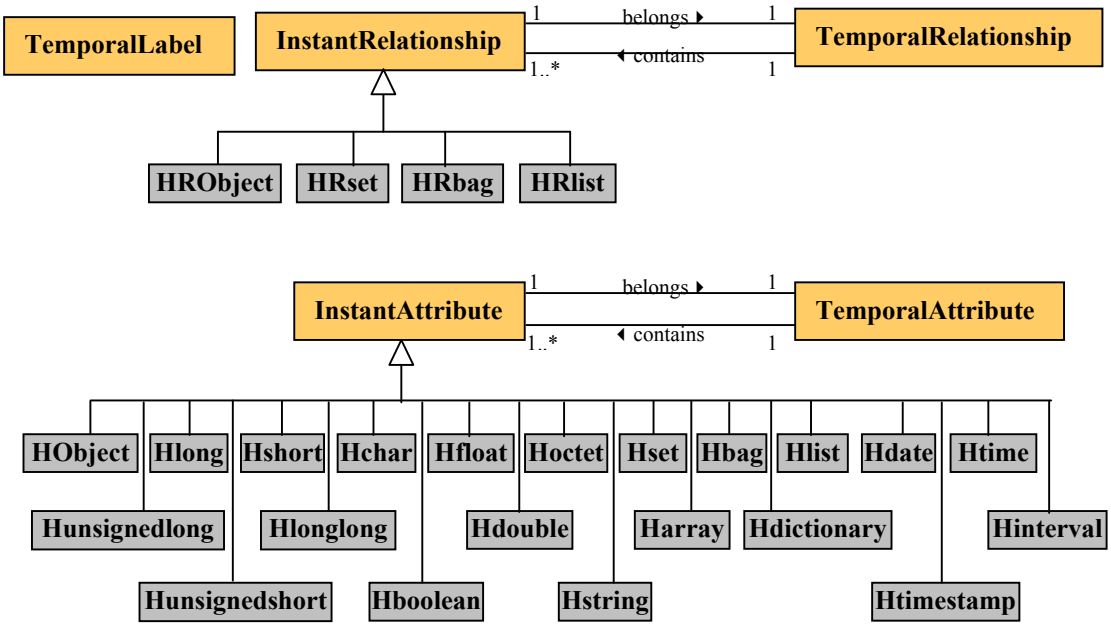

FIGURA 4.2 - Hierarquia de Classes Temporais

O TVM apresenta uma limitação que permite uma propriedade ser definida como temporal apenas se for declarada como de um tipo objeto, pois as classes *InstantAttribute* e *InstantRelationship* (responsáveis pelo armazenamento dos valores do histórico das propriedades temporais) são capazes de guardar apenas o histórico de tipos objetos. Dessa forma, foram definidas classes específicas para armazenar os valores dos históricos dos tipos literais. No caso de atributos temporais, classes específicas que representam os tipos literais foram definidas como subclasses de *InstantAttribute*. Também, foi definida uma subclasse para suportar os tipos objetos, o que no TVM, era feito pela própria classe *InstantAttribute*. Para relacionamentos temporais, foram definidas subclasses específicas apenas para os tipos literais coleções *set*, *bag*, *list*, como também para um tipo objeto, pois por definição do padrão, um relacionamento nunca pode ser definido como de um tipo literal atômico, literal estruturado ou dos tipos literais coleções *array* e *dictionary*.

4.1.3 Herança por Refinamento e Herança por Extensão

O ODMG trabalha com o conceito de herança por refinamento. Como o TVM usa o conceito de herança por extensão (ou relacionamento de herança por extensão), então esse foi acrescentado ao TV_ODMG, sendo seu controle feito através dos novos tipos

acrescentados ao padrão. Assim como no TVM, não é possível que um par de classes apresente os dois tipos de herança ao mesmo tempo, devido às diferenças nos conceitos.

Quanto ao comportamento de classes e interfaces em relação à herança, o TV_ODMG propõe que:

- Uma interface pode apresentar apenas herança do tipo *is-a*, podendo herdar apenas de outra(s) interface(s).
- Uma classe normal (não-temporal e não-versionável) pode ter herança do tipo *extends*, podendo herdar de apenas uma outra classe normal; e herança *is-a*, podendo herdar de uma ou várias interfaces.
- Uma classe temporal versionável pode ter apenas herança por extensão, sendo possível herdar de apenas uma classe desse mesmo tipo. Porém, há a exceção de que todas as classes temporais versionáveis herdam por refinamento (*extends*), implicitamente e diretamente, de *TemporalVersion*.

No ODMG, se uma classe não herda de outra, então ela é incluída como subclasse da interface *Object*. Isso continua sendo válido no TV_ODMG, porém, há mais uma característica a ser analisada antes de definir uma classe como subclasse de *Object*. Se houver a cláusula *hasVersion* na definição da classe, então ela é incluída como subclasse de *TemporalVersion*.

As classes normais não podem apresentar herança por extensão porque herdam de *Object*, a qual não possui suporte para esse conceito. No caso das interfaces, esse tipo de herança nem faz sentido, pois elas não definem estado, apenas comportamento.

4.1.4 Objetos Temporais Versionáveis

Um objeto temporal versionável é um objeto atômico que suporta versões e tempo. Esse tipo de objeto aparece, somente, como instância de alguma subclasse de *TemporalVersion*, pois essa não deve ser diretamente instanciada, ou seja, o usuário não deve lidar diretamente com ela.

Um objeto temporal versionável pode apresentar-se como um objeto versionado, uma versão ou um objeto ainda sem versões (embora possua capacidade para tê-las).

A seguir, é mostrado um exemplo de duas classes temporais versionáveis cujas instâncias serão objetos temporais versionáveis:

```
class Veículo hasVersion
  (extent veículos)
{
attribute string motor;
temporal_attribute string combustível;
};

class Automóvel hasVersion byExtension Veículo correspondence n:m
   (extent automóveis)
{
attribute short nro_volumes;
temporal_attribute string acessórios;
};
```

A classe Veículo apresenta o atributo não-temporal motor e o atributo temporal combustível. A classe Automóvel possui o atributo não-temporal nro_volumes e o atributo temporal acessórios e herda por extensão de Veículo. Ambas possuem a cláusula *hasVersion*; logo, herdam por refinamento de *TemporalVersion*.

Todos os objetos temporais versionáveis devem ser persistentes e, conforme o TVM, possuem o atributo temporal *alive* associado.

4.1.5 Identificadores de objetos

Todos os objetos sendo normais, ou não, possuem um OID definido pelo sistema. No TV_ODMG, é definido mais um identificador através do atributo *tvOID* (definido em *TemporalVersion*) que é do tipo *OIDt*. Esse identificador adicional mantém as características definidas pelo TVM. Porém, apenas os objetos temporais versionáveis o possuem, diferentemente do TVM, em que todos os objetos o possuem por ser definido na classe *Object*.

4.1.6 *Extent*

O *extent* é o conjunto de todas as instâncias de uma classe. No caso dos objetos temporais versionáveis, tanto os objetos ativos como os inativos (que foram excluídos logicamente) estarão no *extent*. A condição para se obter apenas os objetos ativos, ou todos, deve ser informada na consulta.

4.1.7 Chaves

Uma chave é representada por um determinado atributo cujo valor identificará o objeto e será único dentro de um *extent*. O conceito de chave não se aplica aos objetos temporais versionáveis, pois com a derivação de uma versão, seria gerado um erro, já que não é possível ter o mesmo valor para a chave em dois objetos pertencentes a um mesmo *extent*. Além disso, o TVM não prevê esse conceito.

4.1.8 Hierarquia Completa de Tipos

A hierarquia completa de tipos da extensão do padrão ODMG é ilustrada a seguir (fig. 4.3). Os tipos apresentados em fonte normal são concretos e diretamente instanciáveis, os que estão em negrito são abstratos, e aqueles que estão em itálico são concretos e não devem ser instanciados diretamente pelo usuário. Os tipos que estendem a hierarquia estão com um sinal (+) ao lado:

Literal_type
- **Atomic_literal**
  - long
  - long long
  - short
  - unsigned long
  - unsigned short
  - float
  - double
  - boolean
  - octet
  - char
  - string
  - enum<>
- **Collection_literal**
  - set<>
  - bag<>
  - list<>
  - array<>
  - dictionary<>
- **Structured_literal**
  - date
  - time
  - timestamp
  - interval
  - structure<>

Object_type
- **Atomic_object**
- \+ *TemporalObject*
- \+ *TemporalVersion*
- \+ *VersionedObjectControl*
- \+ *InstantAttribute*
- \+ *HObject*
- \+ *Hlong*
- \+ *Hlonglong*
- \+ *Hshort*
- \+ *Hunsignedlong*
- \+ *Hunsignedshort*
- \+ *Hfloat*
- \+ *Hdouble*
- \+ *Hboolean*
- \+ *Hoctet*
- \+ *Hchar*
- \+ *Hstring*
- \+ *Hset*
- \+ *Hbag*
- \+ *Hlist*
- \+ *Harray*
- \+ *Hdictionary*
- \+ *Hdate*
- \+ *Htime*
- \+ *Htimestamp*
- \+ *Hinterval*
- \+ *InstantRelationship*
- \+ *HRObject*
- \+ *HRset*
- \+ *HRbag*
- \+ *HRlist*
- \+ *TemporalAttribute*
- \+ *TemporalRelationship*
- \+ *TemporalLabel*
- \+ *OIDt*
- \+ TVEntity
- **Collection_object**
  - Set<>
  - Bag<>
  - List<>
  - Array<>
  - Dictionary<>
- **Structured_object**
  - Date
  - Time
  - Timestamp
  - Interval

FIGURA 4.3 - Hierarquia completa de tipos estendida

No TVM, há a definição de uma classe chamada *Name*, a qual serve para dar nomes aos objetos. No entanto, essa capacidade já existe no ODMG; pois são definidas

operações na interface *Database* para lidar com nomes de objetos. Logo, essa classe *Name* não foi acrescentada ao TV_ODMG.

4.1.9 Modelando Propriedades de Estado

No armazenamento dos valores do histórico de uma propriedade, a vantagem de se ter os tempos de transação e de validade associados diretamente a cada um desses valores, em vez de ao objeto, é que as alterações ocorridas no valor de uma propriedade implicam o armazenamento apenas daquela informação. Se esses tempos forem vinculados ao objeto, terão que ser armazenados os valores de todas as propriedades do objeto devido à alteração do valor de apenas uma, causando repetição de informações.

Os valores das propriedades temporais possuem associados os tempos de transação inicial e final, assim como os tempos de validade inicial e final, através dos rótulos pré-definidos *tTimei*, *tTimef, vTimei* e *vTimef*. O funcionamento desses rótulos é semelhante ao definido no TVM.

No TV_ODMG, o tempo de validade inicial do primeiro valor de cada propriedade será sempre igual à validade inicial do objeto, e o tempo de validade final ficará inicialmente em aberto (*nil*). Na atualização do valor da propriedade, é possível determinar os tempos de validade para esse valor. Dependendo dos tempos determinados, é feita uma correção nos tempos de validade dos valores do histórico que possuírem tempos de validade que colidam com os determinados. Para corrigir ou finalizar tempos de validade, sempre são criadas no histórico novas instâncias para o mesmo valor da propriedade temporal, a fim de poder representar as alterações desejadas sem perder a história do valor (de acordo com a seção 2.1.1).

O funcionamento em relação ao tempo de transação e à exclusão de objetos permanece igual ao definido no TVM.

4.1.9.1 Atributos

Apenas as classes temporais versionáveis podem ter atributos temporais, podendo essas terem tanto atributos temporais como não-temporais. As classes normais não podem ter propriedades temporais porque o controle do histórico dessas propriedades é feito por métodos definidos na classe *TemporalObject* e as classes normais não tem acesso a esses métodos. Porém, há uma exceção em relação às classes básicas (*TemporalObject*, *TemporalVersion* e *VersionedObjectControl*) do TV_ODMG, as quais são consideradas classes normais especiais ou classes temporais, pois essas possuem acesso aos métodos definidos na *TemporalObject*, possibilitando que tenham atributos temporais.

Além disso, é possível definir valores *default* para os atributos.

4.1.9.2 Relacionamentos

O relacionamento definido no ODMG é o de associação, o qual representa um vínculo entre dois tipos. Assim, uma instância de uma classe é associada a uma ou mais instâncias de uma outra classe.

No TV_ODMG foi introduzido o relacionamento de agregação, que é uma forma especial de associação utilizada para representar que um tipo de objeto é composto, pelo menos em parte, de outro em uma relação de todo/parte [FUR 98]. Segundo [JOR 98], é possível definir uma agregação tanto como uma referência quanto como um

relacionamento, e as partes podem, ou não, referenciar seu todo. Dessa forma, foi definido nesta extensão que os componentes referenciam o todo, pois, como mencionado anteriormente, o relacionamento de agregação é uma especialização do de associação, e este último exige que seja definido o inverso do relacionamento.

A declaração de uma agregação é similar a uma associação. A seguir, é mostrado um exemplo com uma classe Pedido e outra ItemPedido; a primeira representa a agregação (todo), e a segunda, a componente (parte).

```
class Pedido hasVersion
{
 aggregateOf set<ItemPedido> contem
        inverse ItemPedido::pertence;
};

class ItemPedido hasVersion
{
componentOf Pedido pertence
        inverse Pedido:: contem;
};
```

Os relacionamentos de associação podem ser estabelecidos entre duas classes normais, ou entre classes normais e temporais versionáveis, ou entre duas temporais versionáveis. Entretanto, os de agregação são permitidos apenas entre classes temporais versionáveis.

É preciso diferenciar os relacionamentos de associação e os de agregação, principalmente por causa das configurações que utilizam esse último tipo. As classes normais não podem ter esse tipo de relacionamento porque há restrições na exclusão de objetos que estão envolvidos em relacionamentos de agregação, e não é possível impor essas restrições para as classes normais. Porém, isso não impõe uma grande perda, pois é importante ressaltar que os objetos das classes normais não poderiam ser configurações; apenas os objetos temporais versionáveis é que podem porque herdam da classe *TemporalVersion*, na qual são definidas as características para dar suporte a configurações.

Tanto relacionamentos de associação quanto de agregação podem ser definidos como temporais, sendo que, nesse caso, os seus inversos também devem ser definidos como temporais. Porém, apenas classes temporais versionáveis aceitam declarações desse tipo. As classes normais não podem apresentar relacionamentos temporais porque não possuem suporte ao tempo.

As cardinalidades são as mesmas suportadas pelo ODMG. Abaixo, é mostrado um exemplo com duas classes que apresentam relacionamentos de associação temporais:

```
class Departamento hasVersion
{
temporal_relationship set<Empregado> emprega
        inverse Empregado:: trabalha_em;
};

class Empregado hasVersion
{
temporal_relationship Departamento trabalha_em
        inverse Departamento:: emprega;
};
```

A seguir, é mostrado o exemplo de agregação anterior, mas, agora, com relacionamentos temporais:

```
class Pedido hasVersion
{
 temporal_aggregateOf set<ItemPedido> contem
        inverse ItemPedido::pertence;
};

class ItemPedido hasVersion
{
temporal_componentOf Pedido pertence
        inverse Pedido:: contem;
};
```

4.1.10 Relações

A tabela 4.1 mostra as possíveis relações entre classes temporais versionáveis, classes normais e interfaces.

TABELA 4.1 - Resumo das Relações

	Associação não-temporal	Associação temporal	Herança *extends*	Herança *is-a*	Herança por extensão	Agregação (temporal ou não)
classe normal x classe normal	√		√			
classe normal x classe_tv	√		√ *			
classe_tv x classe_tv	√	√			√	√
interface x classe normal	√			√ **		
interfaces x interfaces	√			√		

(*) É permitido apenas quando a superclasse é a classe normal *TemporalVersion*.

(**) É permitido apenas quando a classe normal é que herda da interface.

4.1.11 Definição das Classes Acrescentadas ao ODMG

Nesta seção, é apresentada a definição das classes que especificam os novos tipos acrescentados à hierarquia de tipos do padrão ODMG. Algumas operações dessas classes envolvem acesso a metadados, os quais armazenam informações sobre as características acrescentadas ao padrão, tais como: as classes temporais versionáveis, os relacionamentos de herança por extensão, os relacionamentos de agregação, os relacionamentos temporais de associação e os atributos temporais.

Algumas das classes, mostradas nesta seção, não apresentam todas as suas operações, mas a definição completa de todas as classes pode ser encontrada no anexo 1.

Embora os tipos *TemporalObject* e *TemporalVersion* não devam ser diretamente instanciados, ou seja, devam ser instanciados apenas através de suas subclasses, eles são definidos como classes para que as subclasses possam herdar seu estado, ou seja, suas

propriedades. Pois, se fossem definidos como interfaces, suas propriedades deveriam ser novamente definidas nas subclasses, isto é, nas classes da aplicação do usuário. Essas duas classes não apresentam *extent* porque não se pretende instanciá-las diretamente. Outra característica que envolve a hierarquia de classes básicas é que o usuário não deve definir subclasses de *TemporalObject*, assim como de *VersionedObjectControl*.

A classe *TemporalObject* define características para lidar com o tempo de vida dos objetos temporais versionáveis, assim como com a história deles. Possui a definição do atributo temporal *alive*, que indica quando um objeto está ativo ou não na base de dados; esse atributo deve ser inicializado com o valor *default true*.

Em relação ao TVM, foram excluídos da *TemporalObject* os construtores que possuem parâmetros a serem passados para a classe *Object*. Pois, no TV_ODMG, as características que são definidas na *Object* do TVM estão na *TemporalVersion*, como explicado anteriormente. Foram incluídas as operações *setTemporalAttribute* e *setTemporalRelationship* com parâmetros de tempo para possibilitar que o usuário especifique os tempos de validade para um novo valor de uma propriedade temporal. Se essas operações forem chamada sem passagem de parâmetros para os tempos, é porque a validade inicial será o momento atual. Nesse caso, as operações atualizam a instância da classe da aplicação com o valor atual da propriedade. Além disso, foram definidas novas opções dessas operações para a passagem do valor das propriedades de tipos literais.

```
class TemporalObject
{
temporal_attribute boolean alive default true;

Object TemporalObject ( );
boolean delete ( );
boolean getAlive ( );
timestamp getLifetimeI ( );
timestamp getLifetimeF ( );
set<InstantAttribute> getObjectHistory ( );
void closeTemporalLabels ( );
set<InstantAttribute> getAttributeHistory (in string attrName);
InstantAttribute getAttributeValueAt (in string attrName, in timestamp i);
set<InstantRelationship> getRelationshipHistory (in string relName);
InstantRelationship getRelationshipValueAt (in string relName, in timestamp i);
void setTemporalAttribute (in string attrName, in Object newValue);
void setTemporalAttribute (in string attrName, in Object newValue, in timestamp iValidTime,
    in timestamp fValidTime);
void setTemporalAttribute (in string attrName, in long newValue);
void setTemporalAttribute (in string attrName, in long newValue, in timestamp iValidTime,
    in timestamp fValidTime);
...
void setTemporalAttribute (in string attrName, in set<t> newValue);
void setTemporalAttribute (in string attrName, in set<t> newValue, in timestamp iValidTime,
    in timestamp fValidTime);
...
void setTemporalAttribute (in string attrName, in date newValue);
void setTemporalAttribute (in string attrName, in date newValue, in timestamp iValidTime,
    in timestamp fValidTime);
...
void setTemporalRelationship (in string relName, in Object newValue);
void setTemporalRelationship (in string relName, in Object newValue,
     in timestamp iValidTime, in timestamp fValidTime);
void setTemporalRelationship (in string relName, in set<Object> newValue);
void setTemporalRelationship (in string relName, in set<Object> newValue,
```

```
    in timestamp iValidTime, in timestamp fValidTime);
void setTemporalRelationship (in string relName, in bag<Object> newValue);
void setTemporalRelationship (in string relName, in bag<Object> newValue,
    in timestamp iValidTime, in timestamp fValidTime);
void setTemporalRelationship (in string relName, in list<Object> newValue);
void setTemporalRelationship (in string relName, in list<Object> newValue,
    in timestamp iValidTime, in timestamp fValidTime);
};
```

A classe *TemporalVersion* define características para lidar com herança por extensão e com versões. Foi introduzido o relacionamento *belongs* para ligar o objeto versionado e suas versões ao seu correspondente controle. Essa classe incorporou o atributo *tvOID* e algumas operações que no TVM eram definidas em *Object*. As operações realocadas são: *getClassName*, *getClassId*, *getCorrespondenceDesc*, *getCorrespondenceAsc*, *findVersion*, *getEntityId*, *getCorrespondence*, *verifyEntityName* e *getTVOid*. O comportamento da *getEntityId* foi alterado, pois agora a consulta para retornar o identificador da entidade é feita na classe *TVEntity* e não mais nos metadados.

Os atributos *ascendant* e *descendant* servem para definir a ligação entre os objetos de uma superclasse e uma subclasse em uma hierarquia de extensão. Na definição da cláusula *byExtension*, a qual especifica a herança por extensão, é estabelecido o número de ascendentes e descendentes que os objetos das classes envolvidas poderão ter; depois de especificado, isso deve ser controlado pelo sistema.

Em seus construtores, devem ser inicializados os atributos *configuration* e *status* com os respectivos valores *default false* e *w*. O valor *w* corresponde ao estado *working*, os outros valores para o status seriam *s*, *c*, *d*, respectivamente para *stable*, *consolidated* e *deactivated*.

Na especificação a seguir, os três primeiros construtores da classe criam objetos sem versões. Os dois últimos construtores geram um objeto versionado ou versões de um objeto, os quais são chamados pela operação *derive*.

```
class TemporalVersion extends TemporalObject
{
attribute OIDt tvOID;
temporal_attribute set<Object> ascendant;
temporal_attribute set<Object> descendant;
temporal_attribute set<Object> successor;
temporal_attribute char status default 'w';
attribute set<Object> predecessor;
attribute boolean configuration default false;
relationship VersionedObjectControl belongs
     inverse VersionedObjectControl contains;

Object TemporalVersion ( );
Object TemporalVersion (in set<Object> ascendant);
Object TemporalVersion (in string entityName);
Object TemporalVersion (in short entityId, in short classId, in short versionId);
                          //serve para gerar um objeto versionado ou uma versão
Object TemporalVersion (in set<Object>  predecId, in set<Object> ascendId,
    in boolean config);          //serve para gerar uma versão

boolean addAscendant (in Object ascendId);
boolean addDescendant (in Object descendId);
void addSucessor (in Object succId);
```

```
void delete (in boolean allReferences);
void deleteObjectTree (in boolean allReferences);
Object derive (in set<Object> versionId);
Object derive (in set<Object> versionId, in set<Object> ascendId, in boolean config);
set<Object> getAscendant ( );
set<Object> getAscendant (in boolean all);
Object getConfiguration ( );
boolean isConfiguration ( );
set<Object> getCompletObject ( );
set<Object> getDescendant ( );
set<Object> getDescendant (in boolean all);
set<Object> getPredecessor ( );
char getStatus ( );
set<Object> getSucessor (in boolean onlyConfigured);
Object getVersionedObjectId ( );
Object getVersionedObjectControl ( );
boolean isDeleteAllowed (in boolean allReferences);
boolean isDeleteTreeAllowed (in boolean allReferences);
void promote (in boolean allAscendant, in boolean allReferenced);
boolean removeAscendant (in Object ascendId);
void removeDescendant (in Object descendId);
void removeSuccessor (in Object succId);
boolean restore (in Object oid);
void setAscendant (in Object ascendId);
void setDescendant (in Object descendId);
void setStatus (in char newStatus);
boolean setSuccessor (in set<Object> succId);
boolean verifyAscendId (in set<Object> ascendId);
OIDt getTVOid ( );
string getCorrespondence (in string ascendClassName);
string getCorrespondenceAsc (in string ascendClassName);
string getCorrespondenceDesc (in string descendClassName);
string getClassName ( );
short getEntityId (in string entityName);
boolean verifyEntityName (in string entityName);
short findVersion (in short entityId, in short classId);
string getClassId ( );      //retorna o identificador da subclasse de TemporalVersion
                            //que foi instanciada
};
```

A *VersionedObjectControl* define o controle dos objetos versionados. Esse tipo não deve ser diretamente instanciado pelo usuário, apenas pelo sistema. Em seu construtor devem ser inicializados os atributos *versionCount*, *configurationCount*, *userCurrentFlag* e *nextVersionNumber* com os respectivos valores *default* 2, 0, *false* e 3.

Comparando ao TVM, foi excluído um dos construtores, e o outro foi alterado, sendo esse chamado pela operação *derive* da *TemporalVersion*, se o objeto ainda não possuir versões.

```
class VersionedObjectControl extends TemporalObject
   (extent VersionedObjectControls)
{
temporal_attribute short versionCount default 2;
temporal_attribute short configurationCount default 0;
temporal_attribute Object firstVersion;
temporal_attribute Object lastVersion;
temporal_attribute Object currentVersion;
```

```
temporal_attribute boolean userCurrentFlag default false;
attribute short nextVersionNumber default 3;
relationship set<TemporalVersion> contains
     inverse TemporalVersion belongs;

Object VersionedObjectControl (in Object firstV, in Object lastV);
void delete(in short entityId, in short classId);
short getConfigurationCount ( );
short getVersionCount ( );
short getNextVersionNumber ( );
boolean getUserCurrentFlag ( );
Object getFirstVersion ( );
Object getLastVersion ( );
Object getCurrentVersion ( );
void setCurrentVersion ( );
void setCurrentVersion (in Object newVersionId);
void updateConfigurationCount ( );
void setFirstVersion (in Object first);
void setLastVersion (in Object last);
void setUserCurrentFlag (in boolean flag);
void updateVersionCount ( );
void updateNextVersionNumber ( );
};
```

A classe *TVEntity* foi definida para que sejam armazenados o número e o nome das entidades. O construtor com o parâmetro *name* é usado para o usuário definir um nome de entidade, e o outro é usado pelo sistema quando o usuário não define um nome específico. As outras operações recuperam os valores dos atributos definidos na classe.

```
class TVEntity
  (extent TVEntitys)
{
attribute short id;
attribute string name;

Object TVEntity (in string name);
Object TVEntity ( );      //cria um entidade com um nome definido pelo sistema
short getId ( );
string getName ( );
};
```

A classe *OIDt* declara um atributo para armazenar o valor do identificador adicional definido para os objetos temporais versionáveis, o qual apresenta a estrutura "id entidade, id classe, nº versão". Comparando ao TVM, foi acrescentada a operação *getEntityName*, a qual obtém o número da entidade através de *getEntityNr*, consulta em *TVEntity* o nome da entidade que corresponde àquele número e o retorna.

```
class OIDt
  (extent OIDts)
{
attribute string value;

Object OIDt (in string oid);
short getClassNr ( );
short getEntityNr ( );
short getVersionNr ( );
string getValue ( );
```

```
string getEntityName ( );
};
```

A classe *TemporalLabel* declara atributos para armazenar os tempos de transação e de validade definidos para os valores das propriedades temporais e para os tempos de vida dos objetos temporais versionáveis.

```
class TemporalLabel
    (extent TemporalLabels)
{
attribute timestamp tTimei;
attribute timestamp tTimef;
attribute timestamp vTimei;
attribute timestamp vTimef;

Object TemporalLabel (in timestamp iValidTime, in timestamp fValidTime,
     in timestamp iTransTime, in timestamp fTransTime);

timestamp getTTimei ( );
timestamp getTTimef ( );
timestamp getVTimei ( );
timestamp getVTimef ( );
void setTTimef (in timestamp i);
void setVTimef (in timestamp i);
};
```

As classes *TemporalRelationship* e *TemporalAttribute* armazenam o histórico das propriedades temporais, e as classes especializadas de *InstantAttribute* e *InstantRelationship* armazenam cada um dos valores que formam esse histórico.

No momento da instanciação de uma classe da aplicação, o sistema cria uma instância de *TemporalAttribute* ou *TemporalRelationship* para cada uma das propriedades temporais que a instância da classe da aplicação em questão, terá. Após isso, é criada uma instância de uma das classes especializadas de *InstantAttribute* (de acordo com o tipo do atributo) ou de *InstantRelationship* (de acordo com o tipo do relacionamento) para o primeiro valor da propriedade e uma instância de *TemporalLabel* para atribuição dos valores do rótulo temporal. Para cada um dos valores seguintes da propriedade, também haverá uma instância em uma dessas classes especializadas, para que seja armazenado o histórico de valores, como também, serão associados os tempos do rótulo temporal. Esse primeiro valor da propriedade é o valor informado na instanciação do objeto para ser válido a partir daquele momento (início da vida do objeto), ou *nil*, caso não seja informado, sendo o tempo de validade inicial desse último, também igual ao determinado para o início da vida do objeto.

As operações *setTemporalAttribute* e *setTemporalRelationship* (definidas em *TemporalObject*) atualizam o valor de uma propriedade temporal, como mencionado anteriormente. Essas operações chamam a operação *updateValue* (definida em *TemporalAttribute e TemporalRelationship*), com ou sem passagem de parâmetros para os tempos, conforme o caso.

A operação *updateValue* é responsável por criar, de acordo com as regras da seção 2.1.1, instâncias das classes especializadas de *InstantAttribute* e *InstantRelationship* para formarem o histórico de valores de uma propriedade temporal. No TV_ODMG, essa operação é definida com duas opções, uma sem parâmetros para passagem dos tempos de validade, e a outra com. A primeira pode ser invocada pelo sistema ou

indiretamente pelo usuário (pois o usuário chama *setTemporalAttribute* ou *setTemporalRelationship*, que chamam *updateValue*), quando não se tem uma validade inicial definida para o valor da propriedade. Nesse caso, é assumido o momento atual como validade inicial. Enquanto a segunda pode ser chamada indiretamente pelo usuário quando se deseja passar tempos de validade definidos.

As classes *TemporalAttribute* e *TemporalRelationship* definem um construtor que recebe o valor e a validade inicial e outro que recebe o valor, a validade inicial e a validade final, sendo que o uso de um ou de outro depende de a validade final do valor ter sido ou não determinada. Esses valores recebidos por parâmetros são usados para criar uma instância de uma das classes especializadas de *InstantAttribute* ou de *InstantRelationship* para representar o primeiro valor da propriedade. As instâncias seguintes dessas classes são criadas pela operação *updateValue*, como será descrito a seguir. Algumas das operações definidas em ambas as classes são descritas a seguir:

- *updateValue* – essa operação sem parâmetros para tempos deve chamar os construtores de uma das classes especializadas de *InstantAttribute* ou de *InstantRelationship* (conforme o caso) que possuem parâmetros para valor e tempo de validade inicial; deve ser passado para esses o respectivo valor e uma validade inicial igual ao momento atual. Esses construtores são chamados também quando a operação *updateValue* com parâmetros para tempos receber uma validade inicial válida e uma validade final igual a *nil*; entretanto, nesse caso, é passada para os construtores essa validade inicial válida, ao invés do momento atual. Porém, se ambos os tempos de validade forem válidos, então os construtores chamados são aqueles que possuem parâmetros para valor, tempo de validade inicial e final. No caso da atualização de um atributo de tipo literal atômico, o *updateValue* recebe o valor numa *string*. Entretanto, a operação deve fazer um *cast* dessa *string* para o tipo do atributo (essa informação está no atributo *type* em *TemporalAttribute*), e então chamar o construtor correspondente passando o valor no formato do tipo do atributo.
- *closeLabels* - finaliza os tempos de validade finais associados aos valores do histórico de uma propriedade, no caso de um objeto ser excluído logicamente (de acordo com as regras da seção 2.1.1).
- *getCurrent* - retorna o valor do relacionamento *has_as_current*.
- *getHistory* - retorna todos os valores do histórico com os rótulos de tempo.
- *getHistoryOfValue* - retorna o histórico dos rótulos temporais do valor passado por parâmetro.
- *getValueAt* - retorna o valor do histórico que é válido em um instante de tempo específico passado por parâmetro.

Em relação ao TVM, nas classes *TemporalAttribute* e *TemporalRelationship* do TV_ODMG foram adicionadas novas versões dos construtores e das operações que no TVM possuem como parâmetro um tipo objeto para passagem do valor da propriedade, para que esses possam receber pelos parâmetros os tipos literais também. Além disso, nesta extensão, os construtores sempre recebem o tempo de validade inicial, pois se não for definido um tempo pelo usuário, então esse será igual ao tempo de validade inicial de vida do objeto. Na operação *updateValue*, foi excluído o parâmetro que passava o instante que devia ser alterado, pois isso é determinado pelos tempos de validade

passados, como também foi criada uma nova versão sem os tempos de validade no caso desses não serem informados, ou seja, o tempo de validade inicial é o momento atual.

```
class TemporalRelationship
   (extent TemporalRelationships)
{
relationship set<InstantRelationship> contains
     inverse InstantRelationship belongs;
relationship InstantRelationship has_as_current   //valor corrente do relacionamento
     inverse InstantRelationship is_current_of;

Object TemporalRelationship (in Object val,  in timestamp iValidTime);
Object TemporalRelationship (in Object val,  in timestamp iValidTime,
      in timestamp fValidTime);
Object TemporalRelationship (in set<Object> val,  in timestamp iValidTime);
Object TemporalRelationship (in set<Object> val,  in timestamp iValidTime,
      in timestamp fValidTime);
Object TemporalRelationship (in list<Object> val,  in timestamp iValidTime);
Object TemporalRelationship (in list <Object> val,  in timestamp iValidTime,
      in timestamp fValidTime);
Object TemporalRelationship (in bag<Object> val,  in timestamp iValidTime);
Object TemporalRelationship (in bag<Object> val,  in timestamp iValidTime,
      in timestamp fValidTime);

InstantRelationship getCurrent ( );
set<InstantRelationship> getHistory ( );
set<TemporalLabel> getHistoryOfValue (in Object val);
set<TemporalLabel> getHistoryOfValue (in set<Object> val);
set<TemporalLabel> getHistoryOfValue (in list<Object> val);
set<TemporalLabel> getHistoryOfValue (in bag<Object> val);
InstantRelationship getValueAt (in timestamp at);
void updateValue (in Object val);
void updateValue (in Object val, in timestamp iValidTime, in timestamp fValidTime);
void updateValue (in set<Object> val);
void updateValue (in set<Object>val, in timestamp iValidTime, in timestamp fValidTime);
void updateValue (in list<Object>val);
void updateValue (in list<Object>val, in timestamp iValidTime, in timestamp fValidTime);
void updateValue (in bag<Object>val);
void updateValue (in bag<Object>val, in timestamp iValidTime, in timestamp fValidTime);
void closeLabels ( );
};
```

A classe *TemporalAttribute* possui, além dos relacionamentos e das operações descritos anteriormente, um atributo para armazenar o tipo do atributo temporal e uma operação para retornar esse valor. Além disso, todas as operações dessa classe que servem para lidar com literais atômicos recebem esses valores no formato de *string*, como mencionado anteriormente no caso da operação *updateValue*.

```
class TemporalAttribute
   (extent TemporalAttributes)
{
attribute string type;
relationship set<InstantAttribute> contains
     inverse InstantAttribute belongs;
relationship InstantAttribute has_as_current       // valor corrente do atributo
     inverse InstantAttribute is_current_of;

Object TemporalAttribute (in string type, in Object val, in timestamp iValidTime,
      in timestamp fValidTime);     //construtor para os tipos objetos
Object TemporalAttribute (in string type, in Object val, in timestamp iValidTime);
```

```
                                        //construtor para os tipos objetos
Object TemporalAttribute (in string type, in string val, in timestamp iValidTime,
     in timestamp fValidTime);     //construtor para os tipos literais atômicos
Object TemporalAttribute (in string type, in string val, in timestamp iValidTime);
                                        //construtor para os tipos literais atômicos
Object TemporalAttribute (in string type, in set<t> val, in timestamp iValidTime,
     in timestamp fValidTime);
Object TemporalAttribute (in string type, in set<t> val, in timestamp iValidTime);
...
Object TemporalAttribute (in string type, in date val, in timestamp iValidTime,
     in timestamp fValidTime);
Object TemporalAttribute (in string type, in date val, in timestamp iValidTime);
...
InstantAttribute getCurrent ( );
set<InstantAttribute> getHistory ( );
InstantAttribute getValueAt (in timestamp at);
set<TemporalLabel> getHistoryOfValue (in Object val);       // para tipos objetos
set<TemporalLabel> getHistoryOfValue (in string val);       //para tipos literais atômicos
set<TemporalLabel> getHistoryOfValue (in set<t> val);
set<TemporalLabel> getHistoryOfValue (in bag<t> val);
set<TemporalLabel> getHistoryOfValue (in list<t> val);
set<TemporalLabel> getHistoryOfValue (in array<t> val);
set<TemporalLabel> getHistoryOfValue (in dictionary<t,v> val);
set<TemporalLabel> getHistoryOfValue (in date val);
set<TemporalLabel> getHistoryOfValue (in time val);
set<TemporalLabel> getHistoryOfValue (in timestamp val);
set<TemporalLabel> getHistoryOfValue (in interval val);
string getType ( );
void updateValue (in Object val);    // para tipos objetos
void updateValue (in Object val, in timestamp iValidTime, in timestamp fValidTime);
                                                  // para tipos objetos
void updateValue (in string val);     //para tipos literais atômicos
void updateValue (in string val, in timestamp iValidTime, in timestamp fValidTime);
                                                        //para tipos literais atômicos
void updateValue (in set<t> val);
void updateValue (in set<t> val, in timestamp iValidTime, in timestamp fValidTime);
...
void updateValue (in date val);
void updateValue (in date val, in timestamp iValidTime, in timestamp fValidTime);
...
void closeLabels ( );
};
```

Em cada uma das classes *InstantRelationship* e *InstantAttribute* é definido um atributo do tipo *TemporalLabel* (para armazenar os tempos de validade e de transação dos valores das propriedades), uma operação para retornar os tempos e duas operações para definir os tempos de validade e de transação finais.

```
class InstantRelationship
   (extent InstantRelationships)
{
attribute TemporalLabel tempLabel;
relationship TemporalRelationship belongs
     inverse TemporalRelationship contains;
relationship TemporalRelationship is_current_of
     inverse TemporalRelationship has_as_current;

TemporalLabel getTempLabel ( );
```

```
void setFTransactionTime (in timestamp i);  //é chamada pelo updateValue
void setFValidTime (in timestamp i);        //é chamada pelo updateValue
};

class InstantAttribute
  (extent InstantAttributes)
{
attribute TemporalLabel tempLabel;
relationship TemporalAttribute belongs
    inverse TemporalAttribute contains;
relationship TemporalAttribute is_current_of
    inverse TemporalAttribute has_as_current;

TemporalLabel getTempLabel ( );
void setFTransactionTime (in timestamp i);  //é chamada pelo updateValue
void setFValidTime (in timestamp i);        //é chamada pelo updateValue
};
```

Cada uma das classes especializadas de *InstantAttribute* e de *InstantRelationship* especifica um atributo para armazenar o valor da propriedade temporal e uma operação para recuperar esse valor. Também define um construtor para receber o valor e a validade inicial e outro que recebe o valor, a validade inicial e a validade final, sendo que o uso de um ou de outro depende de a validade final do valor ter sido ou não especificada.

```
class HRObject extends InstantRelationship
   (extent HRObjects)
{
attribute Object value;
Object HObject (in Object val, in timestamp iValidTime, in timestamp fValidTime);
Object HObject (in Object val, in timestamp iValidTime);
Object getValue ( );
};

class HRSet extends InstantRelationship
   (extent HRSets)
{
attribute set<Object> value;
Object HRSet (in set<Object> val, in timestamp iValidTime, in timestamp fValidTime);
Object HRSet (in set<Object> val, in timestamp iValidTime);
Object getValue ( );
};

...

class HObject extends InstantAttribute
   (extent HObjects)
{
attribute Object value;
Object HObject (in Object val, in timestamp iValidTime, in timestamp fValidTime);
Object HObject (in Object val, in timestamp iValidTime);
Object getValue ( );
};

class Hlong extends InstantAttribute
   (extent Hlongs)
{
attribute long value;
```

```
Object Hlong (in string val, in timestamp iValidTime, in timestamp fValidTime);
Object Hlong (in string val, in timestamp iValidTime);
long getValue ( );
};

...

class HSet extends InstantRelationship
   (extent HSets)
{
attribute set<t> value;
Object HSet (in set<t> val, in timestamp iValidTime, in timestamp fValidTime);
Object HSet (in set<t> val, in timestamp iValidTime);
Object getValue ( );
};
...
```

Comparando ao TVM, pode-se notar que várias operações das classes descritas nessa seção usam o tipo *Object* no lugar em que era usado o tipo *OIDt* no TVM. Isso se deve ao fato de que no TVM todos os objetos possuem um *tvOID*, e no TV_ODMG, apenas os objetos temporais versionáveis é que possuem *tvOID*. Então, para manter uma uniformidade entre os objetos, todos são manipulados pelo OID fornecido pelo sistema.

4.2 TV_ODL

A TV_ODL caracteriza-se pela inclusão de algumas novas regras à ODL, assim como pela modificação de algumas regras existentes. A sintaxe completa da ODL pode ser encontrada em [CAT 2000].

A indicação das regras que foram modificadas aparece entre parênteses na forma (nx#) ou (n#). O "n" e o "x" correspondem à regra definida no ODMG que está sendo alterada. O "#" indica que houve alteração para suportar as classes temporais versionáveis. As regras que foram inseridas são representadas por (nTy), onde o "n" corresponde ao número da regra definida no ODMG que é tida como ponto de partida para a nova, o T determina que é para suportar as classes temporais versionáveis, e o "y" - o número da nova regra. As regras que não possuem # nem T são regras da ODL que aparecem neste texto, apenas, para melhorar o entendimento das outras. Termos em negrito são palavras reservadas da linguagem.

4.2.1 Especificação e Características do Tipo

Para especificação e definição das características de um tipo, algumas regras foram alteradas e outras introduzidas, a fim de possibilitar a declaração de classes temporais versionáveis.

Nas regras a seguir, o termo *hasVersion* indica que a classe é temporal versionável, o *byExtension* indica que a classe herda por extensão de outra classe temporal versionável, e o *correspondence* aponta a cardinalidade entre os objetos da classe com seus ascendentes na classe da qual herda. É possível definir também um *extent* para esse tipo de classe.

(2a#) <class>::= <class_dcl> | <class_forward_dcl> | <tv_class_dcl>

(2c#) <class_forward_dcl>::= **class** <identifier> |

class <tv_class_name> **hasVersion**

(2T1) <tv_class_dcl>::= <tv_class_header >
{<tv_interface_body >**}**

(2T2) <tv_class_header>::= **class** <tv_class_name> **hasVersion**
[**byExtension <**tv_ascendant>]
[**(** [<extent_spec>] **)**]

(2T3) <tv_class_name>::= <identifier>

(2T4) **<**tv_ascendant> ::= <tv_class_name> [**correspondence** <corresp_options>]

(2T5) <corresp_options>::= **1:1** | **1:n** | **n:1** | **n:m**

4.2.2 Propriedades do Tipo

Foram acrescentadas regras para suportar as propriedades das classes temporais versionáveis, as quais podem ser propriedades temporais ou não-temporais.

(8T1) <tv_interface_body >::= <tv_export> |
<tv_export> <tv_interface_body>

(9T1) <tv_export>::= <temp_attr_dcl>**;** | <temp_rel_dcl>**;** | <aggr_dcl>**;**
| <temp_aggr_dcl>**;** | <export>

(9*) <export>::= <type_dcl>**;** | <const_dcl>**;** | <except_dcl>**;** | <attr_dcl>**;** |
<rel_dcl>**;** | <op_dcl>**;**

Os atributos temporais e as declarações de valores *default* para atributos devem ser definidos de acordo com as seguintes regras:

(65T1) <temp_attr_dcl >::= [**readonly**] **temporal_attribute** <temp_domain_type>
<attr_list>

(65T2) <temp_domain_type>::= <simple_type_spec>

(30) <simple_type_spec> ::= <base_type_spec> | <template_type_spec>
| <scoped_name>

(31*) <base_type_spec> ::= <floating_pt_type> | <integer_type> | <char_type>
| <boolean_type> | <octet_type> | <date_type>
| <time_type> | <interval_type> | <timestamp_type>

(32*) <template_type_spec> ::= <array_type> | <string_type> | <coll_type>

(11) <scoped_name> ::= <identifier> | :: <identifier>| <scoped_name> :: <identifier>

(65a#) <attr_list>::= <attribute_name> [<fixed_array_size>] [<default_dcl>]
[,<attr_list>]

(65T3) <default_dcl> ::= **default** <literal>

Os relacionamentos de associação temporais devem ser definidos de acordo com as regras:

(65T4) <temp_rel_dcl >::= **temporal_relationship** <target_of_path> <identifier>
inverse <inverse_traversal_path>

(65e) <target_of_path>::= <identifier> | <coll_spec> **<** <identifier> **>**

(65f) <inverse_traversal_path>::= <identifier>::<identifier>

E por fim, os relacionamentos de agregação devem ser declarados de acordo com as seguintes regras:

(65T6) <aggr_dcl>::= **aggregateOf** <target_of_path> <identifier>
inverse <inverse_traversal_path>
| **componentOf** <target_of_path> <identifier>
inverse <inverse_traversal_path>

(65T7) <temp_aggr_dcl> ::= **temporal_aggregateOf** <target_of_path>
<identifier> **inverse** <inverse_traversal_path>
| **temporal_componentOf** <target_of_path> <identifier>
inverse <inverse_traversal_path>

Neste capítulo, todas as definições de classes mostradas utilizam a sintaxe descrita nesta seção.

4.3 TV_OQL

A TV_OQL estende a OQL com características que possibilitam a manipulação de tempo e versões, baseando-se nas definições feitas em [MOR 2001b]. Esta seção apresenta as características da TV_OQL, bem como sua sintaxe.

4.3.1 Manipulação de Tempo

Algumas expressões de tempo são especificadas para serem aplicadas sobre as propriedades temporais, devendo retornar os tempos de validade ou transação definidos para os valores das mesmas. Esses tempos são representados por literais do tipo *timestamp*. A seguir, são apresentadas tais expressões pré-definidas:

- *tPeriod* - período de tempo de transação, constituído por um tempo de transação inicial e um tempo de transação final, devendo o inicial ser menor que o final;
- *vPeriod* - período de tempo de validade, constituído por um tempo de validade inicial e um tempo de validade final, devendo o inicial ser menor que o final;
- *tiInstant* - tempo de transação inicial;
- *tfInstant* - tempo de transação final;
- *viInstant* - tempo de validade inicial;
- *vfInstant* - tempo de validade final.

Existem também expressões pré-definidas para lidar com o tempo de vida dos objetos temporais versionáveis:

- *iLifetime* - tempo de vida inicial do objeto (referente à última vida do objeto, ou seja, à última vez que o *alive* possuiu valor *true*);
- *fLifetime* - tempo de vida final do objeto (referente à última vida do objeto).

Os períodos de transação e validade são constituídos de um instante inicial e de um final, mas não são compatíveis com os mesmos operadores suportados pelos instantes. Dessa forma, são usados operadores especiais que geram expressões booleanas. Esses operadores são descritos a seguir supondo a existência de dois períodos chamados *a* e *b*, e um instante *c*:

- *periodIntersect* - testa se *a* e *b* possuem algum ponto em comum;
- *overlap* - *a overlap b*, se todos os pontos de *b* estão contidos nos pontos de *a*;
- *equal* - se os dois períodos são iguais (começam e terminam em instantes iguais);
- *after*:
  - *a after b* - se todos os pontos de *a* estão depois de todos os pontos de *b*;
  - *c after b* - se o ponto *c* está depois de todos os pontos de *b*;
- *into*:
  - *a into b* - se todos os pontos de *a* estão contidos nos pontos de *b*;
  - *c into b* - se o ponto *c* está contido nos pontos de *b*;
- *before*:
  - *a before b* - se todos os pontos de *a* estão antes de todos os pontos de *b*;
  - *c before b* - se o ponto *c* está antes de todos os pontos de *b*.

Também podem ser definidos períodos de forma explícita. Supondo que *x* e *y* são ambos literais *timestamp*, ou *date*, ou *time*, então:

- [x..y] - define um período com um valor inicial e um final;
- [..y] - determina um período que começa no passado infinito e termina em *y*;
- [x ..] - especifica um período que inicia em *x* e termina no futuro infinito.

Foram acrescentados na TV_OQL, os termos *nowDate*, *nowTime* e *nowTimestamp*, que correspondem, respectivamente, à data do sistema, à hora do sistema, e à data e hora do sistema.

Nos próximos exemplos, supõe-se a existência da classe temporal versionável Empregado com o *extent* Empregados, com o atributo não-temporal nome do tipo *string* e o atributo temporal salário do tipo *float*. A consulta, a seguir, obtém o valor e o tempo de validade inicial do atual salário dos empregados. O retorno é um *bag<struct(salario:float, vTimei: timestamp)>*.

```
select e.salario, e.salario.viInstant
from Empregados e
```

Se, ao invés de *viInstant*, fosse usado *vPeriod*, o retorno da consulta seria um *bag<struct(salario:float, vTimei: timestamp, vTimef: timestamp)>*.

Para obter o momento em que um determinado empregado foi demitido, pode-se selecionar o tempo de vida final do mesmo, considerando que os empregados são excluídos logicamente. O retorno dessa consulta é um *bag<timestamp>*.

```
select fLifetime
from Empregados e
where e.nome= “João”
```

A próxima consulta obtém os nomes dos funcionários que recebem salário superior a R$ 2.000,00 desde 10/03/2001 até hoje, retornando um *bag<string>*.

```
select e.nome
from Empregados e
where e.salario > 2000.00 and
        e.salario.vPeriod overlap [timestamp’10-03-2001’..nowTimestamp]
```

4.3.2 Manipulação de Versões

Para lidar com versões, foram definidos o termo *versions*, as expressões de versões, as expressões de navegação nas hierarquias e as expressões de acesso ao controle do objeto versionado.

O termo *versions* é aplicado sobre um conjunto de objetos temporais versionáveis, indicando que é para analisar os objetos temporais versionáveis que ainda não possuem versões, como também todas as versões dos objetos versionados, excluindo da consulta apenas os objetos versionados. É importante lembrar que o objeto versionado contém os valores da versão corrente, por isso não deve entrar na consulta para não haver duplicidade de informações.

Para os próximos exemplos, é considerada a existência de uma classe temporal versionável Automóvel com *extent* Automóveis e com o atributo motor do tipo *string*. A consulta, a seguir, obtém o motor de todas as versões e objetos temporais versionáveis que ainda não possuem versões da classe Automóvel (exclui apenas os objetos versionados). O retorno dessa consulta é um *bag<string>*.

```
select x.motor
from Automoveis.versions x
```

Sem o *versions*, a consulta retorna os objetos temporais versionáveis que ainda não possuem versões e as versões corrente dos objetos versionados, excluindo da consulta os objetos versionados e suas versões que não são corrente.

As expressões de versões são expressões booleanas que testam características que envolvem versões. São definidas por:

- *isFirst* - retorna *true*, se é a primeira versão de um objeto versionado;
- *isFirstAt*(*instant*) - retorna *true*, se é a primeira versão de um objeto versionado no momento *instant*;
- *isLast* - retorna *true*, se é a última versão de um objeto versionado;
- *isLastAt*(*instant*) - retorna *true*, se é a última versão de um objeto versionado no momento *instant*;
- *isCurrent* - retorna *true*, se é a versão corrente de algum objeto versionado;
- *isCurrentAt*(*instant*) - retorna *true*, se é a versão corrente de algum objeto versionado no momento *instant*;

- *isUserCurrent* - retorna *true*, se é a versão corrente de algum objeto versionado e se foi definida pelo usuário;
- *isUserCurrentAt*(*instant*) - retorna *true*, se no momento *instant* a versão é a corrente de algum objeto versionado e se naquele instante consta como definida pelo usuário;
- *isConfiguration* - retorna *true*, se é uma versão configurada.

Há também a função *entity* que é aplicada sobre objetos temporais versionáveis e representa o nome de uma entidade, o qual está armazenado na classe *TVEntity*. Considerando a consulta anterior, supõe-se que exista um objeto versionado na classe Automóvel representando o carro Palio (ou entidade Palio), e três versões desse carro, EX, ELX e EL. Se for desejado o motor da versão corrente de apenas um determinado objeto versionado, neste caso do Palio, então deve-se especificar a entidade através do termo *entity*, como na consulta:

```
select x.motor
from Automoveis.versions x
where x.isCurrent and x.entity= "Palio"
```

As expressões de navegação na hierarquia de extensão acessam os ascendentes e descendentes de um objeto na hierarquia. Essas podem ser aplicadas sobre todos os objetos temporais versionáveis. São elas:

- *ascendant* - retorna uma coleção do tipo *set* com o(s) ascendente(s). Dependendo da correspondência definida para a herança por extensão, o conjunto de ascendentes pode conter um ou mais elementos.
- *descendant* - retorna uma coleção do tipo *set* com o(s) descendente(s). Dependendo da correspondência definida para a herança por extensão, o conjunto de descendentes pode conter um ou mais elementos.
- *isAscendantOf*(*tvObject*) - retorna *true*, se é ascendente da versão *tvObject*.
- *isDescendantOf*(*tvObject*) - retorna *true*, se é descendente da versão *tvObject*.

As expressões de navegação na hierarquia de derivação são aplicadas sobre versões para acessar seus sucessores e predecessores:

- *predecessor* - retorna uma coleção do tipo *set* contendo o conjunto de predecessores do objeto;
- *successor* - retorna uma coleção do tipo *set* contendo o conjunto de sucessores do objeto;
- *isSuccessorOf*(*tvObject*) - retorna *true*, se é a sucessora da versão *tvObject*;
- *isSuccessorOfAt*(*tvObject*, *instant*) - retorna *true* se é sucessora da versão *tvObject* no momento *instant*;
- *isPredecessorOf*(*tvObject*) - retorna *true*, se é predecessora da versão *tvObject*.

As expressões de acesso ao controle do objeto versionado são aplicadas sobre versões ou objetos versionados, permitindo o acesso ao controle do objeto versionado correspondente de forma transparente. São elas:

- *configurationCount* - retorna o número de versões configuradas do objeto versionado;
- *currentVersion* - retorna a versão corrente do objeto versionado;
- *firstVersion* - retorna a primeira versão do objeto versionado;
- *lastVersion* - retorna a última versão do objeto versionado;
- *nextVersionNumber* - retorna o número da próxima versão do objeto versionado;
- *userCurrentFlag* - retorna o valor booleano, definido para o *flag* de versão corrente especificada pelo usuário;
- *versionCount* - retorna o número de versões do objeto versionado.

A próxima consulta obtém o conjunto de versões que deram origem à versão corrente do carro Palio.

```
select x.predecessor
from Automoveis.versions x
where x.isCurrent and x.entity= “Palio”
```

A consulta seguinte retorna se a versão corrente do carro Palio foi definida pelo usuário (se retornar *true*) ou se foi definida pelo sistema (se retornar *false*).

```
select x.userCurrentFlag
from Automoveis.versions x
where x.isCurrent and x.entity= “Palio”
```

4.3.3 Manipulação de Estado

As expressões de estado são expressões booleanas aplicadas sobre um objeto temporal versionável para testar seu estado. São representadas por:

- *isWorking* - retorna *true* se apresenta estado *working*;
- *isWorkingAt*(*instant*) - retorna *true* se apresenta estado *working* no momento *instant*;
- *isStable* - retorna *true* se apresenta estado *stable*;
- *isStableAt*(*instant*) - retorna *true* se apresenta estado *working* no momento *instant*;
- *isConsolidated* - retorna *true* se apresenta estado *consolidated*;
- *isConsolidatedAt*(*instant*) - retorna *true* se apresenta estado *consolidated* no momento *instant*;
- *isDeactivated* - retorna *true* se apresenta estado *deactivated*;
- *isDeactivatedAt*(*instant*) - retorna *true* se apresenta estado *deactivated* no momento *instant*.

Por exemplo, a próxima consulta obtém o motor das versões no estado *stable*:

```
select x.motor
from Automoveis.versions x
where x.entity= “Palio” and x.isStable
```

4.3.4 Recuperação do Histórico

A palavra reservada *ever* possibilita lidar com todos os valores do histórico de uma propriedade temporal, podendo ser especificada nas cláusulas *select* ou *where*.

Se o *ever* for especificado no *select*, a consulta retorna o histórico das propriedades temporais definidas no *select* e considera o histórico dos valores das propriedades temporais na cláusula *where*. Porém, se o *ever* for declarado no *where*, é considerado o histórico dos valores das propriedades temporais nessa cláusula e retornado os valores atuais das propriedades temporais definidas no *select*. É importante notar que o resultado da consulta com *ever* depende das propriedades serem temporais ou não.

Após o termo *ever*, são considerados os históricos de todas as propriedades temporais. Se for desejado o estado atual de apenas algumas dessas propriedades, pode-se usar a função *present* associada a cada uma delas. O *present* é aplicado sobre expressões com propriedades temporais e pode ser usado tanto no *select* quanto no *where*, caso tenha sido especificado o *ever* para essas cláusulas.

Supõe-se uma classe Empregado com o *extent* Empregados, com o atributo não-temporal nome do tipo *string* e com os atributos temporais salário do tipo *float* e cargo do tipo *string*. A próxima consulta obtém o histórico do salário do empregado que, atualmente, possui o cargo de gerente. Essa consulta retorna um *bag<bag<float>>*.

```
select ever e.salario
from Empregados e
where present(e.cargo= “gerente”)
```

A consulta seguinte retorna o salário atual dos funcionários que alguma vez foram gerente e que ganham mais que R$2.000,00, atualmente. A consulta retorna um *bag<float>*.

```
select e.salario
from Empregados e
where ever e.cargo= “gerente” and present(e.salario > 2000.00)
```

A próxima consulta obtém o nome, o histórico de salários, os tempos de validade iniciais desses e o histórico de cargos de um empregado que atualmente possui salário maior que R$ 2.000,00. O retorno é um *bag<struct(nome: string, salario: bag<struct(value: float, vTimei: timestamp)>, cargo: bag<string>)>*.

```
select ever nome, salario, salario.vInstant, cargo
from Empregados e
where present(salario > 2000.00)
```

Outros fatores também devem ser consideradas quanto ao uso do *ever*:

- Com ou sem o *ever* são analisados tanto os objetos ativos como os desativados (que foram excluídos logicamente). Deve ser especificado explicitamente a exclusão dos objetos desativados na consulta.
- O termo *ever* presente em um *select* não é considerado em outro *select* aninhado.
- O *ever* é aplicado apenas sobre propriedades temporais. Por exemplo, na consulta "*select ever* p *from* Pessoas p", esse termo não tem efeito algum, pois *p* representa um objeto e não uma propriedade.
- As expressões pré-definidas (de tempo, de versões e de estado) especificadas pela TV_OQL sofrem efeito do *ever* (com exceção das que passam como parâmetro um instante específico), pois lidam com propriedades temporais, embora isso seja transparente para o usuário.
- Em uma expressão de caminho, o *ever* é aplicado apenas sobre o último termo especificado no caminho, sendo que só terá algum efeito se esse termo for uma propriedade temporal ou uma das expressões pré-definidas como mencionado no item anterior.

4.3.5 Sintaxe da TV_OQL

Esta seção apresenta as alterações e inclusões realizadas na OQL, cuja sintaxe completa é definida em [CAT 2000].

A seguir são apresentadas as novas regras, como também as modificadas, as quais estão precedidas por (*).

As primeiras regras a serem alteradas são a do s*elect* e a do w*here* para a inclusão da palavra reservada *ever*.

(*) selectExpr ::= **select** [**ever**] [**distinct**] projectionAttributes
fromClause
[whereClause]
[groupClause]
[orderClause]

(*) whereClause ::= **where** [**ever**] expr

A regra *projectionAttributes*, da OQL, considera a definição de uma expressão de acordo com a regra *expr*, por meio da qual se chega à definição da regra *postfixExpr*, a qual foi estendida. Além das expressões definidas na OQL, a TV_OQL permite definir expressões de tempo, de acesso ao controle do objeto versionado, de navegação na hierarquia de extensão, de navegação na hierarquia de derivação, de versões e de estado. A seguir é descrita a sintaxe para essas expressões, para o uso do termo *versions*, para definição de uma entidade e para o uso do termo *present* que anula o efeito do *ever*:

(*) postfixExpr ::= primaryExpr {**[**index**]**}
| primaryExpr {(**.** | →) identifier [argList]} [(**.** | →) (tvExpr | **versions**)]

tvExpr ::= tempExpr | stateExpr | versionExpr | vocExpr | navegationExpr | **entity**

(*) primaryExpr ::= conversionExpr | collectionExpr | aggreagateExpr | undefinedExpr |
objectConstruction | structConstruction | collectionConstruction |

identifier [argList] | queryParam | literal | **(**query**)** | preDefLifetime |
stateExpr | versionExpr | vocExpr | navegationExpr | **entity** |
presentExpr

tempExpr ::= preDefInstant | preDefLifetime | preDefPeriod

preDefInstant ::= **tiInstant** | **tfInstant** | **viInstant** | **vfInstant**

preDefLifetime ::= **iLifetime** | **fLifetime**

preDefPeriod ::= **tPeriod** | **vPeriod**

stateExpr ::= **isWorking** | **isWorkingAt(**instant**)** | **isStable** | **isStableAt(**instant**)** |
isConsolidated | **isConsolidatedAt(**instant**)** | **isDeactivated** |
isDeactivatedAt(instant**)**

versionExpr ::= **isFirst** | **isFirstAt(**instant**)** | **isLast** | **isLastAt(**instant**)** |
isCurrent | **isCurrentAt(**instant**)** | **isUserCurrent** |
isUserCurrentAt(instant**)** | **isConfiguration**

vocExpr ::= **configurationCount** | **currentVersion** | **firstVersion** | **lastVersion** |
nextVersionNumber | **userCurrentFlag** | **versionCount**

navegationExpr ::= **ascendant** | **descendant** | **predecessor** | **successor |**
isSuccessorOf(tvObject**)** | **isSuccessorOfAt(**tvObject, instant**)** |
isPredecessorOf(tvObject**)** | **isAscendantOf(**tvObject**)** |
isDescendantOf(tvObject**)**

instant ::= primaryExpr {(**.** | →) identifier [argList]} (**.** | →) (preDefInstant | preDefLifetime)
| timestampLiteral | **nowTimestamp**

tvObject ::= identifier

presentExpr ::= **present (**expr**)**

A regra *andthenExpr* da OQL foi estendida para contemplar a consulta com períodos e instantes temporais. Essa regra pode ser alcançada pela regra *expr*.

(*) andthenExpr ::= equalityExpr {**andthen** equalityExpr} |
periodExpr {**andthen** periodExpr}

periodExpr ::= tPeriodExpr (**periodIntersect** | **overlap** | **equal**) tPeriodExpr
| tMixExpr (**after** | **into** | **before**) tPeriodExpr

tPeriodExpr::= primaryExpr{(**.** | →) identifier [argList]} (**.** | →) preDefinedPeriod
| definedPeriod

definedPeriod ::= **[**iInstant**..**[fInstant]**]** | **[..**fInstant**]**

iInstant ::= tInstantExpr

fInstant ::= tInstantExpr

tInstantExpr ::= primaryExpr {(**.** | →) identifier [argList]}
(**.** | →) (preDefInstant | preDefLifetime)
| definedInstant

definedInstant ::= timestampLiteral | dateLiteral | timeLiteral | **nowDate | nowTime**
| nowTimestamp

tMixExpr ::= tPeriodExpr | tInstantExpr

Os termos correspondentes à data e/ou à hora do sistema são adicionados à regra *literal*.

(*) literal ::= booleanLiteral | longLiteral | doubleLiteral | charLiteral | stringLiteral |
dateLiteral | timeLiteral | timestampLiteral | **nil** | **undefined** | **nowDate**
| nowTime | nowTimestamp

Mais exemplos com o uso dessa linguagem são apresentados no estudo de caso.

4.4 Considerações Finais

Neste capítulo, foi apresentado o TV_ODMG que é a extensão proposta do padrão ODMG com suporte para tempo e versões. Para fornecer esse suporte, essa extensão baseou-se nas características definidas pelo Modelo Temporal de Versões. Foi apresentado como as versões e a temporalidade podem ser integradas ao Modelo de Objetos do ODMG apenas adicionando características, sem alterar sua forma inicial para que a extensão não se torne incompatível com aplicações existentes. Também foi exibida a sintaxe para definição de classes com suporte a tempo e versões e a linguagem de consulta que permite lidar com a extensão proposta.

Foi mostrado que algumas características do TVM ao serem adaptadas ao ODMG foram modificadas devido a peculiaridades do padrão ou a limitações do próprio TVM.

5 Integração com o Ambiente ODMG

Para usar o TV_ODMG em um ODBMS com suporte ao ODMG, sem precisar de um ODBMS específico com suporte a essa extensão, deve ser feito um mapeamento das características acrescentadas ao padrão. A estrutura necessária para isso é composta de:

- processador de TV_ODL - faz a análise dos arquivos que entram com a definição do esquema para verificar se estão de acordo com a especificação da TV_ODL, e então, mapea para ODL, de acordo com as regras descritas na seção 5.2, para poder ser processado pelo ambiente ODMG;
- processador de TV_OQL - faz a análise dos arquivos que entram com a definição das consultas para verificar se estão de acordo com a especificação da TV_OQL, e então, mapea para OQL, de acordo com as regras descritas na seção 5.3, para poder ser processado pelo ambiente ODMG.

Essa integração com o ambiente ODMG é ilustrada na figura 5.1. A biblioteca de classes mostrada contém as definições das classes correspondentes aos tipos acrescentados ao ODMG, e o ambiente ODMG é composto por um ODBMS que suporta a ODL abstrata como, por exemplo, o Orient ODBMS [ORI 2001].

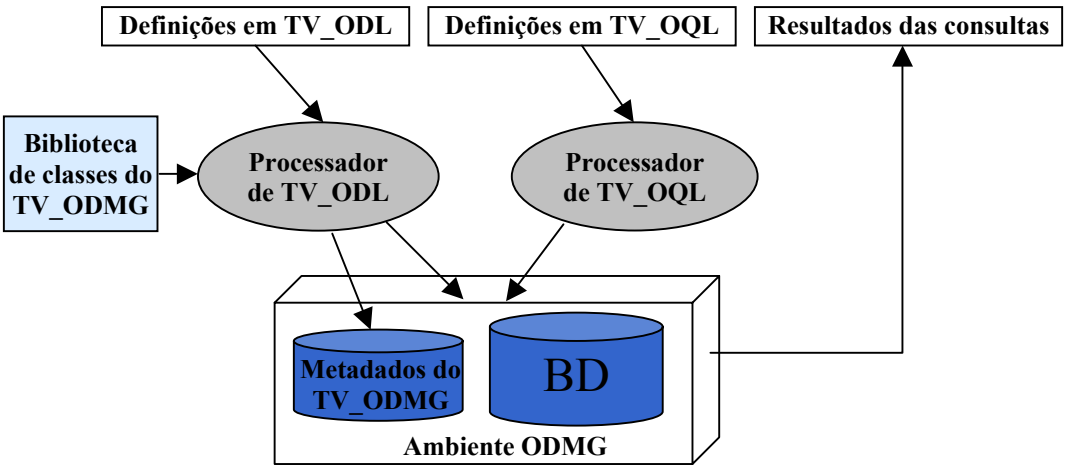

FIGURA 5.1 - Integração com o ambiente ODMG

Além disso, devem ser definidas *triggers* para o sistema criar, na instanciação de um objeto, instâncias de *TemporalRelationship* ou *TemporalAttribute* para que através dessas seja armazenado o primeiro valor do histórico de cada uma das propriedades temporais do objeto, como também, para o sistema criar instâncias das classes especializadas de *InstantRelationship* ou de *InstantAttribute* quando os valores dessas propriedades forem atualizados para que os mesmos sejam também armazenados no histórico.

5.1 Metadados

Uma estrutura complementar aos metadados do ODMG é definida para armazenar as características específicas do TV_ODMG. Essa estrutura não altera a pré-existente. A figura 5.2 ilustra esse complemento, no qual são definidas:

- classe *tv_class* - armazena algumas informações sobre as classes que são temporais versionáveis: o nome da classe, se essa é classe raiz na hierarquia

por extensão, o *id* da classe (número usado para formar o *tvOID* dos objetos temporais versionáveis);

- classe *tv_libClass* - armazena o nome das classes básicas da biblioteca de classes (*TemporalObject*, *TemporalVersion* e *VersionedObjectControl*);
- interface *tv_relationship* - define as operações que podem ser usadas pelas classes que herdam dela;
- classe *tv_attribute* - armazena informações sobre os atributos que foram definidos com um valor *default* e/ou que são temporais;
- classe *tv_temp_association* - armazena informações sobre os relacionamentos de associação que são temporais;
- classe *tv_aggregation* - armazena informações sobre os relacionamentos de agregação, como o nome da agregação, se é temporal, e se a classe representa o todo ou um componente;
- classe *tv_ascClass* - armazena informações sobre a classe da qual uma determinada classe herda por extensão;
- classe *tv_descClass* - armazena informações sobre as classes que herdam por extensão de uma determinada classe.

Cada uma dessas classes também armazenam os relacionamentos que as associam as outras classes.

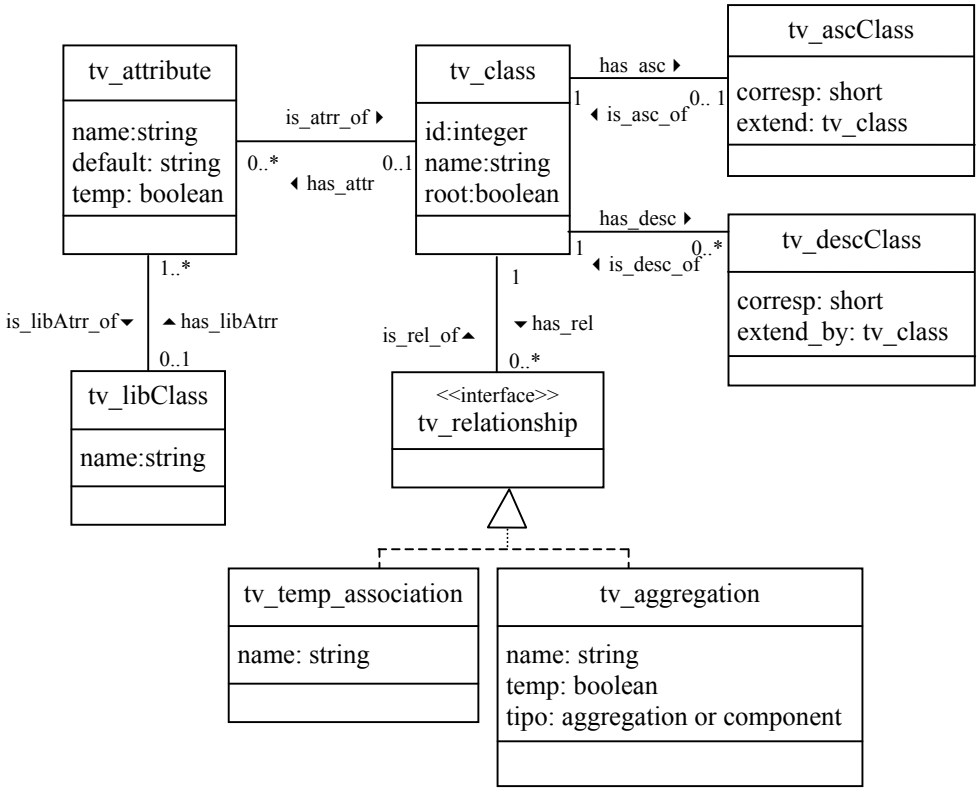

FIGURA 5.2 - Metadados do TV_ODMG

Essa estrutura apresentada deve ser previamente definida no ODBMS para que os processadores de TV_ODL e TV_OQL sejam usados.

5.2 Regras para o Processador de TV_ODL

O processador de TV_ODL tem como entrada um arquivo em TV_ODL e como saída um arquivo em ODL. Algumas regras são estabelecidas para que o processador faça esse mapeamento sem que as características do TV_ODMG sejam perdidas. A entrada pode ser um arquivo contendo a biblioteca de classes ou um arquivo fornecido pelo usuário. No primeiro caso, as únicas regras a serem aplicadas são as seguintes:

- Análise léxica, sintática e semântica da TV_ODL das classes básicas da biblioteca.
- As classes *TemporalObject*, *TemporalVersion* e *VersionedObjectControl* devem gerar instâncias nos metadados (na classe *tv_libClass*).
- Quando houver uma declaração com "**temporal_attribute**", deve ser armazenado nos metadados (na classe *tv_attribute*) que aquele atributo é temporal (indicando que seu histórico de valores será armazenado), a cláusula deve ser substituída por "**attribute**" e deve ser acrescentado um atributo do tipo *TemporalAttribute* chamado "temp_*nomeAtributo*", onde *nomeAtributo* é o nome do atributo declarado como temporal. Esse novo atributo serve para referenciar a instância correspondente a seu histórico em *TemporalAttribute*. Exemplo:

TV_ODL	ODL
... **temporal_attribute float** salario; ...	... **attribute float** salario; **attribute TemporalAttribute** temp_salario; ...

- O termo "**default**" indica que o valor declarado como *default* para uma determinada propriedade deve ser armazenado nos metadados (na classe *tv_attribute*). Após mapeado, esse termo não deve mais aparecer na definição da propriedade. Exemplo:

TV_ODL	ODL
... **attribute string** estado **default** 'RS'; ...	... **attribute string** estado; ...

No caso de um arquivo fornecido pelo usuário, há algumas regras de integridade e outras de mapeamento a serem seguidas. As regras de integridade são:

- O arquivo em TV_ODL fornecido pelo usuário não deve conter declarações de herança por refinamento ou por extensão das classes definidas na biblioteca de classes do TV_ODMG.
- Deve ser retornado um erro se o usuário definir, dentro de uma mesma classe, propriedades com nomes "temp_*nomeAtributo*" e "temp_*nomeRel*", onde *nomeAtributo* e *nomeRel* sejam os nomes de propriedades declaradas como temporais. Exemplo:

TV_ODL

```
...
temporal_attribute float salario;
attribute float temp_salario;
...
```

As regras de mapeamento são:

- Análise léxica, sintática e semântica da TV_ODL.
- Devem ser armazenadas informações nos metadados (na classe *tv_class*) sobre cada classe que possuir a cláusula "**hasVersion**" e depois substituir o termo por "**extends** TemporalVersion". Exemplo:

TV_ODL	ODL
class Veiculo **hasVersion** {...};	**class** Veiculo **extends** TemporalVersion {...};

- As cláusulas "**byExtension**" e "**correspondence**" implicam armazenar nos metadados (classes *tv_class* e *tv_ascClass*) que determinada classe herda por extensão de outra e a correspondência definida para essa herança; para a outra classe também deve ser armazenado (classes *tv_class* e *tv_descClass*) que essa está sendo estendida e a respectiva correspondência. Após isso, as declarações do "**byExtension**" e "**correspondence**" não devem mais aparecer na definição da classe, ou seja, devem ser apagadas após o mapeamento da classe. Uma *trigger* deve permitir ou não, de acordo com a correspondência definida na herança, que seja inserido mais de um elemento nos conjuntos definidos pelas propriedades *ascendant* e *descendant* da classe *TemporalVersion*. Exemplo:

TV_ODL	ODL

class Automovel **hasVersion**
  byExtension Veiculo
  correspondence n:m
{...};
	class Automovel **extends** TemporalVersion {...};

- Quando houver uma declaração com "**temporal_attribute**", deve ser armazenado nos metadados (na classe *tv_attribute*) que aquele atributo é temporal (indicando que seu histórico de valores será armazenado), substituída a cláusula por "**attribute**" e acrescentado um atributo do tipo *TemporalAttribute* chamado "temp_*nomeAtributo*", onde *nomeAtributo* é o nome do atributo declarado como temporal. Esse novo atributo serve para referenciar a instância que corresponde a seu histórico em *TemporalAttribute*. Assim, na atribuição do primeiro valor à propriedade temporal, esse novo atributo gerado deve receber a respectiva referência.
- Quando houver uma declaração com "**temporal_relationship**", deve ser armazenado nos metadados (na classe *tv_temp_association*) que aquele relacionamento é temporal, substituída a cláusula por "**relationship**" e acrescentado um atributo do tipo *TemporalRelationship* chamado "temp_*nomeRel*", onde *nomeRel* é o nome do relacionamento declarado como temporal. Esse novo atributo referencia a instância que corresponde a seu histórico em *TemporalRelationship*. Dessa forma, na atribuição do primeiro

valor à propriedade temporal, esse novo atributo gerado deve receber a respectiva referência. Exemplo:

TV_ODL	ODL
... **temporal_relationship set**<Emp> emprega **inverse** Emp::trabalha_em; ...	... **relationship set**<Emp> emprega **inverse** Emp::trabalha_em; **attribute TemporalRelationship** temp_emprega; ...

- Para as declarações "**aggregateOf**", "**temporal_aggregateOf**", "**componentOf**" e "**temporal_componentOf**" deve-se armazenar nos metadados (na classe *tv_aggregation*) que o relacionamento é de agregação e se é temporal, como também informar se representa a agregação ou o componente. Após, deve-se substituir o termo por "**relationship**". Se for temporal, deve-se acrescentar um atributo do tipo *TemporalRelationship* chamado "temp_*nomeRel*", onde *nomeRel* é o nome do relacionamento. Exemplo:

TV_ODL	ODL
... **agregateOf set**<ItemPedido> contem **inverse** ItemPedido::pertence; ... **componentOf** Pedido pertence **inverse** Pedido::contem; ...	... **relationship set**<ItemPedido> contem **inverse** ItemPedido::pertence; ... **relationship** Pedido pertence **inverse** Pedido::contem; ...

- O termo "**default**" indica que o valor, declarado como *default* para uma determinada propriedade, deve ser armazenado nos metadados (na classe *tv_attribute*). Após isso, a declaração deve ser apagada. Porém, se o usuário definir valores *default*, esses só serão realmente usados se o usuário implementar sua aplicação consultando esses valores nos metadados.

5.3 Regras para o Processador de TV_OQL

O processador de TV_OQL possui como entrada um arquivo em TV_OQL e como saída um arquivo em OQL. As regras para este mapeamento são descritas a seguir:

- Análise léxica, sintática e semântica da TV_OQL.
- Os termos "**nowDate**", "**nowTime**" e "**nowTimestamp**" devem ser substituídos, respectivamente, por um literal do tipo *date* com a data do sistema, *time* com a hora do sistema e *timestamp* com a data e hora do sistema.
- Períodos definidos - se apresentar o valor infinito para valor inicial ou final do período, esse deve ser substituído por um literal de valor muito pequeno e muito grande, respectivamente, dos tipos *time*, *date* ou *timestamp*, de acordo com o outro valor do período definido. Exemplo:

TV_OQL	OQL
select nome **from** Empregados **where** admissao **into [**..**date**'20-03-2002'**]**	**select** nome **from** Empregados **where** admissao **into** **[date**'01-01-0001'..**date**'20-03-2002'**]**

- **iLifetime** - substituir pelo método "getLifetimeI".
- **fLifetime** - substituir pelo método "getLifetimeF".
- **tPeriod** - quando aparece na cláusula *select*, deve-se tirar o termo e a expressão de caminho que o antecede, e substituir por "**tiInstant**, **tfInstant**". A expressão de caminho que o antecedia deve preceder também *tiInstant* e *tfInstant*. Na cláusula *where*, esse termo aparece, apenas, quando está envolvido nas regras de operadores de períodos, descritas adiante. Exemplo:

TV_OQL	OQL
select e.salario.**tPeriod** **from** Empregados e	**select** e.salario.**tiInstant**, e.salario.**tfInstant** **from** Empregados e

- **vPeriod** - quando aparece na cláusula *select*, deve-se tirar o termo e a expressão de caminho que o antecede, e substituir por "**viInstant**, **vfInstant**". A expressão de caminho que o antecedia deve preceder também *viInstant* e *vfInstant*. Na cláusula *where*, esse termo aparece, apenas, quando está envolvido nas regras de operadores de períodos, descritas a seguir. Exemplo:

TV_OQL	OQL
select e.salario.**vPeriod** **from** Empregados e	**select** e.salario.**viInstant**, e.salario.**vfInstant** **from** Empregados e

- **periodIntersect** - Supondo *a* e *b* dois períodos, "*a* **periodIntersect** *b*" deve ser substituído por "*aInstantI* <= *bInstantF* **and** *aInstantF* >= *bInstantI*", sendo *aInstantI* o instante inicial do período *a*, *aInstantF* o instante final de *a*, *bInstantI* o instante inicial do *b*, e *bInstantF* o instante final do *b*. Porém, se algum desses instantes finais for representado pelo tempo final de validade ou de transação do valor de uma propriedade, então o mapeamento deve ser da forma descrita logo abaixo, pois esses valores podem conter valor *nil*, indicando, no caso da validade, que o valor da propriedade é válido para sempre, e no caso da transação, que se acredita que as informações referentes aos outros tempos do valor da propriedade são válidas para sempre (como descrito na seção 2.1.1):
  - Se *bInstantF* representar um dos tempos - "(*aInstantI* <= *bInstantF* **or** *bInstantF* = **nil**) **and** *aInstantF* >= *bInstantI*".
  - Se *aInstantF* representar um dos tempos - "*aInstantI* <= *bInstantF* **and** (*aInstantF* >= *bInstantI* **or** *aInstantF* = **nil**)".
  - Se *bInstantF* e *aInstantF* representarem um dos tempos - "(*aInstantI* <= *bInstantF* **or** *bInstantF* = **nil**) **and** (*aInstantF* >= *bInstantI* **or** *aInstantF* = **nil**)".

- **overlap** - Supondo *a* e *b* dois períodos, "*a* **overlap** *b*" deve ser substituído por "*aInstantI* <= *bInstantI* **and** *aInstantF* >= *bInstantF*". Entretanto, se *aInstantF* for representado pelo tempo final de validade ou de transação do valor de uma propriedade, então deve ser mapeado para "*aInstantI* <= *bInstantI* **and** (*aInstantF* >= *bInstantF* **or** *aInstantF* = **nil**)".
- **equal** - Supondo *a* e *b* dois períodos, "*a* **equal** *b*" deve ser substituído por "*aInstantI* = *bInstantI* **and** *aInstantF* = *bInstantF*".
- **after** (com dois períodos) - Supondo *a* e *b* dois períodos, "*a* **after** *b*" deve ser substituído por "*aInstantI* > *bInstantF* **and** *aInstantF* > *bInstantF*". Porém, se *aInstantF* for representado pelo tempo final de validade ou de transação do valor de uma propriedade, então deve ser mapeado para "*aInstantI* > *bInstantF* **and** (*aInstantF* > *bInstantF* **or** *aInstantF* = **nil**)".
- **after** (com um instante e um período) - Supondo *x* um instante e *b* um período, "*x* **after** *b*" deve ser substituído por "*x* > *bInstantF*".
- **into** (com dois períodos) - Supondo *a* e *b* dois períodos, "*a* **into** *b*" deve ser substituído por "*aInstantI* > *bInstantI* **and** *aInstantF* < *bInstantF*". Porém, se *bInstantF* for representado pelo tempo final de validade ou de transação do valor de uma propriedade, então deve ser mapeado para "*aInstantI* > *bInstantI* **and** (*aInstantF* < *bInstantF* **or** *bInstantF* = **nil**)". E se *bInstantF* e *aInstantF* indicarem um dos tempos, então deve ser mapeado para "*aInstantI* > *bInstantI* **and** (*aInstantF* < *bInstantF* **or** (*bInstantF* = **nil and** *aInstantF* != **nil**))".
- **into** (com um instante e um período) - Supondo *x* um instante e *b* um período, "*x* **into** *b*" deve ser substituído por "*x* > *bInstantI* **and** *x* < *bInstantF*". Se *bInstantF* for representado pelo tempo final de validade ou de transação do valor de uma propriedade, então o mapeamento deve ser "*x* > *bInstantI* **and** (*x* < *bInstantF* **or** (*bInstantF* = **nil and** *aInstantF* != **nil**))".
- **before** (com dois períodos) - Supondo *a* e *b* dois períodos, "*a* **before** *b*" deve ser substituído por "*aInstantI* < *bInstantI* **and** *aInstantF* < *bInstantI*".
- **before** (com um instante e um período) - Supondo *x* um instante e *b* um período, "*x* **before** *b*" deve ser substituído por "*x* < *bInstantI*".
- Nas regras dos operadores *periodIntersect*, *overlap*, *equal*, *after*, *into* e *before* definidas acima, se *a* for um período determinado com "**tPeriod**", então *aInstantI* e *aInstantF* devem ser substituídos por "**tiInstant**" e "**tfInstant**", respectivamente; e a expressão de caminho que precedia o *tPeriod* deve anteceder esses termos também. Entretanto, se *a* for um período determinado com "**vPeriod**", então *aInstantI* e *aInstantF* devem ser substituídos por "**viInstant**" e "**vfInstant**", respectivamente; e a expressão de caminho que precedia o *vPeriod* deve anteceder esses termos também. Porém, se *a* for um período definido, deve-se substituir *aInstantI* e *aInstantF* pelos respectivos valores inicial e final. O mesmo vale para o período *b* em relação a *bInstantI* e *bInstantF*. Além disso, esses termos que representam instantes devem ser também mapeados de acordo com as regras de instantes descritas a seguir.
- **tiInstant** - deve-se substituir o termo e o nome da propriedade que o antecede por "temp_*propName*.has_as_current.tempLabel.tTimei", onde *propName* é o nome da propriedade em questão. Exemplo:

TV_OQL	OQL
select e.salario.**tiInstant** **from** Empregados e	**select** e.temp_salario.has_as_current.tempLabel.tTimei **from** Empregados e

- **viInstant** - deve-se substituir o termo e o nome da propriedade que o antecede por "temp_*propName*.has_as_current.tempLabel.vTimei", onde *propName* é o nome da propriedade em questão.
- **tfInstant** - deve-se substituir o termo e o nome da propriedade que o antecede por "temp_*propName*.has_as_current.tempLabel.tTimef", onde *propName* é o nome da propriedade em questão.
- **vfInstant** - deve-se substituir o termo e o nome da propriedade que o antecede por "temp_*propName*.has_as_current.tempLabel.vTimef", onde *propName* é o nome da propriedade em questão.
- O termo "**entity**" deve ser substituído por "tvOID.getEntityName".
- **versions** - para cada expressão de caminho com *versions* da consulta, deve-se acrescentar na cláusula *where* a condição "tvOID.getVersionNr!=0" antecedida da expressão de caminho que precedia o respectivo *versions* ou do *alias* (se houver) que representa o respectivo caminho. Após a análise de todos os *versions*, esses termos devem ser excluídos da consulta.
- Se não é especificada a expressão de caminho com "**versions**" para os conjuntos de objetos temporais versionáveis definidos no *from*, então deve-se acrescentar na cláusula *where*, para cada um desses conjuntos, a condição: "((tvOID.getVersionNumber=1 **and is_undefined**(e.belongs)) **or isCurrent**)", devendo essas expressões serem antecedidas por suas respectivas expressões de caminho.
- **isWorking** - este termo deve ser substituído por "status= 'w'".
- **isWorkingAt(***instant***)** - esse termo deve ser substituído por "getAttributeValueAt(status, instant).value='w'".
- **isStable** - esse termo deve ser substituído por "status = 's'".
- **isStableAt(***instant***)** - esse termo deve ser substituído por "getAttributeValueAt(status, instant).value= 's'".
- **isConsolidated** - esse termo deve ser substituído por "status= 'c'".
- **isConsolidatedAt**(*instant*) - esse termo deve ser substituído por "getAttributeValueAt(status, instant).value= 'c'".
- **isDeactivated** - esse termo deve ser substituído por "status= 'd'".
- **isDeactivatedAt**(*instant*) - esse termo deve ser substituído por "getAttributeValueAt(status, instant).value= 'd'".
- **isFirst** - esse termo e a expressão de caminho que o antecede devem ser substituídos por "same_as(belongs.firstVersion)". A expressão de caminho que precedia *isFirst* deve anteceder também o método *same_as* e a propriedade *belongs*.

- **isFirstAt(***instant***)** - substituir o termo e a expressão de caminho que o antecede por ”same_as(belongs.temp_firstVersion.getValueAt(instant). getValue)”. A expressão de caminho que precedia *isFirstAt* deve anteceder também o método *same_as* e a propriedade *belongs*.
- **isLast** - esse termo e a expressão de caminho que o antecede devem ser substituídos por “same_as(belongs.lastVersion)”. A expressão de caminho que precedia *isLast* deve anteceder também o método *same_as* e a propriedade *belongs*.
- **isLastAt(***instant***)** - substituir o termo e a expressão de caminho que o antecede por ”same_as(belongs.temp_lastVersion.getValueAt(instant).getValue)”. A expressão de caminho que precedia *isLastAt* deve anteceder também o método *same_as* e a propriedade *belongs*.
- **isSuccessorOf(***tvObject***)** - esse termo e a expressão de caminho que o antecede devem ser substituídos por “exists y in tvObject.successor: same_as(y)”. A expressão de caminho que precedia *isSucessorOf* deve anteceder também o método *same_as*.
- **isSuccessorOfAt(***tvObject, instant***)** - esse termo e a expressão de caminho que o antecede devem ser substituídos por “exists y in tvObject.temp_successor.getValueAt(instant).getValue.value: same_as(y)”. A expressão de caminho que precedia *isSucessorOf* deve anteceder também o método *same_as*.
- **isPredecessorOf(***tvObject***)** - esse termo e a expressão de caminho que o antecede devem ser substituídos por “exists y in tvObject.predecessor: same_as(y)”. A expressão de caminho que precedia *isPredecessorOf* deve anteceder também o método *same_as*.
- **isAscendantOf(***tvObject***)** - esse termo e a expressão de caminho que o antecede devem ser substituídos por “exists y in tvObject.ascendant: same_as(y)”. A expressão de caminho que precedia *isAscendantOf* deve anteceder também o método *same_as*.
- **isDescendantOf(***tvObject***)** - esse termo e a expressão de caminho que o antecede devem ser substituídos por “exists y in tvObject.descendant: same_as(y)”. A expressão de caminho que precedia *isDescendantOf* deve anteceder também o método *same_as*.
- **isCurrent** - substituir o termo e a expressão de caminho que o antecede por “same_as(belongs.currentVersion)”. A expressão de caminho que precedia *isCurrent* deve anteceder também o método *same_as* e a propriedade *belongs.*
- **isCurrentAt(***instant***)** - esse termo e a expressão de caminho que o antecede devem ser substituídos por “same_as (belongs.temp_currentVersion. getValueAt(instant).getValue)”. A expressão de caminho que precedia *isCurrentAt* deve anteceder também o método *same_as* e a propriedade *belongs*.
- **isUserCurrent** - substituir o termo e a expressão de caminho que o antecede por “same_as(belongs.currentVersion) **and** belongs.userCurrentFlag = **true**”. A expressão de caminho que precedia *isUserCurrent* deve anteceder também o método *same_as* e as duas propriedades *belongs*.

- **isUserCurrentAt(***instant***)** - substituir o termo e a expressão de caminho que o antecede por "same_as (belongs.temp_currentVersion.getValueAt(instant). getValue) **and** belongs.temp_userCurrentFlag.getValueAt(instant).getValue= **true**". A expressão de caminho que precedia *isUserCurrentAt* deve anteceder também o método *same_as* e as duas propriedades *belongs*.
- **isConfiguration** - substituir o termo por "isConfiguration", pois esse é um método definido em *TemporalVersion*.
- Nas expressões de estado e algumas de versões (*isSuccessorOf*, *isSuccessorOfAt*, *isAscendantOf*, *isDescendantOf*, *isPredecessorOf*, *isUserCurrent*, *isUserCurrentAt*) descritas acima, se os termos forem usados como operandos em operações (com exceção daquelas formadas pelos operadores *and*, *or*, *andthen* e *orelse*), então a expressão gerada pelo mapeamento deve ser colocada entre parênteses, para não haver ambigüidade.
- **configurationCount** - o termo deve ser substituído por "belongs.configurationCount".
- **currentVersion** - substituir o termo por "belongs.currentVersion".
- **firstVersion** - substituir o termo por "belongs.firstVersion".
- **lastVersion** - substituir o termo por "belongs.lastVersion".
- **nextVersionNumber** - substituir por "belongs.nextVersionNumber".
- **userCurrentFlag** - substituir o termo por "belongs.userCurrentFlag".
- **versionCount** - substituir o termo por "belongs.versionCount".
- **ascendant** - permanece o termo "ascendant", porém, não mais como palavra reservada.
- **descendant** - permanece o termo "descendant", porém, não mais como palavra reservada.
- **predecessor** - permanece o termo "predecessor", porém, não mais como palavra reservada.
- **successor** - permanece o termo "successor", porém, não mais como palavra reservada.
- **ever** - o mapeamento do *ever* é definido na próxima seção.
- **present** - as partes em que foi aplicado o termo *present* devem ser desconsideradas em relação aos efeitos do *ever*. Após a análise em conjunto com o *ever*, esse termo deve ser excluído.

5.3.1 Regras para o Mapeamento do *Ever*

O *ever* tem efeito sobre propriedades temporais e sobre a maioria das expressões pré-definidas. Entretanto, algumas dessas como *tiInstant*, *viInstant*, *tfInstant*, *vfInstant*, *iLifeTime* e *fLifeTime* devem ser mapeadas de forma especial na presença do *ever*, desconsiderando o mapeamento apresentado anteriormente. É importante ressaltar que se for aplicado o *present* sobre esses termos, então os mesmos não sofrerão o efeito do *ever*. Esta seção apresenta todas essas questões junto ao mapeamento do *ever*.

Para se realizar o mapeamento das consultas, deve-se: i) detectar se o *ever* foi especificado; ii) mapear as consultas de acordo com as regras descritas na seção anterior, observando aquelas que são tratadas de forma especial na presença do *ever*; iii) aplicar as regras descritas nesta seção.

A seguir, são descritos os passos a serem considerados no mapeamento de consultas que possuem *ever*. Alguns passos são exemplificados supondo uma classe Pessoa com *extent* Pessoas e os atributos nome, end, profissão, salário, bônus, rg e data_nsc, sendo os cinco primeiros temporais. A próxima consulta demonstra o uso do *ever* seguido dos passos de seu mapeamento.

```
select ever nome, end, end.viInstant, rg
from Pessoas.versions pes
where data_nsc>date'01-01-1960'
   and bonus+salario=500.00
   and profissao= "B"
   and profissao.viInstant< timestamp'01-01-1998 0:0:0'
```

Passo 1:

- contar número de propriedades no *select* da TV_OQL (contar como apenas uma propriedade, quando é selecionado o histórico de mais de uma das características de uma mesma propriedade temporal, as quais são: o valor da propriedade, o tempo de transação inicial, o tempo de transação final, o tempo de validade inicial e o tempo de validade final, sendo esses tempos indicados por *tiInstant*, *tfInstant*, *viInstant* e *vfInstant*);
- se possuir mais de uma propriedade, então o *select* em OQL contém um *struct* formado por cada uma das propriedades que devem ser selecionadas, possuindo as temporais um *select* aninhado para a seleção do histórico (descrito no passo 2) e as não-temporais apenas selecionam seu valor atual;
- se não houver propriedades no *select* da TV_OQL, apenas * ou um *identifier* para retornar referências a objetos, então a cláusula mapeada deve permanecer com * ou o *identifier*;
- o *from* da OQL continua igual ao da TV_OQL;
- no *where* da TV_OQL, as expressões que envolvem apenas propriedades que não precisam analisar o histórico (dependendo se a propriedade é temporal ou não, ou se o *present* foi aplicado ou não sobre aquela propriedade temporal) permanecem iguais no *where* da OQL;
- no *where* da TV_OQL, as expressões com, pelo menos, uma propriedade que necessite análise de histórico são substituídas, no *where* da OQL, por um teste de existência que verifica se, em algum momento na história, as expressões com essas propriedades são satisfeitas de forma simultânea (descrito no passo 4).

Quando mencionado que as propriedades estão envolvidas em expressões, não são consideradas expressões formadas pelos operadores *and*, *or*, *andthen* e *orelse*.

As partes que apresentam (?) são preenchidas pelas regras posteriores.

```
select struct(nome: ?, end: ?, rg: rg)
from Pessoas pes
where data_nsc>date'01-01-1960'
    and pes.tvOID.getVersionNumber!=0
    and exists (?)
```

Passo 2:

Para a seleção do histórico de propriedades temporais, é criado um *select* aninhado na OQL, que deve ser definido por "**select** *valor* **from** *selectAninhado2*", em que *valor* deve ser substituído:

- por "value", se for desejado os valores do histórico da propriedade;
- por "tempLabel.tTimei", "tempLabel.vTimei", "tempLabel.tTimef" ou "tempLabel.vTimef", conforme o caso, se for desejado os tempos associados aos valores do histórico da propriedade;
- pelas situações descritas nos passos 2.1, 2.2 e 2.3;

E *selectAninhado2* deve ser substituído por uma consulta que retorna os diferentes objetos (não deve ser confundido com diferentes valores) do histórico de valores da propriedade em questão que satisfazem as condições com propriedades, que requerem análise do histórico, definidas na cláusula *where* da TV_OQL (descrito no passo 3).

```
select struct(nome: (select value from ?),
              end: ?
              rg: rg)
from Pessoas pes
where data_nsc>date'01-01-1960'
   and pes.tvOID.getVersionNumber!=0
   and exists (?)
```

Passo 2.1:

Para as propriedades que possuíam a seleção do histórico de mais de uma de suas características no *select* da TV_OQL (como descrito no passo 1), deve-se indicar no *select* da OQL, do passo 2, todas as características que devem ser retornadas.

```
select struct(nome: (select value from ?),
              end: (select value, tempLabel.vTimei from ?),
              rg: rg)
from Pessoas pes
where data_nsc>date'01-01-1960'
    and pes.tvOID.getVersionNumber!=0
    and exists (?)
```

Passo 2.2:

Para a seleção do histórico das expressões de tempo *iLifetime* e *fLifetime* deve-se usar no *select* do passo 2 "tempLabel.vTimei" e "tempLabel.vTimef", respectivamente, sendo a propriedade temporal em questão, nesse caso, o *alive*.

Passo 2.3:

Se houver uma expressão envolvendo mais de uma propriedade (por exemplo, prop1+prop2) no *select* da TV_OQL, e se tem que ser selecionado o histórico de, ao menos, uma das propriedades envolvidas, então deve-se colocar, no *select* do passo 2, a expressão correspondente, usando o respectivo valor do histórico ("value", "tempLabel.tTimei", "tempLabel.vTimei", "tempLabel.tTimef" ou

"tempLabel.vTimef") para cada uma das temporais e o valor atual para as não-temporais.

Passo 3:

Para a seleção dos diferentes objetos do histórico de valores de uma propriedade temporal, os quais satisfazem as condições especificadas na cláusula *where* da TV_OQL que envolvem propriedades que requerem a seleção de seus históricos, deve-se considerar que os valores do histórico a serem selecionados e os valores dos históricos das propriedades, as quais representam as condições, devem ser válidos ao mesmo tempo. Dessa forma, deve ser definido: "**select distinct** x **from** y **where** z", em que:

- x - identificador que representa os objetos do histórico de valores da propriedade a ser selecionada;
- y - seleção dos objetos do histórico de valores da propriedade temporal a ser selecionada. Além disso, para cada uma das propriedades do *where* da TV_OQL, as quais requerem a análise do histórico, é feita a seleção dos objetos de seu histórico de valores que satisfazem as condições do *where* da TV_OQL, que envolvem a respectiva propriedade, podendo essas condições referirem-se ao valor da propriedade ou aos tempos associados ao valor;
- z - intersecção dos tempos do que é definido no *from*, se houver a definição de mais de um conjunto no *from*.

Por questões de espaço e clareza, na intersecção dos tempos, *tempLabel.vTimei* e *tempLabel.vTimef*, estão representados, respectivamente, por *vi* e *vf*, e as consultas que deveriam estar no lugar de *selectAninhadoNome* e *selectAninhadoEnd* são apresentadas no final do trecho mapeado.

Se alguma dessas propriedades, do *where* da TV_OQL, as quais requerem a seleção do histórico, estiver envolvida em uma expressão com mais de uma propriedade, então olhar o passo 5.

```
select struct(nome: (select value from selectAninhadoNome),
              end: (select value, tempLabel.vTimei from selectAninhadoEnd),
              rg: rg)
from Pessoas pes
where data_nsc>date'01-01-1960'
   and pes.tvOID.getVersionNumber!=0
   and exists (?)

selectAninhadoNome
select distinct n
from (select * from pes.temp_nome.contains) n,
      (select * from pes.temp_profissao.contains
        where value= "B" and tempLabel.vTimei < timestamp'01-01-1998 0:0:0') p,
      ?
where  ((n.vi <= p.vf or p.vf=nil) and (n.vf >= p.vi or n.vf=nil)) and
         ?

selectAninhadoEnd
select distinct e
from (select * from pes.temp_end.contains) e,
      (select * from pes.temp_profissao.contains
        where value= "B" and tempLabel.vTimei < timestamp'01-01-1998 0:0:0') p,
      ?
where  ((e.vi <= p.vf or p.vf=nil) and (e.vf >= p.vi or e.vf=nil)) and
         ?
```

Passo 3.1

No caso das expressões de tempo *iLifetime* e *fLifetime*, a propriedade cujos objetos do histórico de valores serão selecionados é o *alive*.

Passo 3.2:

Se houver uma expressão envolvendo mais de uma propriedade no *select* da TV_OQL (como descrito em 2.3), na qual deve ser analisado o histórico de ao menos uma, então para essa expressão o que deve ser alterado em relação ao passo 3 é que deve-se colocar no *select* os identificadores que representam os objetos do histórico de valores de cada uma das propriedades temporais da expressão; e no *from* deve-se colocar a seleção dos objetos dos históricos dessas propriedades temporais envolvidas na expressão.

Passo 4:

Para testar se em algum momento da história as condições especificadas na cláusula *where* da TV_OQL, as quais envolvem propriedades que necessitam da análise do histórico, são satisfeitas de forma simultânea, deve ser definido: "**select distinct * from** x **where** y", em que:

- x - para cada uma dessas propriedades são selecionados os objetos do seu histórico de valores que satisfazem as condições do *where* da TV_OQL, que envolvem a respectiva propriedade, podendo essas condições referirem-se ao valor da propriedade ou aos tempos associados ao valor;
- y - intersecção dos tempos do que é definido no *from*, se houver a definição de mais de um conjunto no *from*.

Se alguma dessas propriedades que requerem a seleção do histórico estiver envolvida em uma expressão com mais de uma propriedade, então olhar o passo 5.

```
select struct(nome: (select value from selectAninhadoNome),
                end: (select value, tempLabel.vTimei from selectAninhadoEnd),
                rg: rg)
from Pessoas pes
where data_nsc>date'01-01-1960'
   and pes.tvOID.getVersionNumber!=0
   and exists (selectAninhadoCondiçõesTemporais)

selectAninhadoNome
select distinct n
from (select * from pes.temp_nome.contains) n,
       (select * from pes.temp_profissao.contains
         where value= "B" and tempLabel.vTimei < timestamp'01-01-1998 0:0:0') p,
        ?
where  ((n.vi <= p.vf or p.vf=nil) and (n.vf >= p.vi or n.vf=nil)) and
         ?

selectAninhadoEnd
select distinct e
from (select * from pes.temp_end.contains) e,
       (select * from pes.temp_profissao.contains
         where value= "B" and tempLabel.vTimei < timestamp'01-01-1998 0:0:0') p,
        ?
where  ((e.vi <= p.vf or p.vf=nil) and (e.vf >= p.vi or e.vf=nil)) and
         ?
```

```
selectAninhadoCondiçõesTemporais
select distinct *
from (select * from pes.temp_profissao.contains
        where value= “B” and tempLabel.vTimei < timestamp’01-01-1998 0:0:0’) p,
       ?
where  ?
```

Passo 5:

Se houver uma expressão envolvendo mais de uma propriedade no *where* da TV_OQL, na qual deve ser analisado o histórico de ao menos uma, então depois de aplicar os passos 3 ou 4, conforme o caso, deve-se aplicar o passo 5. Para a expressão em questão, se houver apenas uma propriedade que deve ter o histórico analisado, então no *from* é acrescentada a seleção dos objetos do histórico dessa propriedade que satisfazem a expressão. Se houver mais de uma propriedade que deve ter o histórico analisado, então deve ser acrescentado no *from* a seleção dos históricos dessas, devendo a condição definida pela expressão correspondente aparecer apenas no *where*.

```
select struct(nome: (select value from selectAninhadoNome),
                rg: rg)
from Pessoas pes
where data_nsc>date’01-01-1960’
   and pes.tvOID.getVersionNumber!=0
   and exists (selectAninhadoCondiçõesTemporais)

selectAninhadoNome
select distinct n
from (select * from pes.temp_nome.contains) n,
      (select * from pes.temp_profissao.contains
        where value= “B” and tempLabel.vTimei < timestamp’01-01-1998 0:0:0’) p,
      (select * from pes.temp_salario.contains) s,
      (select * from pes.temp_bonus.contains) b
where  ((n.vi <= p.vf or p.vf=nil) and (n.vf >= p.vi or n.vf=nil)) and
        ((n.vi <= s.vf or s.vf=nil) and (n.vf >= s.vi or n.vf=nil)) and
        ((p.vi <= s.vf or s.vf=nil) and (p.vf >= s.vi or p.vf=nil)) and
        ((n.vi <= b.vf or b.vf=nil) and (n.vf >= b.vi or n.vf=nil)) and
        ((p.vi <= b.vf or b.vf=nil) and (p.vf >= b.vi or p.vf=nil)) and
        ((b.vi <= s.vf or s.vf=nil) and (b.vf >= s.vi or b.vf=nil)) and
        s.value + b.value=500.00

selectAninhadoEnd
select distinct e
from (select * from pes.temp_end.contains) e,
      (select * from pes.temp_profissao.contains
        where value= “B” and tempLabel.vTimei < timestamp’01-01-1998 0:0:0’) p,
      (select * from pes.temp_salario.contains) s,
      (select * from pes.temp_bonus.contains) b
where  ((e.vi <= p.vf or p.vf=nil) and (e.vf >= p.vi or e.vf=nil)) and
         ((e.vi <= p.vf or p.vf=nil) and (e.vf >= p.vi or e.vf=nil)) and
        ((e.vi <= s.vf or s.vf=nil) and (e.vf >= s.vi or e.vf=nil)) and
        ((p.vi <= s.vf or s.vf=nil) and (p.vf >= s.vi or p.vf=nil)) and
        ((e.vi <= b.vf or b.vf=nil) and (e.vf >= b.vi or e.vf=nil)) and
        ((p.vi <= b.vf or b.vf=nil) and (p.vf >= b.vi or p.vf=nil)) and
        ((b.vi <= s.vf or s.vf=nil) and (b.vf >= s.vi or b.vf=nil)) and
        s.value + b.value=500.00
```

```
selectAninhadoCondiçõesTemporais
select distinct *
from (select * from pes.temp_profissao.contains
        where value= “B” and tempLabel.vTimei < timestamp’01-01-1998 0:0:0’) p,
      (select * from pes.temp_salario.contains) s
      (select * from pes.temp_bonus.contains) b
where  ((p.vi <= s.vf or s.vf=nil) and (p.vf >= s.vi or p.vf=nil)) and
        ((p.vi <= b.vf or b.vf=nil) and (p.vf >= b.vi or p.vf=nil)) and
        ((b.vi <= s.vf or s.vf=nil) and (b.vf >= s.vi or b.vf=nil)) and
        s.value + b.value=500.00
```

5.4 Considerações Finais

Neste capítulo, foi apresentada uma proposta da estrutura necessária para usar o TV_ODMG em um ODBMS que possua suporte apenas ao ODMG, e não à extensão. Como parte dessa estrutura foram mostrados os processadores de TV_ODL e TV_OQL cujas regras de funcionamento foram também exibidas. As regras para o processador de TV_ODL mostraram como transformar declarações em TV_ODL para declarações em ODL. As regras para o processador de TV_OQL apresentaram como transformar consultas em TV_OQL para consultas na sintaxe da OQL, mostrando a aplicabilidade da linguagem, pois várias características ficam transparentes para o usuário, como também consultas simples em TV_ODL podem tornar-se complexas se forem feitas em ODL. Foi definida também a estrutura complementar dos metadados do ODMG para que seja possível armazenar informações referentes às características especificadas para o TV_ODMG.

A utilização do TV_ODMG e as regras mostradas neste capítulo são exemplificadas no estudo de caso apresentado no próximo capítulo.

6 Estudo de Caso

Para exemplificar a extensão do padrão ODMG proposta e o mapeamento dessa extensão, é apresentado um estudo de caso distribuído nos seguintes itens:

- modelagem conceitual de uma aplicação;
- definição das classes em TV_ODL;
- mapeamento para ODL das classes definidas com a TV_ODL;
- exemplos de instâncias para a aplicação;
- consultas que podem ser realizadas sobre essa aplicação usando a TV_OQL;
- mapeamento para OQL das consultas definidas com a TV_OQL.

6.1 Modelagem da Aplicação

Nesta seção, é apresentada a modelagem de uma fábrica de latas. Essas latas podem ser de várias formas e tamanhos, possuindo cada uma a descrição de seus detalhes e uma especificação. Esses produtos são desenvolvidos por grupos de trabalho formados por empregados da fábrica. As classes que descrevem essa aplicação são:

- *Can* - apresenta a capacidade da lata em litros, se essa possui alça, e os relacionamentos com a especificação e com o grupo de trabalho que a desenvolve;
- *CanDetails* - possui os detalhes da lata, como o tipo de tampa e se a lata é revestida internamente;
- *Specification* - especifica uma lata descrevendo sua altura; seu diâmetro inferior e superior (se for cilíndrica) ou suas larguras (se não for cilíndrica); e o arquivo com o desenho da mesma;
- *Workgroup* - define um telefone e um *email* de contato para um grupo de trabalho, assim como o relacionamento desse grupo com os empregados que trabalham nele e as latas que desenvolve;
- *Employee* - define um número de identificação para um empregado, seu nome, seu *email*, sua data de admissão, seu salário, um relacionamento com o grupo em que trabalha e um relacionamento com seu endereço;
- *Address* - determina um nome de rua, um número, um complemento e um CEP para um endereço, como também um relacionamento com a pessoa que vive nele.

A figura 6.1 ilustra essa especificação em um diagrama de classes, no qual as características de versões e/ou tempo são usadas:

- nos atributos *name*, *hiring_date*, *salary*, *capacity*, *internal_coating*, *cover_kind* e *design*, para armazenar o histórico dos seus valores;
- nos relacionamentos de associação *works_in* e *has*, para armazenar o histórico de quais empregados trabalharam em quais grupos de trabalho;

- nos relacionamentos de associação *developed_by* e *develops*, para armazenar o histórico de quais grupos de trabalho desenvolveram quais latas;
- nos relacionamentos de associação *specified_by* e *specifies*, para armazenar o histórico de especificações de cada lata;
- nas classes *Employee*, *Workgroup* e *Specification*, para que possam definir propriedades temporais;
- nas classes *Can* e *CanDetails*, para poderem definir: propriedades temporais; herança por extensão, possibilitando que inicialmente se defina uma lata, e posteriormente sejam definidos seus detalhes; e versões, representando as alternativas de cada tipo de lata.

O diagrama, a seguir, é apresentado em UML com adição da notação descrita pelo TVM, na seção 2.1.2, para as características de tempo e versões.

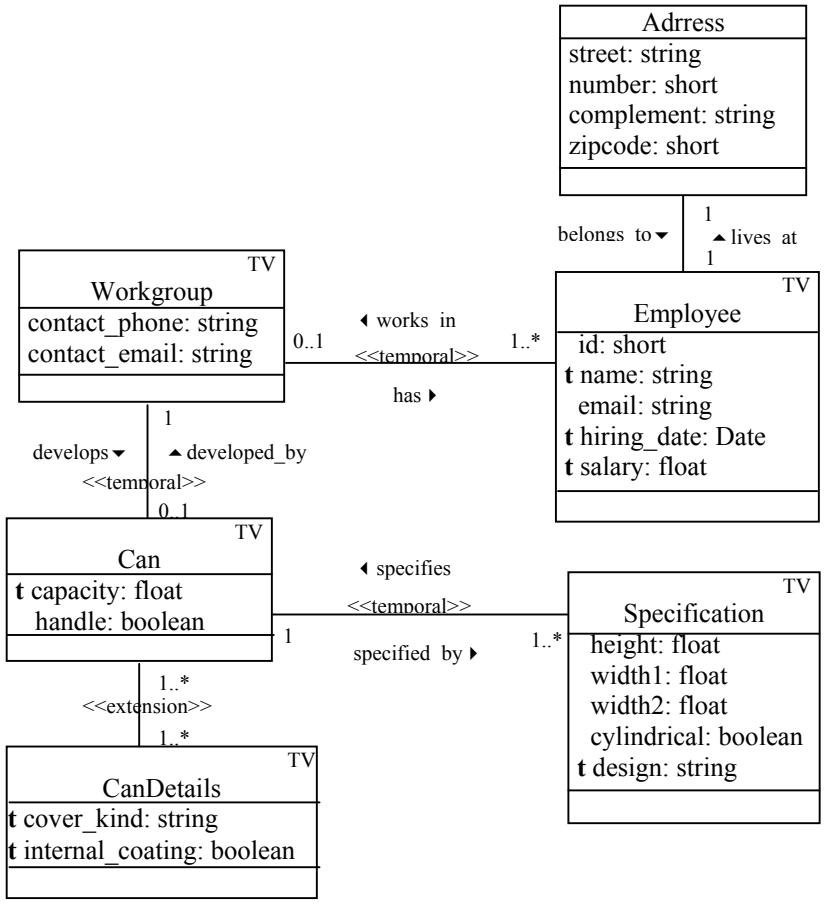

FIGURA 6.1 - Diagrama de classes do estudo de caso

6.2 Definição das Classes em TV_ODL e Mapeamento para ODL

A tabela 6.1 apresenta as descrições dessas classes em TV_ODL, bem como o mapeamento dessas para ODL (usando as regras definidas na seção 5.2).

TABELA 6.1 - Descrição das classes da aplicação em TV_ODL e ODL

TV_ODL	ODL
class Address **(extent** Addresses**)** **{** **attribute string** street; **attribute short** number; **attribute string** complement; **attribute short** zipcode; **relationship** Employee belongs_to **inverse** Employee::lives_at; **};**	**class** Address **(extent** Addresses**)** **{** **attribute string** street; **attribute short** number; **attribute string** complement; **attribute short** zipcode; **relationship** Employee belongs_to **inverse** Employee::livesAt; **};**
class Employee **hasVersion** **(extent** Employees**)** **{** **attribute short** id; **temporal_attribute string** name; **attribute string** email; **temporal_attribute Date** hiring_date; **temporal_attribute float** salary; **temporal_relationship** Workgroug works_in **inverse** Workgroup::has; **relationship** Address lives_at **inverse** Address::belongs_to; **};**	**class** Employee **extends** TemporalVersion **(extent** Employees**)** **{** **attribute short** id; **attribute string** name; **attribute** TemporalAttribute temp_name; **attribute string** email; **attribute Date** hiring_date; **attribute** TemporalAttribute temp_hiring_date; **attribute float** salary; **attribute** TemporalAttribute temp_ salary; **relationship** Workgroup works_in **inverse** Workgroup::has; **attribute** TemporalRelationship temp_works_in; **relationship** Address livesAt **inverse** Address::belongs_to; **};** //Criar instâncias nos metadados em: //*tv_class* //*tv_attribute* para o atributo *name* //*tv_attribute* para o atributo *hiring_date* //*tv_attribute* para o atributo *salary* //*tv_temp_association* para o rel. *works_in*
class Workgroup **hasVersion** **(extent** Workgroups**)** **{** **attribute string** contact_phone; **attribute string** contact_email; **temporal_relationship set**<Can> develops **inverse** Can:: developed_by; **temporal_relationship set**<Employee> has **inverse** Employee:: works_in; **};**	**class** Workgroup **extends** TemporalVersion **(extent** Workgroups**)** **{** **attribute string** contact_phone; **attribute string** contact_email; **relationship set**<Can> develops **inverse** Can:: developed_by; **attribute** TemporalRelationship temp_develops; **relationship set**<Employee> has **inverse** Employee:: works_in; **attribute** TemporalRelationship temp_has; **};** //Criar instâncias nos metadados em: // *tv_class* // *tv_temp_association* para o rel. *develops* // *tv_temp_association* para o rel. *has*
class Can **hasVersion** **(extent** Cans**)** **{** **attribute float** capacity; **temporal_relationship** Workgroup developed_by **inverse** Workgroup::develops; **temporal_ relationship set**<Specification> specified_by **inverse** Specification::specifies; **};**	**class** Can **extends** TemporalVersion **(extent** Cans**)** **{** **attribute float** capacity; **relationship** Workgroup developed_by **inverse** Workgroup::develops; **attribute** TemporalRelationship temp_developed_by; **relationship** Specification specified_by **inverse** Specification:: specifies; **attribute** TemporalRelationship temp_specified_by; **};** //Criar instâncias nos metadados em: // *tv_class* // *tv_temp_association* para o rel. *developed_by* // *tv_temp_association* para o rel. *specified_by*

TV_ODL	ODL

class CanDetails **hasVersion**
  byExtension Can **correspondence n:m**
  (extent CansDetails**)**
{
temporal_attribute string cover_kind;
temporal_attribute boolean internal_coating;
};
	class CanDetails **extends** TemporalVersion
	  (extent CansDetails**)**
	{
	attribute string cover_kind;
	attribute TemporalAttribute temp_cover_kind;
	attribute boolean internal_coating;
	attribute TemporalAttribute temp_internal_coating;
	};
	//Criar instâncias nos metadados em:
	// *tv_class*
	//*tv_attribute* para o atributo *cover_kind*
	//*tv_attribute* para o atributo *internal_coating*
	//*tv_ascClass informando que estende a classe Can*
	//*tv_descClass, ligada à classe Can, informando que*
	//*Can é estendida por CanDetails*

class Specification **hasVersion**
  (extent Specifications**)**
{
attribute float height;
attribute float width1;
attribute float width2;
attribute boolean cylindrical;
temporal_ relationship Can specifies
  inverse Can:: specified_by;
};
	class Specification **extends** TemporalVersion
	  (extent Specifications**)**
	{
	attribute float height;
	attribute float width1;
	attribute float width2;
	attribute boolean cylindrical;
	relationship Can specifies
	  inverse Can::specified_by;
	attribute TemporalRelationship temp_specifies;
	};
	//Criar instâncias nos metadados em:
	// *tv_class*
	// *tv_temp_association* para o rel. *specifies*

6.3 Exemplos de Instâncias

Todas as instâncias que pertencem a uma classe temporal versionável estão associadas a uma entidade através de seu *tvOID*. O usuário pode definir um nome para cada uma dessas entidades; porém, não é obrigado, sendo, neste caso, criados nomes sugestivos para as mesmas. As instâncias que representam essas entidades criadas são armazenadas na classe *TVEntity*.

Nas instâncias da aplicação modelada, cada tipo de lata representa uma entidade, podendo haver várias alternativas para cada uma dessas, as quais são representadas através das versões. Neste exemplo, foram definidos pelo usuário os nomes de entidades galão, balde, mini e cônica, as quais estão representadas por várias versões distribuídas nas classes *Can* e *CanDetails*, devido à herança por extensão. Na figura 6.2, são ilustradas as instâncias da classe *TVEntity*.

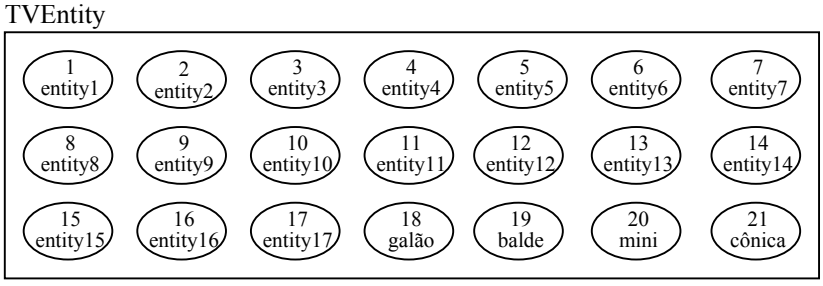

FIGURA 6.2 - Instâncias da classe *TVEntity*

Na figura 6.3, são ilustradas possíveis instâncias para as classes da aplicação. As linhas contínuas representam os relacionamentos, e as setas tracejadas representam a

correspondência entre ascendentes e descendentes na hierarquia por extensão. Os números ao lado de cada objeto estão representando os OIDs dos mesmos, a fim de permitir, neste exemplo, a representação de referências a objetos. Para as classes temporais versionáveis (*Employee*, *Workgroup*, *Specification*, *Can* e *CanDetails*) são mostrados apenas os valores atuais das instâncias e não estão representados aqueles valores das propriedades herdadas de *TemporalObject* e *TemporalVersion*, com exceção de *ascendant* e *descendant*. As propriedades herdadas dessas classes são: *alive*, *tvOID*, *ascendant*, *descendant*, *successor*, *status*, *predecessor*, *configuration* e *belongs*. Portanto, os valores atuais para essas propriedades são apresentados na tabela 6.2.

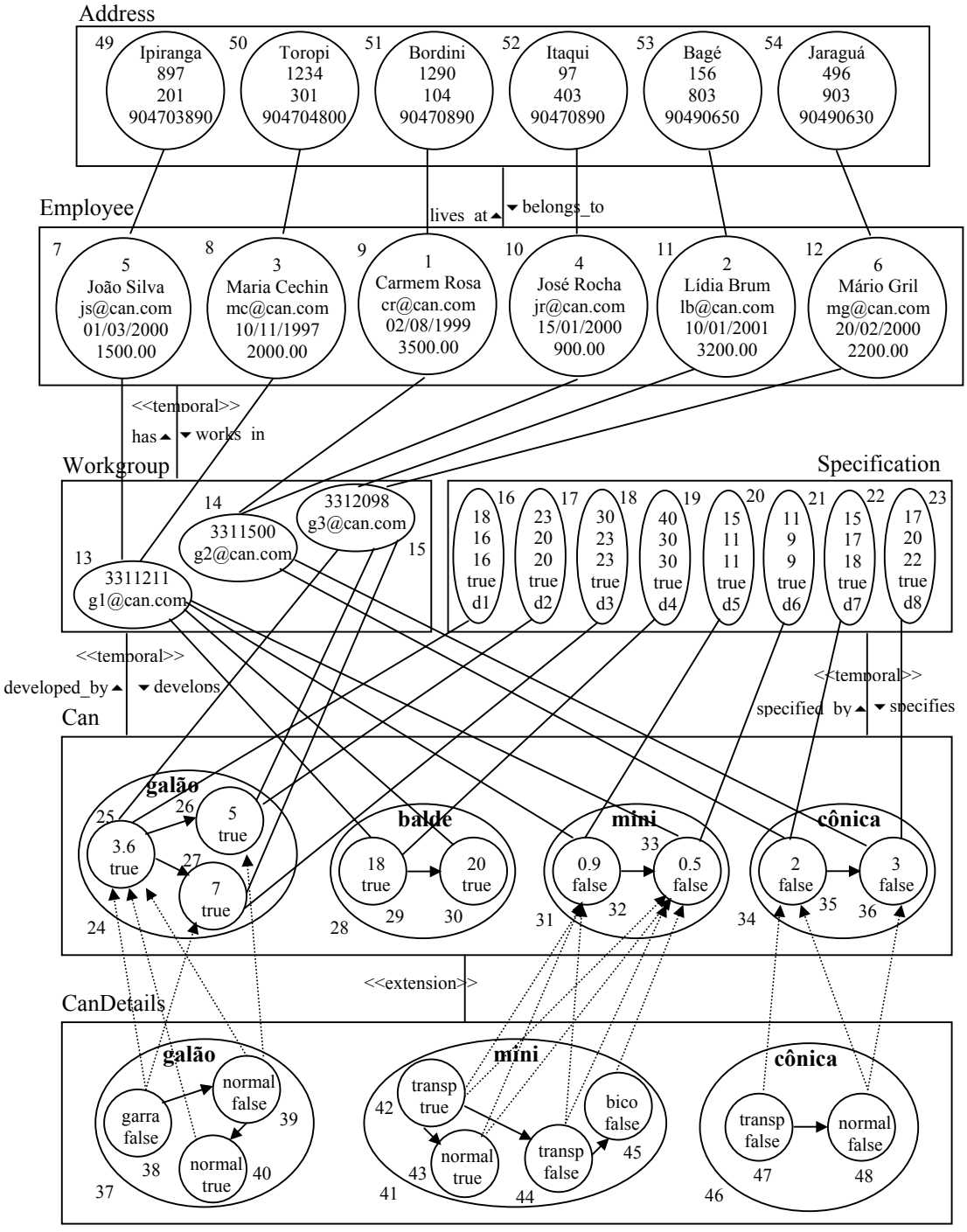

FIGURA 6.3 - Instâncias das classes da aplicação

TABELA 6.2 - Valores atuais das propriedades herdadas

objeto	alive	tvOID	ascendant	descendant	successor	status	predecessor	configuration	belongs
7	true	1,3,1	nil	nil	nil	w	nil	false	nil
8	true	2,3,1	nil	nil	nil	w	nil	false	nil
9	true	3,3,1	nil	nil	nil	w	nil	false	nil
10	true	4,3,1	nil	nil	nil	w	nil	false	nil
11	true	5,3,1	nil	nil	nil	w	nil	false	nil
12	true	6,3,1	nil	nil	nil	w	nil	false	nil
13	true	7,4,1	nil	nil	nil	w	nil	false	nil
14	true	8,4,1	nil	nil	nil	w	nil	false	nil
15	true	9,4,1	nil	nil	nil	w	nil	false	nil
16	true	10,5,1	nil	nil	nil	w	nil	false	nil
17	true	11,5,1	nil	nil	nil	w	nil	false	nil
18	true	12,5,1	nil	nil	nil	w	nil	false	nil
19	true	13,5,1	nil	nil	nil	w	nil	false	nil
20	true	14,5,1	nil	nil	nil	w	nil	false	nil
21	true	15,5,1	nil	nil	nil	w	nil	false	nil
22	true	16,5,1	nil	nil	nil	w	nil	false	nil
23	true	17,5,1	nil	nil	nil	w	nil	false	nil
24	true	18,6,0	nil	nil	nil	w	nil	false	55
25	true	18,6,1	nil	38,39,40	26,27	s	nil	false	55
26	true	18,6,2	nil	39	nil	w	25	false	55
27	true	18,6,3	nil	38	nil	w	25	false	55
28	true	19,6,0	nil	nil	nil	w	nil	false	56
29	true	19,6,1	nil	nil	30	s	nil	false	56
30	true	19,6,2	nil	nil	nil	w	29	false	56
31	true	20,6,0	nil	nil	nil	w	nil	false	57
32	true	20,6,1	nil	42,43,44	33	s	nil	false	57
33	true	20,6,2	nil	42,43,44,45	nil	w	32	false	57
34	true	21,6,0	nil	nil	nil	w	nil	false	58
35	true	21,6,1	nil	47,48	36	c	nil	false	58
36	true	21,6,2	nil	48	nil	w	35	false	58
37	true	18,7,0	nil	nil	nil	w	nil	false	59
38	true	18,7,1	25,27	nil	39	s	nil	false	59
39	true	18,7,2	25,26	nil	40	s	38	false	59
40	true	18,7,3	25	nil	nil	w	39	false	59
41	true	20,7,0	nil	nil	nil	w	nil	false	60
42	true	20,7,1	32,33	nil	43,44	s	nil	false	60
43	true	20,7,2	32,33	nil	nil	w	42	false	60
44	true	20,7,3	32,33	nil	45	s	42	false	60
45	true	20,7,4	33	nil	nil	w	44	false	60
46	true	21,7,0	nil	nil	nil	w	nil	false	61
47	true	21,7,1	35	nil	48	s	nil	false	61
48	true	21,7,2	35,36	nil	nil	w	47	false	61

Como mostrado na tabela anterior, todos os objetos versionados e suas respectivas versões estão associados a uma instância de *VersionedObjectControl* através do relacionamento *belongs*. As instâncias dessa classe são apresentadas na tabela 6.3.

TABELA 6.3 - Valores atuais das instâncias da classe *VerisonedObjectControl*

objeto	alive	version Count	configuration Count	first Version	last Version	current Version	user Current Flag	next Version Number	contains
55	true	3	0	25	27	26	true	4	24,25,26,27
56	true	2	0	29	30	30	false	3	28,29,30
57	true	2	0	32	33	33	false	3	31,32,33

objeto	alive	version Count	configuration Count	first Version	last Version	current Version	user Current Flag	next Version Number	contains
58	true	2	0	35	36	35	true	3	34,35,36
59	true	3	0	38	40	40	false	4	37,38,39,40
60	true	4	0	42	45	44	true	5	41,42,43, 44,45
61	true	2	0	47	48	48	false	3	46,47,48

A seguir, é mostrado, para cada um dos objetos apresentados, o histórico de valores das suas propriedades temporais. Esse histórico é composto por outros objetos que são instâncias da classe *InstantRelationship* ou das subclasses de *InstantAttribute*, conforme o caso. Para simplificar, não serão mostrados os históricos de valores dos objetos versionados e dos controles dos objetos versionados, pois, neste estudo de caso, não serão feitas consultas sobre os históricos dos mesmos.

Entre as propriedades que as classes da aplicação herdam de *TemporalVersion* e *TemporalObject*, existem várias temporais, como: *alive*, *ascendant*, *descendant*, *successor* e *status*. Entretanto, para as classes *Employee*, *Workgroup* e *Specification* não foram definidas alterações nos valores dessas propriedades ao longo do tempo, ou seja, seus tempos de validade e transação são iguais aos tempos da primeira e única vida do objeto, que é definida pelo *alive*. Portanto, para simplificação da apresentação das instâncias dessas classes, dentre essas propriedades são apresentados apenas os valores do atributo *alive*.

Os tempos de validade e transação são do tipo *timestamp*, que é representado por data e hora. Entretanto, nos históricos apresentados a seguir, são mostradas apenas datas para simplificar.

A tabela 6.4 demonstra os históricos de valores das propriedades temporais das instâncias da classe *Employee*.

TABELA 6.4 - Histórico das instâncias da classe *Employee*

objeto	propriedade	obj	value	tempLabel. vTimeI	tempLabel. vTimeF	tempLabel. tTimeI	tempLabel. tTimeF
7	**name**	**701**	João Silva	01/03/2000	nil	01/03/2000	nil
	hiring_date	**700**	01/03/2000	01/03/2000	nil	01/03/2000	nil
	salary	**702**	1500.00	01/03/2000	nil	01/03/2000	nil
	works_in	**704**	13	01/03/2000	nil	01/03/2000	nil
	alive	**706**	true	01/03/2000	nil	01/03/2000	nil
8	**name**	**808**	Maria Souza	10/11/1997	nil	10/11/1997	14/04/1999
		809	Maria Souza	10/11/1997	14/04/1999	15/04/1999	nil
		810	Maria Cechin	15/04/1999	nil	15/04/1999	nil
	hiring_date	**800**	10/11/1997	10/11/1997	nil	10/11/1997	nil
	salary	**801**	1000.00	10/11/1997	nil	10/11/1997	04/11/1999
		802	1000.00	10/11/1997	31/11/1999	05/11/1999	nil
		803	2000.00	01/12/1999	nil	05/11/1999	nil
	works_in	**804**	14	10/11/1997	nil	10/11/1997	24/02/2000
		805	14	10/11/1997	24/02/2000	25/02/2000	nil
		806	13	25/02/2000	nil	25/02/2000	nil
	alive	**807**	true	10/11/1997	nil	10/11/1997	nil

objeto	propriedade	obj	value	tempLabel. vTimeI	tempLabel. vTimeF	tempLabel. tTimeI	tempLabel. tTimeF
9	name	908	Carmem Perez	02/08/1999	nil	02/08/1999	19/10/2000
		909	Carmem Perez	02/08/1999	16/10/2000	20/10/2000	nil
		910	Carmem Rosa	17/10/2000	nil	20/10/2000	nil
	hiring_date	900	02/08/1999	02/08/1999	nil	02/08/1999	nil
	salary	901	3000.00	02/08/1999	nil	02/08/1999	17/11/2000
		902	3000.00	02/08/1999	20/11/2000	18/11/2000	nil
		903	3500.00	21/11/2000	nil	18/11/2000	nil
	works_in	904	15	02/08/1999	nil	02/08/1999	17/11/2000
		905	15	02/08/1999	20/11/2000	18/11/2000	nil
		906	14	21/11/2000	nil	18/11/2000	nil
	alive	907	true	02/08/1999	nil	02/08/1999	nil
10	name	223	José Rocha	15/01/2000	nil	15/01/2000	nil
	hiring_date	110	15/01/2000	15/01/2000	nil	15/01/2000	nil
	salary	111	900.00	15/01/2000	nil	15/01/2000	nil
	works_in	112	14	15/01/2000	nil	15/01/2000	nil
	alive	113	true	15/01/2000	nil	15/01/2000	nil
11	name	220	Lídia Brum	10/01/2001	nil	10/01/2001	nil
	hiring_date	211	10/01/2001	10/01/2001	nil	10/01/2001	nil
	salary	212	3200.00	10/01/2001	nil	10/01/2001	nil
	works_in	213	15	10/01/2001	nil	10/01/2001	nil
	alive	214	true	10/01/2001	nil	10/01/2001	nil
12	name	221	Mário Gril	20/02/2000	nil	20/02/2000	nil
	hiring_date	120	20/02/2000	20/02/2000	nil	20/02/2000	nil
	salary	121	2000.00	20/02/2000	nil	20/02/2000	24/08/2000
		122	2000.00	20/02/2000	19/08/2000	25/08/2000	nil
		123	2200.00	20/08/2000	nil	25/08/2000	nil
	works_in	124	15	20/02/2000	nil	20/02/2000	nil
	alive	125	true	20/02/2000	nil	20/02/2000	nil

A tabela 6.5 apresenta os históricos de valores das propriedades temporais das instâncias da classe *Workgroup*.

TABELA 6.5 - Histórico das instâncias da classe *Workgroup*

objeto	propriedade	obj	value	tempLabel. vTimeI	tempLabel. vTimeF	tempLabel. tTimeI	tempLabel. tTimeF
13	has	130	nil	10/02/2000	nil	10/02/2000	24/02/2000
		131	nil	10/02/2000	24/02/2000	25/02/2000	nil
		132	8	25/02/2000	nil	25/02/2000	28/02/2000
		133	8	25/02/2000	28/02/2000	01/03/2000	nil
		134	7,8	01/03/2000	nil	01/03/2000	nil
	develops	137	nil	10/02/2000	nil	10/02/2000	24/02/2000
		138	nil	10/02/2000	24/02/2000	25/02/2000	nil
		139	32	25/02/2000	nil	25/02/2000	02/09/2000
		103	32	25/02/2000	02/09/2000	03/09/2000	nil
		104	32,33	03/09/2000	nil	03/09/2000	15/12/2001
		105	32,33	03/09/2000	15/12/2001	16/12/2001	nil
		106	32,33,29	16/12/2001	nil	16/12/2001	14/01/2002
		107	32,33,29	16/12/2001	14/01/2002	15/01/2002	nil
		108	32,33,29,30	15/01/2002	nil	15/01/2002	nil
	alive	109	true	10/02/2000	nil	10/02/2000	nil

objeto	propriedade	obj	value	tempLabel. vTimeI	tempLabel. vTimeF	tempLabel. tTimeI	tempLabel. tTimeF
14	has	**140**	nil	05/11/1997	nil	05/11/1997	09/11/1997
		141	nil	05/11/1997	09/11/1997	10/11/1997	nil
		142	8	10/11/1997	nil	10/11/1997	14/01/2000
		143	8	10/11/1997	14/01/2000	15/01/2000	nil
		144	8,10	15/01/2000	nil	15/01/2000	24/02/2000
		145	8,10	15/01/2000	24/02/2000	25/02/2000	nil
		146	10	25/02/2000	nil	25/02/2000	17/11/2000
		147	10	25/02/2000	20/11/2000	18/11/2000	nil
		148	9,10	21/11/2000	nil	18/11/2000	nil
	develops	**149**	nil	05/11/1997	nil	05/11/1997	10/11/1997
		114	nil	05/11/1997	11/11/1997	11/11/1997	nil
		115	35	11/11/1997	nil	11/11/1997	15/04/2000
		116	35	11/11/1997	15/04/2000	16/04/2000	nil
		117	35,36	16/04/2000	nil	16/04/2000	nil
	alive	**118**	true	05/11/1997	nil	05/11/1997	nil
15	has	**151**	nil	22/07/1999	nil	22/07/1999	01/08/1999
		152	nil	22/07/1999	01/08/1999	02/08/1999	nil
		153	9	02/08/1999	nil	02/08/1999	19/02/2000
		154	9	02/08/1999	19/02/2000	20/02/2000	nil
		155	9,12	20/02/2000	nil	20/02/2000	17/11/2000
		156	9,12	20/02/2000	20/11/2000	18/11/2000	nil
		157	12	21/11/2000	nil	18/11/2000	09/01/2001
		158	12	21/11/2000	09/01/2001	10/01/2001	nil
		159	11,12	10/01/2001	nil	10/01/2001	nil
	develops	**126**	nil	22/07/1999	nil	22/07/1999	11/08/1999
		127	nil	22/07/1999	11/08/1999	12/08/1999	
		128	25	12/08/1999	nil	12/08/1999	13/03/2001
		129	25	12/08/1999	13/03/2001	14/03/2001	nil
		119	25,26	14/03/2001	nil	14/03/2001	14/06/2001
		150	25,26	14/03/2001	14/06/2001	15/06/2001	nil
		160	25,26,27	15/06/2001	nil	15/06/2001	nil
	alive	**102**	true	22/07/1999	nil	22/07/1999	nil

A tabela 6.6 mostra os históricos de valores das propriedades temporais das instâncias da classe *Specification*.

TABELA 6.6 - Histórico das instâncias da classe *Specification*

objeto	propriedade	obj	value	tempLabel. vTimeI	tempLabel. vTimeF	tempLabel. tTimeI	tempLabel. tTimeF
16	**design**	**161**	d1	25/09/1999	nil	25/09/1999	nil
	specifies	**162**	25	25/09/1999	nil	25/09/1999	nil
	alive	**163**	true	25/09/1999	nil	25/09/1999	nil
17	**design**	**171**	dx	27/03/2001	nil	27/03/2001	15/09/2001
		172	dx	27/03/2001	30/09/2001	16/09/2001	nil
		173	d2	01/10/2001	nil	16/09/2001	nil
	specifies	**174**	26	27/03/2001	nil	27/03/2001	nil
	alive	**175**	true	27/03/2001	nil	27/03/2001	nil
18	**design**	**181**	d3	15/07/2001	nil	15/07/2001	nil
	specifies	**182**	27	15/07/2001	nil	15/07/2001	nil
	alive	**183**	true	15/07/2001	nil	15/07/2001	nil
19	**design**	**191**	d4	12/01/2002	nil	12/01/2002	nil
	specifies	**192**	29	12/01/2002	nil	12/01/2002	nil
	alive	**193**	true	12/01/2002	nil	12/01/2002	nil

objeto	propriedade	obj	value	tempLabel. vTimeI	tempLabel. vTimeF	tempLabel. tTimeI	tempLabel. tTimeF
20	**design**	**170**	d5	27/03/2000	nil	27/03/2000	nil
	specifies	**180**	32	27/03/2000	nil	27/03/2000	nil
	alive	**190**	true	27/03/2000	nil	27/03/2000	nil
21	**design**	**164**	d6	30/09/2000	nil	30/09/2000	nil
	specifies	**165**	33	30/09/2000	nil	30/09/2000	nil
	alive	**166**	true	30/09/2000	nil	30/09/2000	nil
22	**design**	**184**	dy	20/12/1997	nil	20/12/1997	22/06/2000
		185	dy	20/12/1997	30/06/2000	23/06/2000	nil
		186	d7	01/07/2000	nil	23/06/2000	nil
	specifies	**187**	35	20/12/1997	nil	20/12/1997	nil
	alive	**188**	true	20/12/1997	nil	20/12/1997	nil
23	**design**	**167**	d8	16/05/2000	nil	16/05/2000	nil
	specifies	**168**	36	16/05/2000	nil	16/05/2000	nil
	alive	**169**	true	16/05/2000	nil	16/05/2000	nil

A tabela 6.7 demonstra os históricos de valores das propriedades temporais das instâncias da classe *Can*. Para simplificar, é mostrado apenas o histórico das latas das entidades galão e cônica; não é mostrado o histórico dos objetos versionados dessas, como dito anteriormente, pois neste estudo de caso não são feitas consultas sobre os tempos desses, e os históricos dos mesmos são extensos devido ao fato de armazenarem os valores de todas as versões que já foram corrente.

TABELA 6.7 - Histórico das instâncias da classe *Can*

objeto	propriedade	obj	value	tempLabel. vTimeI	tempLabel. vTimeF	tempLabel. tTimeI	tempLabel. tTimeF
25	**capacity**	**251**	4	12/08/1999	nil	12/08/1999	20/09/1999
		252	4	12/08/1999	20/09/1999	21/09/1999	nil
		253	3.6	21/09/1999	nil	21/09/1999	nil
	specified_by	**254**	nil	12/08/1999	nil	12/08/1999	24/09/1999
		255	nil	12/08/1999	21/09/1999	25/09/1999	nil
		256	16	25/09/1999	nil	25/09/1999	nil
	developed_by	**257**	15	12/08/1999	nil	12/08/1999	nil
	alive	**176**	true	12/08/1999	nil	12/08/1999	nil
	ascendant	**177**	nil	12/08/1999	nil	12/08/1999	nil
	descendant	**204**	nil	12/08/1999	nil	12/08/1999	01/10/1999
		205	nil	12/08/1999	01/10/1999	02/10/1999	nil
		206	38	02/10/1999	nil	02/10/1999	15/04/2000
		207	38	02/10/1999	15/04/2000	16/04/2000	nil
		208	38,39	16/04/2000	nil	16/04/2000	04/01/2001
		209	38,39	16/04/2000	04/01/2001	05/01/2001	nil
		210	38,39,40	05/01/2001	nil	05/01/2001	nil
	successor	**215**	nil	12/08/1999	nil	12/08/1999	13/03/2001
		216	nil	12/08/1999	13/03/2001	14/03/2001	nil
		217	26	14/03/2001	nil	14/03/2001	14/06/2001
		218	26	14/03/2001	14/06/2001	15/06/2001	nil
		219	26,27	15/06/2001	nil	15/06/2001	nil
	status	**250**	w	12/08/1999	nil	12/08/1999	14/06/2001
		258	w	12/08/1999	14/06/2001	15/06/2001	nil
		259	s	15/06/2001	nil	15/06/2001	nil

objeto	propriedade	obj	value	tempLabel.vTimeI	tempLabel.vTimeF	tempLabel.tTimeI	tempLabel.tTimeF
26	capacity	260	5	14/03/2001	nil	14/03/2001	nil
	specified_by	261	nil	14/03/2001	nil	14/03/2001	26/03/2001
		262	nil	14/03/2001	26/03/2001	27/03/2001	nil
		263	17	27/03/2001	nil	27/03/2001	nil
	developed_by	264	15	14/03/2001	nil	14/03/2001	nil
	alive	178	true	14/03/2001	nil	14/03/2001	nil
	ascendant	179	nil	14/03/2001	nil	14/03/2001	nil
	descendant	265	nil	14/03/2001	nil	14/03/2001	04/04/2001
		266	nil	14/03/2001	04/04/2001	05/04/2001	nil
		267	39	05/04/2001	nil	05/04/2001	nil
	successor	268	nil	14/03/2001	nil	14/03/2001	nil
	status	269	w	14/03/2001	nil	14/03/2001	nil
27	capacity	270	7	15/06/2001	nil	15/06/2001	nil
	specified_by	271	nil	15/06/2001	nil	15/06/2001	14/07/2001
		272	nil	15/06/2001	14/07/2001	15/07/2001	nil
		273	18	15/07/2001	nil	15/07/2001	nil
	developed_by	274	15	15/06/2001	nil	15/06/2001	nil
	alive	189	true	15/06/2001	nil	15/06/2001	nil
	ascendant	276	nil	15/06/2001	nil	15/06/2001	nil
	descendant	277	nil	15/06/2001	nil	15/06/2001	19/07/2001
		278	nil	15/06/2001	19/07/2001	20/07/2001	nil
		279	38	20/07/2001	nil	20/07/2001	nil
	successor	275	nil	15/06/2001	nil	15/06/2001	nil
	status	194	w	15/06/2001	nil	15/06/2001	nil
35	capacity	195	2	11/11/1997	nil	11/11/1997	nil
	specified_by	196	nil	11/11/1997	nil	11/11/1997	19/12/1997
		197	nil	11/11/1997	19/12/1997	20/12/1997	nil
		198	22	20/12/1997	nil	20/12/1997	nil
	developed_by	199	14	11/11/1997	nil	11/11/1997	nil
	alive	350	true	11/11/1997	nil	11/11/1997	nil
	ascendant	351	nil	11/11/1997	nil	11/11/1997	nil
	descendant	352	nil	11/11/1997	nil	11/11/1997	22/12/1997
		353	nil	11/11/1997	22/12/1997	23/12/1997	nil
		354	47	23/12/1997	nil	23/12/1997	05/03/2001
		355	47	23/12/1997	05/03/2001	06/03/2001	nil
		356	47,48	06/03/2001	nil	06/03/2001	nil
	successor	357	nil	11/11/1997	nil	11/11/1997	15/04/2000
		358	nil	11/11/1997	15/04/2000	16/04/2000	nil
		359	36	16/04/2000	nil	16/04/2000	nil
	status	304	w	11/11/1997	nil	11/11/1997	15/04/2000
		305	w	11/11/1997	15/04/2000	16/04/2000	nil
		306	s	16/04/2000	nil	16/04/2000	17/04/2000
		307	s	16/04/2000	28/04/2000	18/04/2000	nil
		308	c	29/04/2000	nil	18/04/2000	nil

objeto	propriedade	obj	value	tempLabel. vTimeI	tempLabel. vTimeF	tempLabel. tTimeI	tempLabel. tTimeF
36	capacity	**360**	2.8	16/04/2000	nil	16/04/2000	09/05/2000
		361	2.8	16/04/2000	15/05/2000	18/05/2000	nil
		362	3	16/05/2000	nil	18/05/2000	nil
	specified_by	**363**	nil	16/04/2000	nil	16/04/2000	15/15/2000
		364	nil	16/04/2000	15/05/2000	16/05/2000	nil
		365	23	16/05/2000	nil	16/05/2000	nil
	developed_by	**366**	14	16/04/2000	nil	16/04/2000	nil
	alive	**367**	true	16/04/2000	nil	16/04/2000	nil
	ascendant	**368**	nil	16/04/2000	nil	16/04/2000	nil
	descendant	**369**	nil	16/04/2000	nil	16/04/2000	22/05/2000
		309	nil	16/04/2000	22/05/2000	23/05/2000	nil
		310	48	23/05/2000	nil	23/05/2000	nil
	successor	**311**	nil	16/04/2000	nil	16/04/2000	nil
	status	**312**	w	16/04/2000	nil	16/04/2000	nil

A tabela 6.8 apresenta os históricos de valores das propriedades temporais das instâncias da classe *CanDetails*. Para simplificar, nesta classe também são mostrados os objetos apenas das entidades galão e cônica, bem como, apenas, as versões dessas.

TABELA 6.8 - Histórico das instâncias da classe *CanDetails*

objeto	propriedade	obj	value	tempLabel. vTimeI	tempLabel. vTimeF	tempLabel. tTimeI	tempLabel. tTimeF
38	cover_kind	**380**	normal	02/10/1999	nil	02/10/1999	06/11/1999
		381	normal	02/10/1999	06/11/1999	07/11/1999	nil
		382	garra	07/11/1999	nil	07/11/1999	nil
	internal_coating	**383**	false	02/10/1999	nil	02/10/1999	nil
	alive	**384**	true	02/10/1999	nil	02/10/1999	nil
	ascendant	**385**	25	02/10/1999	nil	02/10/1999	19/07/2001
		386	25	02/10/1999	19/07/2001	20/07/2001	nil
		387	25,27	20/07/2001	nil	20/07/2001	nil
	descendant	**388**	nil	02/10/1999	nil	02/10/1999	nil
	successor	**389**	nil	02/10/1999	nil	02/10/1999	15/04/2000
		313	nil	02/10/1999	15/04/2000	16/04/2000	nil
		314	39	16/04/2000	nil	16/04/2000	nil
	status	**315**	w	02/10/1999	nil	02/10/1999	15/04/2000
		316	w	02/10/1999	15/04/2000	16/04/2000	nil
		317	s	16/04/2000	nil	16/04/2000	nil
39	cover_kind	**390**	normal	16/04/2000	nil	16/04/2000	nil
	internal_coating	**391**	false	16/04/2000	nil	16/04/2000	nil
	alive	**392**	true	16/04/2000	nil	16/04/2000	nil
	ascendant	**393**	25	16/04/2000	nil	16/04/2000	04/04/2001
		394	25	16/04/2000	04/04/2001	05/04/2001	nil
		395	25,26	05/04/2001	nil	05/04/2001	nil
	descendant	**396**	nil	16/04/2000	nil	16/04/2000	nil
	successor	**397**	nil	16/04/2000	nil	16/04/2000	04/01/2001
		398	nil	16/04/2000	04/01/2001	05/01/2001	nil
		399	40	05/01/2001	nil	05/01/2001	nil
	status	**318**	w	16/04/2000	nil	16/04/2000	04/01/2001
		319	w	16/04/2000	04/01/2001	05/01/2001	nil
		320	s	05/01/2001	nil	05/01/2001	nil

objeto	propriedade	obj	value	tempLabel. vTimeI	tempLabel. vTimeF	tempLabel. tTimeI	tempLabel. tTimeF
40	**cover_kind**	**402**	normal	05/01/2001	nil	05/01/2001	nil
	internal_coating	**403**	true	05/01/2001	nil	05/01/2001	nil
	alive	**404**	true	05/01/2001	nil	05/01/2001	nil
	ascendant	**405**	25	05/01/2001	nil	05/01/2001	nil
	descendant	**406**	nil	05/01/2001	nil	05/01/2001	nil
	successor	**407**	nil	05/01/2001	nil	05/01/2001	nil
	status	**408**	w	05/01/2001	nil	05/01/2001	nil
47	**cover_kind**	**470**	transp	23/12/1997	nil	23/12/1997	nil
	internal_coating	**471**	true	23/12/1997	nil	23/12/1997	15/03/1998
		472	true	23/12/1997	15/03/1998	16/03/1998	nil
		473	false	16/03/1998	nil	16/03/1998	nil
	alive	**474**	true	23/12/1997	nil	23/12/1997	nil
	ascendant	**475**	35	23/12/1997	nil	23/12/1997	nil
	descendant	**476**	nil	23/12/1997	nil	23/12/1997	nil
	successor	**477**	nil	23/12/1997	nil	23/12/1997	22/05/2000
		478	nil	23/12/1997	22/05/2000	23/05/2000	nil
		479	48	23/05/2000	nil	23/05/2000	nil
	status	**409**	w	23/12/1997	nil	23/12/1997	22/05/2000
		489	w	23/12/1997	22/05/2000	23/05/2000	nil
		490	s	23/05/2000	nil	23/05/2000	nil
48	**cover_kind**	**480**	normal	23/05/2000	nil	23/05/2000	nil
	internal_coating	**481**	false	23/05/2000	nil	23/05/2000	nil
	alive	**482**	true	23/05/2000	nil	23/05/2000	nil
	ascendant	**483**	36	23/05/2000	nil	23/05/2000	05/03/2001
		484	36	23/05/2000	05/03/2001	06/03/2001	nil
		485	35,36	06/03/2001	nil	06/03/2001	nil
	descendant	**486**	nil	23/05/2000	nil	23/05/2000	nil
	successor	**487**	nil	23/05/2000	nil	23/05/2000	nil
	status	**488**	w	23/05/2000	nil	23/05/2000	nil

6.4 Consultas em TV_OQL e Mapeamento para OQL

Esta seção apresenta algumas consultas em TV_OQL que podem ser realizadas sobre as instâncias das classes definidas na aplicação, bem como o mapeamento das mesmas para OQL (usando as regras definidas na seção 5.3).

Selecionar o atual salário dos empregados que possuem a data de admissão atual entre o período de 01/01/1998 até hoje (supondo que hoje seja 01/01/2002), e quando eles começaram a receber esses valores. Retorno: {(1500.00, 01/03/2000), (3500.00, 21/11/2000), (900.00, 15/01/2000), (3200.00, 10/01/2001), (2200.00, 20/08/2000)}

TV_OQL

```
select e.salary, e.salary.vilnstant
from Employees e
where e.hiring_date.value into [date'01-01-1998'.. nowDate]
```

OQL

```
select e.salary, e.temp_salary.has_as_current.tempLabel.vTimei
from Employees e
where e.hiring_date.value > date'01-01-1998' and
        e.hiring_date.value < date'01-01-2002' and
        ((e.tvOID.getVersionNumber=1 and is_undefined(e.belongs)) or
         e.same_as(e.belongs.currentVersion))
```

Selecionar os empregados que trabalham com latas do tipo cônica, a atual capacidade dessas, como também a especificação atual das mesmas e o período de validade dessas especificações. Retorno:{({9,10}, 2, 22, 20/12/1997, undefined), ({9,10}, 3, 23, 16/05/2000, undefined)}

TV_OQL
```
select c.developed_by.has.value, c.capacity, c.specified_by, c.specified_by.vPeriod
from Cans.versions c
where entity = "cônica"
```

OQL
```
select c.developed_by.has.value, c.capacity, c.specified_by,
       c.temp_specified_by.has_as_current.tempLabel.vTimei,
       c.temp_specified_by.has_as_current.tempLabel.vTimef
from Cans c
where c.tvOID.getEntityName = "cônica" and c.tvOID.getVersionNumber!=0
```

Selecionar um desenho atual de cada tipo de lata e quando esses foram registrados no banco de dados. Retorno:{(d3, 15/07/2001), (undefined, undefined), (d6, 30/09/2000), (d8, 16/05/2000)}

TV_OQL
```
select c.specified_by.design, c.specified_by.design.tiInstant
from Cans c                  //sem versions para pegar as versões corrente
                             //e os objetos que ainda não tem versões, pois é um
                             //desenho de cada tipo
```

OQL
```
select c.specified_by.design,
       c.specified_by.temp_design.has_as_current.tempLabel.tTimei
from Cans c
where ((c.tvOID.getVersionNumber=1 and is_undefined(c.belongs)) or
         c.same_as(c.belongs.currentVersion))
```

Selecionar quais e quando foram formados os diferentes grupos de trabalho que desenvolvem as versões de latas que foram criadas por último em seus respectivos objetos versionados e que possuem *status working*. Retorno:{(15, 22/07/1999), (13, 10/02/2000), (14, 05/11/1997)}

TV_OQL
```
select distinct c.developed_by, c.developed_by.iLifetime
from Cans.versions c
where c.isLast and c.isWorking
```

OQL
```
select distinct c.developed_by, c.developed_by.getLifetimeI
from Cans c
where c.same_as(c.belongs.lastVersion) and c.status= 'w' and
        c.tvOID.getVersionNumber!=0
```

Selecionar os atuais possíveis tipos de tampas para as latas do tipo cônica. Retorno: {transp, normal}

TV_OQL
```
select distinct x.tampa
from (flatten(select c.descendant
              from Cans.versions c
              where c.entity = "cônica")) x
```

OQL
```
select distinct x.tampa
from (flatten(select c.descendant
              from Cans c
              where c.tvOID.getEntityName= "cônica" and
                    c.tvOID.getVersionNumber!=0)) x
```

Selecionar o nome dos empregados que têm salário maior que 3.000,00 e que começaram a receber esse valor no ano de 2001. Retorno:{"Lídia Brum"}

TV_OQL
```
select e.name
from Employees e
where e.salary>3000.00 and e.salary.vilnstant into
                    [timestamp '01-01-2001 0:0:0'..timestamp '31-12-2001 24:0:0']
```

OQL
```
select e.name
from Employees e
where e.salary>3000.00 and
      e.temp_salary.has_as_current.tempLabel.vTimei > timestamp '01-01-2001 0:0:0')
      and
      e.temp_salary.has_as_current.tempLabel.vTimei > timestamp '31-12-2001 24:0:0'
      and ((e.tvOID.getVersionNumber=1 and is_undefined(e.belongs)) or
                                          e.same_as(e.belongs.currentVersion)
```

Selecionar de que lata(s) é derivada a última versão da entidade mini. Retorno:{{32}}

TV_OQL
```
select c.predecessor
from Cans.versions c
where c.entity = "mini" and c.isLast
```

OQL
```
select c.lastVersion.predecessor
from Cans c
where c.tvOID.getEntityName= "mini" and c.same_as(c.belongs.lastVersion) and
      c.tvOID.getVersionNumber!=0
```

Selecionar quando foram formados e o *email* de contato dos grupos de trabalho que já desenvolveram latas das entidades galão ou cônica, que têm ou tiveram capacidade menor que 4 litros e que, atualmente, possuem *status working* ou *stable*. Retorno: {(22/07/1999, "g3@can.com"), (05/11/1997, "g2@can.com")}

```
TV_OQL
select iLifetime, contact_email
from flatten (select ever c.developed_by
                from Can.versions c
                where c.capacity < 4 and (c.entity= “galão” or c.entity= “cônica”) and
                        ( present(c.isWorking) or present(c.isStable) ) )
```

```
OQL
select getLifetimeI, contact_email
from flatten
      (select (select value
                 from (select distinct d
                         from (select * from c.temp_developed_by.contains) d
                                 (select * from c.temp_capacity.contains where value < 4) ca
                         where (d.tempLabel.vTimei <= ca.tempLabel.vTimef
                                                 or ca.tempLabel.vTimef =nil) and
                                 (ca.tempLabel.vTimef >= ca.tempLabel.vTimei
                                                 or d.tempLabel.vTimef =nil) ))
       from Can c
       where c.tvOID.getVersionNumber!=0 and
             (c.tvOID.getEntityName= “galão” or c.tvOID.getEntityName= “cônica”) and
             (c.status= ‘w’ or c.status= ‘s’) and
             exists(select distinct *
                     from (select * from c.temp_capacity.contains where value < 4) ))
```

Selecionar os id dos empregados e os grupos de trabalho de que alguma vez participaram aqueles empregados que foram admitidos depois de 01/01/1999 e que tiveram salário maior que R$ 3.000,00. Retorno: {(1,{14}),(2,{15})}

```
TV_OQL
select ever id, works_in
from Employees emp
where hiring_date > date’01/01/1999’ and salary > 3000.00
```

```
OQL
select struct(id: id, works_in: (select value from selectAninhadoWorksin)),
from Employees emp
where ((emp.tvOID.getVersionNumber=1 and is_undefined(emp.belongs)) or
          emp.same_as(emp.belongs.currentVersion)) and
          exists(selectAninhadoCondições)

//Por uma questão de espaço e melhor entendimento, as consultas que deveriam estar
//no lugar de selectAninhadoCondições e selectAninhadoWorksin estão abaixo.
selectAninhadoCondições
select distinct *
from  (select * from e.temp_hiring_date.contains where value > date’01/01/1999’) h
       (select * from e.temp_salary.contains where value > 3000.00) s
where ((h.tempLabel.vTimei <= s.tempLabel.vTimef or s.tempLabel.vTimef =nil) and
         (h.tempLabel.vTimef >= s.tempLabel.vTimei or h.tempLabel.vTimef =nil))

selectAninhadoWorksin
select distinct w
from (select * from e.temp_works_in.contains) w
       (select * from e.temp_hiring_date.contains where value > date’01/01/1999’) h
       (select * from e.temp_ salary.contains where value > 3000.00) s
where ((w.tempLabel.vTimei <= s.tempLabel.vTimef or s.tempLabel.vTimef =nil) and
       (w.tempLabel.vTimef >= s.tempLabel.vTimei or w.tempLabel.vTimef =nil)) and
       ((w.tempLabel.vTimei <= h.tempLabel.vTimef or h.tempLabel.vTimef =nil) and
       (w.tempLabel.vTimef >= h.tempLabel.vTimei or w.tempLabel.vTimef =nil)) and
        ((h.tempLabel.vTimei <= s.tempLabel.vTimef or s.tempLabel.vTimef =nil) and
         (h.tempLabel.vTimef >= s.tempLabel.vTimei or h.tempLabel.vTimef =nil))
```

6.5 Considerações Finais

Neste capítulo, foi ilustrado um exemplo de aplicação que mostra as características acrescentadas ao padrão ODMG. As classes da aplicação foram definidas em TV_ODL e mapeadas para ODL. Foram ilustradas possíveis instâncias dessas classes a fim de exemplificar as diferentes características das consultas em TV_OQL sobre esses dados, sendo essas consultas também mapeadas para OQL.

7 Conclusões e Trabalhos Futuros

Este trabalho apresentou uma proposta de extensão do padrão ODMG, denominada TV_ODMG, que possibilita lidar com versões de objetos e com o histórico dos dados dos objetos e versões. Para isso, foram estendidos alguns componentes da arquitetura do padrão, como o Modelo de Objetos, a ODL e a OQL, dando origem, respectivamente, ao TV_OM, à TV_ODL e à TV_OQL.

O TV_OM definiu novos tipos objetos na hierarquia de tipos do ODMG para fornecer suporte a algumas definições como: objeto temporal versionável, versão, objeto versionado, configuração, herança por extensão, atributo temporal, relacionamento de associação temporal, relacionamento de agregação temporal e não-temporal, tempo de vida de um objeto, exclusão lógica, tempo de transação e de validade, regras de integridade temporal e estados de um objeto. A TV_ODL mostra como devem ser especificadas as classes temporais versionáveis, a herança por extensão entre elas e as suas propriedades temporais. E por fim, a TV_OQL introduziu uma série de funções para lidar com as características definidas pelo TV_OM.

O diferencial deste trabalho está na definição de uma extensão do padrão ODMG que reúne os conceitos de versão de objeto e dimensão temporal (tempo de transação e de validade), baseando-se no TVM, o qual apresenta duas diferentes ordens de tempo, ramificado para o objeto e linear para cada versão, como também possibilita a presença de versões em diferentes níveis da hierarquia de tipos/classes, pois os outros modelos admitem versões apenas no nível mais especializado da hierarquia.

Dessa forma, a especificação de um sistema com o TV_ODMG pode ser feita considerando as alternativas de projeto, como também a evolução histórica dos dados. Outra característica importante é a facilidade de integração com especificações existentes, pois não é obrigatório que todas as classes sejam temporais versionáveis.

Além disso, para que não fosse necessário o uso de um ODBMS específico para TV_ODMG, foi definida uma estrutura composta de processadores para a TV_ODL e para a TV_OQL que tem a função de mapear essa extensão para o ODMG. Foram especificadas as regras para esses processadores, como também foi definida uma estrutura complementar para os metadados do ODMG para armazenar as características específicas do TV_ODMG.

Um dos objetivos também dessa extensão é servir como base para a extensão dos *bindings*. Se fosse desenvolvida uma extensão diretamente em um determinado *binding*, haveria uma certa dificuldade em fazer a extensão para outro *binding* porque nem todas as características são suportadas pelas três linguagens. Definindo de forma abstrata, tem-se uma especificação mais completa do que todos os *bindings* devem possuir.

Para validar esse trabalho, foi desenvolvida uma modelagem conceitual especificada com a TV_ODL, sendo apresentado, também, o mapeamento dessa para ODL, de acordo com as regras definidas para o processador de TV_ODL. Além disso, foram mostradas possíveis instâncias para a aplicação e possíveis consultas em TV_OQL que poderiam ser aplicadas sobre essas instâncias, como também o mapeamento dessas consultas para OQL, de acordo com as regras do processador de TV_OQL.

Durante o desenvolvimento desse trabalho, houve dificuldade em relação ao entendimento de alguns conceitos do ODMG que não estão expostos de forma clara na literatura.

E por fim, entre os aprimoramentos e futuras extensões que podem ser feitos a esse trabalho estão:

- definir as operações para os metadados;
- definir o corpo dos métodos das classes da biblioteca e as *triggers* que devem ser utilizadas;
- definir, na TV_OQL, o funcionamento do *ever* para as cláusulas *groupby* e *having*;
- implementar os processadores de TV_ODL e TV_OQL;
- estender para os *bindings* C++, Java e Smalltalk;
- implementar um sistema gerenciador que suporte diretamente o TV_ODMG, isto é, sem a necessidade de mapeá-lo para o ODMG.

Anexo 1 Classes do TV_ODMG

Este anexo apresenta a definição completa das classes acrescentadas ao padrão ODMG.

```
class TemporalObject
{
temporal_attribute boolean alive default true;

Object TemporalObject ( );

boolean delete ( );
boolean getAlive ( );
timestamp getLifetimeI ( );
timestamp getLifetimeF ( );
set<InstantAttribute> getObjectHistory ( );
void closeTemporalLabels ( );
set<InstantAttribute> getAttributeHistory (in string attrName);
InstantAttribute getAttributeValueAt (in string attrName, in timestamp i);
set<InstantRelationship> getRelationshipHistory (in string relName);
InstantRelationship getRelationshipValueAt (in string relName, in timestamp i);
void setTemporalAttribute (in string attrName, in Object newValue);
void setTemporalAttribute (in string attrName, in Object newValue, in timestamp iValidTime,
    in timestamp fValidTime);
void setTemporalAttribute (in string attrName, in long newValue);
void setTemporalAttribute (in string attrName, in long newValue, in timestamp iValidTime,
    in timestamp fValidTime);
void setTemporalAttribute (in string attrName, in short newValue);
void setTemporalAttribute (in string attrName, in short newValue, in timestamp iValidTime,
    in timestamp fValidTime);
void setTemporalAttribute (in string attrName, in long long newValue);
void setTemporalAttribute (in string attrName, in long long newValue,
    in timestamp iValidTime, in timestamp fValidTime);
void setTemporalAttribute (in string attrName, in unsigned long newValue);
void setTemporalAttribute (in string attrName, in unsigned long newValue,
    in timestamp iValidTime, in timestamp fValidTime);
void setTemporalAttribute (in string attrName, in unsigned short newValue);
void setTemporalAttribute (in string attrName, in unsigned short newValue,
    in timestamp iValidTime, in timestamp fValidTime);
void setTemporalAttribute (in string attrName, in float newValue);
void setTemporalAttribute (in string attrName, in float newValue, in timestamp iValidTime,
    in timestamp fValidTime);
void setTemporalAttribute (in string attrName, in double newValue);
void setTemporalAttribute (in string attrName, in double newValue, in timestamp iValidTime,
    in timestamp fValidTime);
void setTemporalAttribute (in string attrName, in boolean newValue);
void setTemporalAttribute (in string attrName, in boolean newValue,
    in timestamp iValidTime, in timestamp fValidTime);
void setTemporalAttribute (in string attrName, in octet newValue);
void setTemporalAttribute (in string attrName, in octet newValue, in timestamp iValidTime,
    in timestamp fValidTime);
void setTemporalAttribute (in string attrName, in char newValue);
void setTemporalAttribute (in string attrName, in char newValue, in timestamp iValidTime,
    in timestamp fValidTime);
void setTemporalAttribute (in string attrName, in string newValue);
void setTemporalAttribute (in string attrName, in string newValue, in timestamp iValidTime,
    in timestamp fValidTime);
void setTemporalAttribute (in string attrName, in set<t> newValue);
void setTemporalAttribute (in string attrName, in set<t> newValue, in timestamp iValidTime,
```

```
    in timestamp fValidTime);
void setTemporalAttribute (in string attrName, in list<t> newValue);
void setTemporalAttribute (in string attrName, in list<t> newValue, in timestamp iValidTime,
    in timestamp fValidTime);
void setTemporalAttribute (in string attrName, in bag<t> newValue);
void setTemporalAttribute (in string attrName, in bag<t> newValue, in timestamp iValidTime,
    in timestamp fValidTime);
void setTemporalAttribute (in string attrName, in array<t> newValue);
void setTemporalAttribute (in string attrName, in array<t> newValue,
    in timestamp iValidTime, in timestamp fValidTime);
void setTemporalAttribute (in string attrName, in dictionary<t,v> newValue);
void setTemporalAttribute (in string attrName, in dictionary<t,v> newValue,
    in timestamp iValidTime, in timestamp fValidTime);
void setTemporalAttribute (in string attrName, in date newValue);
void setTemporalAttribute (in string attrName, in date newValue, in timestamp iValidTime,
    in timestamp fValidTime);
void setTemporalAttribute (in string attrName, in time newValue);
void setTemporalAttribute (in string attrName, in time newValue, in timestamp iValidTime,
    in timestamp fValidTime);
void setTemporalAttribute (in string attrName, in timestamp newValue);
void setTemporalAttribute (in string attrName, in timestamp newValue,
    in timestamp iValidTime, in timestamp fValidTime);
void setTemporalAttribute (in string attrName, in interval newValue);
void setTemporalAttribute (in string attrName, in interval newValue, in timestamp iValidTime,
    in timestamp fValidTime);
void setTemporalRelationship (in string relName, in Object newValue);
void setTemporalRelationship (in string relName, in Object newValue,
     in timestamp iValidTime, in timestamp fValidTime);
void setTemporalRelationship (in string relName, in set<Object> newValue);
void setTemporalRelationship (in string relName, in set<Object> newValue,
     in timestamp iValidTime, in timestamp fValidTime);
void setTemporalRelationship (in string relName, in bag<Object> newValue);
void setTemporalRelationship (in string relName, in bag<Object> newValue,
     in timestamp iValidTime, in timestamp fValidTime);
void setTemporalRelationship (in string relName, in list<Object> newValue);
void setTemporalRelationship (in string relName, in list<Object> newValue,
     in timestamp iValidTime, in timestamp fValidTime);
};

class TemporalVersion extends TemporalObject
{
attribute OIDt tvOID;
temporal_attribute set<Object> ascendant;
temporal_attribute set<Object> descendant;
temporal_attribute set<Object> successor;
temporal_attribute char status default 'w';
attribute set<Object> predecessor;
attribute boolean configuration default false;
relationship VersionedObjectControl belongs
     inverse VersionedObjectControl contains;

Object TemporalVersion ( );
Object TemporalVersion (in set<Object> ascendant);
Object TemporalVersion (in string entityName);
Object TemporalVersion (in short entityId, in short classId, in short versionId);
                        //serve para gerar um objeto versionado ou uma versão
Object TemporalVersion (in set<Object>  predecId, in set<Object> ascendId,
    in boolean config);          //serve para gerar uma versão

boolean addAscendant (in Object ascendId);
```

```
boolean addDescendant (in Object descendId);
void addSucessor (in Object succId);
void delete (in boolean allReferences);
void deleteObjectTree (in boolean allReferences);
Object derive (in set<Object> versionId);
Object derive (in set<Object> versionId, in set<Object> ascendId, in boolean config);
set<Object> getAscendant ( );
set<Object> getAscendant (in boolean all);
Object getConfiguration ( );
boolean isConfiguration ( );
set<Object> getCompletObject ( );
set<Object> getDescendant ( );
set<Object> getDescendant (in boolean all);
set<Object> getPredecessor ( );
char getStatus ( );
set<Object> getSucessor (in boolean onlyConfigured);
Object getVersionedObjectId ( );
Object getVersionedObjectControl ( );
boolean isDeleteAllowed (in boolean allReferences);
boolean isDeleteTreeAllowed (in boolean allReferences);
void promote (in boolean allAscendant, in boolean allReferenced);
boolean removeAscendant (in Object ascendId);
void removeDescendant (in Object descendId);
void removeSuccessor (in Object succId);
boolean restore (in Object oid);
void setAscendant (in Object ascendId);
void setDescendant (in Object descendId);
void setStatus (in char newStatus);
boolean setSuccessor (in set<Object> succId);
boolean verifyAscendId (in set<Object> ascendId);
OIDt getTVOid ( );
string getCorrespondence (in string ascendClassName);
string getCorrespondenceAsc (in string ascendClassName);
string getCorrespondenceDesc (in string descendClassName);
string getClassName ( );
short getEntityId (in string entityName);
boolean verifyEntityName (in string entityName);
short findVersion (in short entityId, in short classId);
string getClassId ( );        //retorna o identificador da subclasse de TemporalVersion
                              //que foi instanciada
};

class VersionedObjectControl extends TemporalObject
   (extent VersionedObjectControls)
{
temporal_attribute short versionCount default 2;
temporal_attribute short configurationCount default 0;
temporal_attribute Object firstVersion;
temporal_attribute Object lastVersion;
temporal_attribute Object currentVersion;
temporal_attribute boolean userCurrentFlag default false;
attribute short nextVersionNumber default 3;
relationship set<TemporalVersion> contains
     inverse TemporalVersion belongs;

Object VersionedObjectControl (in Object firstV, in Object lastV);
void delete(in short entityId, in short classId);
short getConfigurationCount ( );
short getVersionCount ( );
short getNextVersionNumber ( );
```

```
boolean getUserCurrentFlag ( );
Object getFirstVersion ( );
Object getLastVersion ( );
Object getCurrentVersion ( );
void setCurrentVersion ( );
void setCurrentVersion (in Object newVersionId);
void updateConfigurationCount ( );
void setFirstVersion (in Object first);
void setLastVersion (in Object last);
void setUserCurrentFlag (in boolean flag);
void updateVersionCount ( );
void updateNextVersionNumber ( );
};

class TVEntity
  (extent TVEntitys)
{
attribute short id;
attribute string name;

Object TVEntity (in string name);
Object TVEntity ( );      //cria um entidade com um nome definido pelo sistema
short getId ( );
string getName ( );
};

class OIDt
  (extent OIDts)
{
attribute string value;

Object OIDt (in string oid);
short getClassNr ( );
short getEntityNr ( );
short getVersionNr ( );
string getValue ( );
string getEntityName ( );
};

class TemporalLabel
    (extent TemporalLabels)
{
attribute timestamp tTimei;
attribute timestamp tTimef;
attribute timestamp vTimei;
attribute timestamp vTimef;

Object TemporalLabel (in timestamp iValidTime, in timestamp fValidTime,
     in timestamp iTransTime, in timestamp fTransTime);

timestamp getTTimei ( );
timestamp getTTimef ( );
timestamp getVTimei ( );
timestamp getVTimef ( );
void setTTimef (in timestamp i);
void setVTimef (in timestamp i);
};

class TemporalRelationship
   (extent TemporalRelationships)
```

```
{
relationship set<InstantRelationship> contains
    inverse InstantRelationship belongs;
relationship InstantRelationship has_as_current   //valor corrente do relacionamento
    inverse InstantRelationship is_current_of;

Object TemporalRelationship (in Object val,  in timestamp iValidTime);
Object TemporalRelationship (in Object val,  in timestamp iValidTime,
     in timestamp fValidTime);
Object TemporalRelationship (in set<Object> val,  in timestamp iValidTime);
Object TemporalRelationship (in set<Object> val,  in timestamp iValidTime,
     in timestamp fValidTime);
Object TemporalRelationship (in list<Object> val,  in timestamp iValidTime);
Object TemporalRelationship (in list <Object> val,  in timestamp iValidTime,
     in timestamp fValidTime);
Object TemporalRelationship (in bag<Object> val,  in timestamp iValidTime);
Object TemporalRelationship (in bag<Object> val,  in timestamp iValidTime,
     in timestamp fValidTime);

InstantRelationship getCurrent ( );
set<InstantRelationship> getHistory ( );
set<TemporalLabel> getHistoryOfValue (in Object val);
set<TemporalLabel> getHistoryOfValue (in set<Object> val);
set<TemporalLabel> getHistoryOfValue (in list<Object> val);
set<TemporalLabel> getHistoryOfValue (in bag<Object> val);
InstantRelationship getValueAt (in timestamp at);
void updateValue (in Object val);
void updateValue (in Object val, in timestamp iValidTime, in timestamp fValidTime);
void updateValue (in set<Object> val);
void updateValue (in set<Object>val, in timestamp iValidTime, in timestamp fValidTime);
void updateValue (in list<Object>val);
void updateValue (in list<Object>val, in timestamp iValidTime, in timestamp fValidTime);
void updateValue (in bag<Object>val);
void updateValue (in bag<Object>val, in timestamp iValidTime, in timestamp fValidTime);
void closeLabels ( );
};

class TemporalAttribute
   (extent TemporalAttributes)
{
attribute string type;
relationship set<InstantAttribute> contains
    inverse InstantAttribute belongs;
relationship InstantAttribute has_as_current       // valor corrente do atributo
    inverse InstantAttribute is_current_of;

Object TemporalAttribute (in string type, in Object val, in timestamp iValidTime,
     in timestamp fValidTime);    //construtor para os tipos objetos
Object TemporalAttribute (in string type, in Object val, in timestamp iValidTime);
                                  //construtor para os tipos objetos
Object TemporalAttribute (in string type, in string val, in timestamp iValidTime,
     in timestamp fValidTime);    //construtor para os tipos literais atômicos
Object TemporalAttribute (in string type, in string val, in timestamp iValidTime);
                                  //construtor para os tipos literais atômicos
Object TemporalAttribute (in string type, in set<t> val, in timestamp iValidTime,
     in timestamp fValidTime);
Object TemporalAttribute (in string type, in set<t> val, in timestamp iValidTime);
Object TemporalAttribute (in string type, in bag<t> val, in timestamp iValidTime,
     in timestamp fValidTime);
Object TemporalAttribute (in string type, in bag<t> val, in timestamp iValidTime);
```

```
Object TemporalAttribute (in string type, in list<t> val, in timestamp iValidTime,
     in timestamp fValidTime);
Object TemporalAttribute (in string type, in list<t> val, in timestamp iValidTime);
Object TemporalAttribute (in string type, in array<t> val, in timestamp iValidTime,
     in timestamp fValidTime);
Object TemporalAttribute (in string type, in array<t> val, in timestamp iValidTime);
Object TemporalAttribute (in string type, in dictionary<t,v> val, in timestamp iValidTime,
     in timestamp fValidTime);
Object TemporalAttribute (in string type, in dictionary<t,v> val, in timestamp iValidTime);
Object TemporalAttribute (in string type, in date val, in timestamp iValidTime,
     in timestamp fValidTime);
Object TemporalAttribute (in string type, in date val, in timestamp iValidTime);
Object TemporalAttribute (in string type, in time val, in timestamp iValidTime,
     in timestamp fValidTime);
Object TemporalAttribute (in string type, in time val, in timestamp iValidTime);
Object TemporalAttribute (in string type, in timestamp val, in timestamp iValidTime,
     in timestamp fValidTime);
Object TemporalAttribute (in string type, in timestamp val, in timestamp iValidTime);
Object TemporalAttribute (in string type, in interval val, in timestamp iValidTime,
     in timestamp fValidTime);
Object TemporalAttribute (in string type, in interval val, in timestamp iValidTime);

InstantAttribute getCurrent ( );
set<InstantAttribute> getHistory ( );
InstantAttribute getValueAt (in timestamp at);
set<TemporalLabel> getHistoryOfValue (in Object val);       // para tipos objetos
set<TemporalLabel> getHistoryOfValue (in string val); //para tipos literais atômicos
set<TemporalLabel> getHistoryOfValue (in set<t> val);
set<TemporalLabel> getHistoryOfValue (in bag<t> val);
set<TemporalLabel> getHistoryOfValue (in list<t> val);
set<TemporalLabel> getHistoryOfValue (in array<t> val);
set<TemporalLabel> getHistoryOfValue (in dictionary<t,v> val);
set<TemporalLabel> getHistoryOfValue (in date val);
set<TemporalLabel> getHistoryOfValue (in time val);
set<TemporalLabel> getHistoryOfValue (in timestamp val);
set<TemporalLabel> getHistoryOfValue (in interval val);
string getType ( );
void updateValue (in Object val);    // para tipos objetos
void updateValue (in Object val, in timestamp iValidTime, in timestamp fValidTime);
                                                    // para tipos objetos
void updateValue (in string val);     //para tipos literais atômicos
void updateValue (in string val, in timestamp iValidTime, in timestamp fValidTime);
                                                          //para tipos literais atômicos
void updateValue (in set<t> val);
void updateValue (in set<t> val, in timestamp iValidTime, in timestamp fValidTime);
void updateValue (in bag<t> val);
void updateValue (in bag<t> val, in timestamp iValidTime, in timestamp fValidTime);
void updateValue (in list<t> val);
void updateValue (in list<t> val, in timestamp iValidTime, in timestamp fValidTime);
void updateValue (in array<t> val);
void updateValue (in array<t> val, in timestamp iValidTime, in timestamp fValidTime);
void updateValue (in dictionary<t,v> val);
void updateValue (in dictionary<t,v> val, in timestamp iValidTime, in timestamp fValidTime);
void updateValue (in date val);
void updateValue (in date val, in timestamp iValidTime, in timestamp fValidTime);
void updateValue (in time val);
void updateValue (in time val, in timestamp iValidTime, in timestamp fValidTime);
void updateValue (in timestamp val);
void updateValue (in timestamp val, in timestamp iValidTime, in timestamp fValidTime);
void updateValue (in interval val);     //para tipos literais atômicos
```

```
void updateValue (in interval val, in timestamp iValidTime, in timestamp fValidTime);
void closeLabels ( );
};

class InstantRelationship
   (extent InstantRelationships)
{
attribute TemporalLabel tempLabel;
relationship TemporalRelationship belongs
     inverse TemporalRelationship contains;
relationship TemporalRelationship is_current_of
     inverse TemporalRelationship has_as_current;

TemporalLabel getTempLabel ( );
void setFTransactionTime (in timestamp i);   //é chamada pelo updateValue
void setFValidTime (in timestamp i);         //é chamada pelo updateValue
};

class InstantAttribute
  (extent InstantAttributes)
{
attribute TemporalLabel tempLabel;
relationship TemporalAttribute belongs
     inverse TemporalAttribute contains;
relationship TemporalAttribute is_current_of
     inverse TemporalAttribute has_as_current;

TemporalLabel getTempLabel ( );
void setFTransactionTime (in timestamp i);  //é chamada pelo updateValue
void setFValidTime (in timestamp i);        //é chamada pelo updateValue
};

class HRObject extends InstantRelationship
   (extent HRObjects)
{
attribute Object value;
Object HObject (in Object val, in timestamp iValidTime, in timestamp fValidTime);
Object HObject (in Object val, in timestamp iValidTime);
Object getValue ( );
};

class HRSet extends InstantRelationship
   (extent HRSets)
{
attribute set<Object> value;
Object HRSet (in set<Object> val, in timestamp iValidTime, in timestamp fValidTime);
Object HRSet (in set<Object> val, in timestamp iValidTime);
Object getValue ( );
};

class HRList extends InstantRelationship
   (extent HRLists)
{
attribute list<Object> value;
Object HRList (in list<Object> val, in timestamp iValidTime, in timestamp fValidTime);
Object HRList (in list<Object> val, in timestamp iValidTime);
Object getValue ( );
};

class HRBag extends InstantRelationship
```

```
   (extent HRBags)
{
attribute bag<Object> value;
Object HRBag (in bag<Object> val, in timestamp iValidTime, in timestamp fValidTime);
Object HRBag (in bag<Object> val, in timestamp iValidTime);
Object getValue ( );
};


class HObject extends InstantAttribute
   (extent HObjects)
{
attribute Object value;
Object HObject (in Object val, in timestamp iValidTime, in timestamp fValidTime);
Object HObject (in Object val, in timestamp iValidTime);
Object getValue ( );
};

class Hlong extends InstantAttribute
   (extent Hlongs)
{
attribute long value;
Object Hlong (in string val, in timestamp iValidTime, in timestamp fValidTime);
Object Hlong (in string val, in timestamp iValidTime);
long getValue ( );
};

class Hlonglong extends InstantAttribute
   (extent Hlonglongs)
{
attribute long long value;
Object Hlonglong (in string val, in timestamp iValidTime, in timestamp fValidTime);
Object Hlonglong (in string val, in timestamp iValidTime);
long long getValue ( );
};

class Hshort extends InstantAttribute
   (extent Hshorts)
{
attribute short value;
Object Hshort (in string val, in timestamp iValidTime, in timestamp fValidTime);
Object Hshort (in string val, in timestamp iValidTime);
short getValue ( );
};

class Hunsignedlong extends InstantAttribute
   (extent Hunsignedlongs)
{
attribute unsigned long value;
Object Hunsignedlong (in string val, in timestamp iValidTime, in timestamp fValidTime);
Object Hunsignedlong (in string val, in timestamp iValidTime);
unsigned long getValue ( );
};

class Hunsignedshort extends InstantAttribute
   (extent Hunsignedshorts)
{
attribute unsigned short value;
Object Hunsignedshort (in string val, in timestamp iValidTime, in timestamp fValidTime);
Object Hunsignedshort (in string val, in timestamp iValidTime);
```

```
unsigned short getValue ( );
};

class Hfloat extends InstantAttribute
   (extent Hfloats)
{
attribute float value;
Object Hfloat (in string val, in timestamp iValidTime, in timestamp fValidTime);
Object Hfloat (in string val, in timestamp iValidTime);
float getValue ( );
};

class Hdouble extends InstantAttribute
   (extent Hdoubles)
{
attribute double value;
Object Hdouble (in string val, in timestamp iValidTime, in timestamp fValidTime);
Object Hdouble (in string val, in timestamp iValidTime);
double getValue ( );
};

class Hboolean extends InstantAttribute
   (extent Hbooleans)
{
attribute boolean value;
Object Hboolean (in string val, in timestamp iValidTime, in timestamp fValidTime);
Object Hboolean (in string val, in timestamp iValidTime);
boolean getValue ( );
};

class Hoctet extends InstantAttribute
   (extent Hoctets)
{
attribute octet value;
Object Hoctet (in string val, in timestamp iValidTime, in timestamp fValidTime);
Object Hoctet (in string val, in timestamp iValidTime);
octet getValue ( );
};

class Hchar extends InstantAttribute
   (extent Hchars)
{
attribute char value;
Object Hchar (in string val, in timestamp iValidTime, in timestamp fValidTime);
Object Hchar (in string val, in timestamp iValidTime);
char getValue ( );
};

class Hstring extends InstantAttribute
   (extent Hstrings)
{
attribute string value;
Object Hstring (in string val, in timestamp iValidTime, in timestamp fValidTime);
Object Hstring (in string val, in timestamp iValidTime);
string getValue ( );
};

class HSet extends InstantRelationship
   (extent HSets)
{
```

```
attribute set<t> value;
Object HSet (in set<t> val, in timestamp iValidTime, in timestamp fValidTime);
Object HSet (in set<t> val, in timestamp iValidTime);
Object getValue ( );
};

class HList extends InstantRelationship
   (extent HLists)
{
attribute list<t> value;
Object HList (in list<t> val, in timestamp iValidTime, in timestamp fValidTime);
Object HList (in list<t> val, in timestamp iValidTime);
Object getValue ( );
};

class HBag extends InstantRelationship
   (extent HBags)
{
attribute bag<t> value;
Object HBag (in bag<t> val, in timestamp iValidTime, in timestamp fValidTime);
Object HBag (in bag<t> val, in timestamp iValidTime);
Object getValue ( );
};

class HArray extends InstantRelationship
   (extent HArrays)
{
attribute array<t> value;
Object HArray (in array<t> val, in timestamp iValidTime, in timestamp fValidTime);
Object HArray (in array<t> val, in timestamp iValidTime);
Object getValue ( );
};

class HDictionary extends InstantRelationship
   (extent HDictionarys)
{
attribute dictionary<t,v> value;
Object HDictionary (in dictionary<t,v> val, in timestamp iValidTime,
    in timestamp fValidTime);
Object HDictionary (in dictionary<t,v> val, in timestamp iValidTime);
Object getValue ( );
};

class HDate extends InstantAttribute
   (extent HDates)
{
attribute date value;
Object HDate (in date val, in timestamp iValidTime, in timestamp fValidTime);
Object HDate (in date val, in timestamp iValidTime);
boolean getValue ( );
};

class HTime extends InstantAttribute
   (extent HTimes)
{
attribute time value;
Object HTime (in time val, in timestamp iValidTime, in timestamp fValidTime);
Object HTime (in time val, in timestamp iValidTime);
octet getValue ( );
};
```

```
class HTimestamp extends InstantAttribute
   (extent HTimestamps)
{
attribute timestamp value;
Object HTimestamp (in timestamp val, in timestamp iValidTime, in timestamp fValidTime);
Object HTimestamp (in timestamp val, in timestamp iValidTime);
char getValue ( );
};

class HInterval extends InstantAttribute
   (extent HIntervals)
{
attribute interval value;
Object HInterval (in interval val, in timestamp iValidTime, in timestamp fValidTime);
Object HInterval (in interval val, in timestamp iValidTime);
string getValue ( );
};
```

Anexo 2 Descrição dos Métodos da Hierarquia

O TVM define os métodos para as operações de sua hierarquia de classes básicas. Entretanto, foram necessárias algumas modificações nesses métodos para adequá-los ao TV_ODMG.

A seguir, são apresentadas, em um pseudo-algoritmo, as descrições desses métodos adaptados. As tabelas A.1, A.2 e A.3 mostram os métodos que correspondem, respectivamente, às classes *TemporalObject*, *VersionedObjectControl* e *TemporalVersion*. As partes em cinza, nas tabelas, mostram os métodos que tiveram como única alteração a substituição da palavra SQL por OQL ou que não sofreram modificações.

TABELA A.1 - Descrição dos métodos da classe *TemporalObject*

Métodos
Object TemporalObject ( ) {alive:=true;}
set<InstantRelationship> getRelationshipHistory (**in string** relName) {OQL SELECT: nos metadados, obtém SE relac. é temporal: OQL SELECT: retorna todos os *InstantRelationship* de relName; SENÃO: retorna valor de relName e rótulo temporal com *nil*;}
InstantRelationship getRelationshipValueAt (**in string** relName, **in timestamp** i) {OQL SELECT: nos metadados, obtém SE relac. é temporal: OQL SELECT: retorna o *InstantRelationship* de relName cujo tempo de validade contém o instante i; SENÃO: retorna valor de relName e rótulo temporal com *nil*;}
void setTemporalAttribute (**in string** attrName, **in Object** newValue) {OQL SELECT: nos metadados, obtém SE attrName é temp.:temp_attrName.updateValue(newValue); SENÃO: ERRO //atributo não é temporal}
void setTemporalAttribute (**in string** attrName, **in Object** newValue, **in timestamp** iValidTime, **in timestamp** fValidTime) {OQL SELECT: nos metadados, obtém SE attrName é temporal: temp_attrName.updateValue (newValue, iValidTime, fValidTime); SENÃO: ERRO //atributo não é temporal}

void setTemporalAttribute (**in string** attrName, **in long** newValue)
{OQL SELECT: nos metadados, obtém SE attrName é temporal:
value:= longToString(newValue);
temp_attrName.updateValue(value);
SENÃO: ERRO //atributo não é temporal}

void setTemporalAttribute (**in string** attrName, **in long** newValue, **in timestamp** iValidTime, **in timestamp** fValidTime)
{OQL SELECT: nos metadados, obtém SE attrName é temporal:
value:= longToString(newValue);
temp_attrName.updateValue(value, iValidTime, fValidTime);
SENÃO: ERRO //atributo não é temporal}

void setTemporalAttribute (**in string** attrName, **in short** newValue)
{OQL SELECT: nos metadados, obtém SE attrName é temporal:
value:= shortToString(newValue);
temp_attrName.updateValue(value);
SENÃO: ERRO //atributo não é temporal}

void setTemporalAttribute (**in string** attrName, **in short** newValue, **in timestamp** iValidTime, **in timestamp** fValidTime)
{OQL SELECT: nos metadados, obtém SE attrName é temporal:
value:= shortToString(newValue);
temp_attrName.updateValue(value, iValidTime, fValidTime);
SENÃO: ERRO //atributo não é temporal}

Métodos

void setTemporalAttribute (**in string** attrName, **in long long** newValue)
{OQL SELECT: nos metadados, obtém SE attrName é temporal:
value:= longlongToString(newValue);
temp_attrName.updateValue(value);
SENÃO: ERRO //atributo não é temporal}

void setTemporalAttribute (**in string** attrName, **in long long** newValue, **in timestamp** iValidTime, **in timestamp** fValidTime)
{OQL SELECT: nos metadados, obtém SE attrName é temporal:
value:= longlongToString(newValue);
temp_attrName.updateValue(value, iValidTime, fValidTime);
SENÃO: ERRO //atributo não é temporal}

void setTemporalAttribute (**in string** attrName, **in unsigned long** newValue)
{OQL SELECT: nos metadados, obtém SE attrName é temporal:
value:= unsignedlongToString(newValue);
temp_attrName.updateValue(value);
SENÃO: ERRO //atributo não é temporal}

void setTemporalAttribute (**in string** attrName, **in unsigned long** newValue, **in timestamp** iValidTime, **in timestamp** fValidTime)
{OQL SELECT: nos metadados, obtém SE attrName é temporal:
value:= unsignedlongToString(newValue);
temp_attrName.updateValue(value, iValidTime, fValidTime);
SENÃO: ERRO //atributo não é temporal}

void setTemporalAttribute (**in string** attrName, **in unsigned short** newValue)
{OQL SELECT: nos metadados, obtém SE attrName é temporal:
value:= unsignedshortToString (newValue);
temp_attrName.updateValue(value);
SENÃO: ERRO //atributo não é temporal}

void setTemporalAttribute (**in string** attrName, **in unsigned short** newValue, **in timestamp** iValidTime, **in timestamp** fValidTime)
{OQL SELECT: nos metadados, obtém SE attrName é temporal:
value:= unsignedshortToString(newValue);
temp_attrName.updateValue(value, iValidTime, fValidTime);
SENÃO: ERRO //atributo não é temporal}

void setTemporalAttribute (**in string** attrName, **in float** newValue)
{OQL SELECT: nos metadados, obtém SE attrName é temporal:
value:= floatToString(newValue);
temp_attrName.updateValue(value);
SENÃO: ERRO //atributo não é temporal}

void setTemporalAttribute (**in string** attrName, **in float** newValue, **in timestamp** iValidTime, **in timestamp** fValidTime)
{OQL SELECT: nos metadados, obtém SE attrName é temporal:
value:= floatToString(newValue);
temp_attrName.updateValue(value, iValidTime, fValidTime);
SENÃO: ERRO //atributo não é temporal}

void setTemporalAttribute (**in string** attrName, **in double** newValue)
{OQL SELECT: nos metadados, obtém SE attrName é temporal:
value:= doubleToString(newValue);
temp_attrName.updateValue(value);
SENÃO: ERRO //atributo não é temporal}

void setTemporalAttribute (**in string** attrName, **in double** newValue, **in timestamp** iValidTime, **in timestamp** fValidTime)
{OQL SELECT: nos metadados, obtém SE attrName é temporal:
value:= doubleToString(newValue);
temp_attrName.updateValue(value, iValidTime, fValidTime);
SENÃO: ERRO //atributo não é temporal}

Métodos

void setTemporalAttribute (**in string** attrName, **in boolean** newValue)
{OQL SELECT: nos metadados, obtém SE attrName é temporal:
value:= booleanToString(newValue);
temp_attrName.updateValue(value);
SENÃO: ERRO //atributo não é temporal}

void setTemporalAttribute (**in string** attrName, **in boolean** newValue, **in timestamp** iValidTime, **in timestamp** fValidTime)
{OQL SELECT: nos metadados, obtém SE attrName é temporal:
value:= booleanToString(newValue);
temp_attrName.updateValue(value, iValidTime, fValidTime);
SENÃO: ERRO //atributo não é temporal}

void setTemporalAttribute (**in string** attrName, **in octet** newValue)
{OQL SELECT: nos metadados, obtém SE attrName é temporal:
value:= octetToString(newValue);
temp_attrName.updateValue(value);
SENÃO: ERRO //atributo não é temporal}

void setTemporalAttribute (**in string** attrName, **in octet** newValue, **in timestamp** iValidTime, **in timestamp** fValidTime)
{OQL SELECT: nos metadados, obtém SE attrName é temporal:
value:= octetToString(newValue);
temp_attrName.updateValue(value, iValidTime, fValidTime);
SENÃO: ERRO //atributo não é temporal}

void setTemporalAttribute (**in string** attrName, **in char** newValue)
{OQL SELECT: nos metadados, obtém SE attrName é temporal:
value:= charToString(newValue);
temp_attrName.updateValue(value);
SENÃO: ERRO //atributo não é temporal}

void setTemporalAttribute (**in string** attrName, **in char** newValue, **in timestamp** iValidTime, **in timestamp** fValidTime)
{OQL SELECT: nos metadados, obtém SE attrName é temporal:
value:= charToString(newValue);
temp_attrName.updateValue(value, iValidTime, fValidTime);
SENÃO: ERRO //atributo não é temporal}

void setTemporalAttribute (**in string** attrName, **in string** newValue) {OQL SELECT: nos metadados, obtém SE attrName é temporal: temp_attrName.updateValue(newValue); SENÃO: ERRO //atributo não é temporal}
void setTemporalAttribute (**in string** attrName, **in string** newValue, **in timestamp** iValidTime, **in timestamp** fValidTime) {OQL SELECT: nos metadados, obtém SE attrName é temporal: temp_attrName.updateValue(newValue, iValidTime, fValidTime); SENÃO: ERRO //atributo não é temporal}
void setTemporalRelationship (**in string** relName, **in Object** newValue) {OQL SELECT: nos metadados, obtém SE relName é temporal: temp_relName.updateValue(newValue); SENÃO: ERRO //relacionamento não é temporal}
void setTemporalRelationship (**in string** relName, **in Object** newValue, **in timestamp** iValidTime, **in timestamp** fValidTime) {OQL SELECT: nos metadados, obtém SE relName é temporal: temp_relName.updateValue(newValue, iValidTime, fValidTime); SENÃO: ERRO //relacionamento não é temporal}
void closeTemporalLabels ( ) {OQL SELECT: nos metadados, obtém os atributos e os relacionamentos temporais do objeto a ser excluído; PARA CADA propriedade retornada: temp_*propriedade*.closeLabels;}

Métodos
void delete ( ) {setTemporalAttribute(alive, false); closeTemporalLabels ( );}
timestamp getLifetimeI ( ); {OQL SELECT: retorna o valor do atributo *vTimei* correspondente ao valor *true* mais recente do atributo temporal *alive*;}
timestamp getLifetimeF ( ) {SE getAlive = false: retorna o valor do atributo *vTimei* correspondente ao valor *true* mais recente do atributo temporal *alive* SENÃO: retorna *nil*;}
set<InstantAttribute> getObjectHistory ( ) {OQL SELECT: retorna os valores do atributo *alive* com os respectivos rótulos de tempo;}
set<InstantAttribute> getAttributeHistory (**in string** attrName) {OQL SELECT: nos metadados, obtém SE *attrName* é temporal: OQL SELECT: retorna todos os *InstantAttribute* de *attrName* SENÃO:retorna o valor de *attrName* e rótulo temporal com *nil*;}
InstantAttribute getAttributeValueAt (**in string** attrName, **in timestamp** i) {OQL SELECT: nos metadados, obtém SE *attrName* é temporal: OQL SELECT: retorna o *InstantAttribute* de *attrName*m cujo tempo de validade contém o instante i SENÃO:retorna o valor de *attrName* e rótulo temporal com *nil*;}

TABELA A.2 - Descrição dos métodos da classe *VersionedObjectControl*

Métodos

Object VersionedObjectControl (**in Object** firstV, **in Object** lastV)
{configurationCount:=0;
currentVersion:=lastV;
firstVersion:=firstV;
lastVersion:=lastV;
nextVersionNumber:=3;
userCurrentFlag:=false;
versionCount:=2;
contains:=(firstV, lastV);}

void updateConfigurationCount ( ) {varX := OQL SELECT: count (versões referenciadas no rel. contains com isConfiguration<>true); setTemporalAttribute(configurationCount, varX);}
void updateNextVersionNumber ( ) {nextVersionNumber:= nextVersionNumber+1;}
void delete (**in short** entityId, **in short** classId) {OQL SELECT: SE existe referência a partir de outro objeto ou versão: ERRO SENÃO: super.delete( ); updateVersionCount( );}
void updateVersionCount ( ) {varX := OQL SELECT: count (versões referenciadas no rel. contains com getVersionNr<>0); setTemporalAttribute(versionCount, varX);}
Object getFirstVersion ( ) {OQL SELECT: retorna o valor do atributo *firstVersion*;}
Object getLastVersion ( ) {OQL SELECT: retorna o valor do atributo *lastVersion*;}
Object getCurrentVersion ( ) {OQL SELECT: retorna o valor do atributo *currentVersion*;}
void setCurrentVersion ( ) {setUserCurrentFlag (*false*) ; setTemporalAttribute(currentVersion, lastVersion) ; varX:=objeto.contains; objVersionado:= OQL SELECT: seleciona o elemento de varX onde elemento.tvOID.getVersionNr=0; PARA CADA prop. de objVersionado: objVersionado.propriedade:=currentVersion.propriedade;}

Métodos
void setCurrentVersion (**in Object** newVersionId) {SE: newVersionId existe: setTemporalAttribute(currentVersion, newVersionId) ; setUserCurrentFlag(true) ; varX:=objeto.contains; objVersionado:= OQL SELECT: seleciona o elemento de varX onde elemento.tvOID.getVersionNr=0; PARA CADA propriedade de objVersionado: objVersionado.propriedade:=currentVersion.propriedade SENÃO: ERRO}
void setFirstVersion (**in Object** first) {setTemporalAttribute(firstVersion, first) ;}
void setLastVersion (**in Object** last) {setTemporalAttribute(lastVersion, last) ;}
short getConfigurationCount ( ) {OQL SELECT: retorna o valor do atributo *configurationCount*;}
short getVersionCount ( ) {OQL SELECT: retorna o valor do atributo *versionCount*;}
short getNextVersionNumber ( ) {OQL SELECT: retorna o valor do atributo *nextVersionNumber*;}
boolean getUserCurrentFlag ( ) {OQL SELECT: retorna o valor do atributo *userCurrentFlag*;}
void setUserCurrentFlag (**in boolean** flag) {setTemporalAttribute(userCurrentFlag, flag) ;}

Alguns métodos (*getClassName*, *getClassId*, *getCorrespondence*, *getCorrespondenceAsc*, *getCorrespondenceDesc*, *getEntityId* e *getTVOid*) não precisaram ter seus algoritmos alterados, mesmo tendo sido realocados da classe *Object* para a *TemporalVersion*.

TABELA A.3 - Descrição dos métodos da classe *TemporalVersion*

Métodos

Object TemporalVersion ( )
{OQL SELECT: nos metadados, SE classe instanciada é subclasse na hierarquia de extensão: ERRO
//se é subclasse precisa de ascendente
SENÃO:
configuration:= false;
status:= "W";
ent := TVEntity( );
varE := ent.getId; varC:=getClassId;
tvOID:=new OIDt(varE, varC, 1);

Object TemporalVersion (**in set**<Object> ascendant);
{SE verifyAscendId (ascendId):
varC:= getClassId;
varX:= um dos elementos de ascendId;
varE:= varX.getTVOID.getEntityNr;
SE existe objeto na classe varC com entidade varE: ERRO //pois deveria ter sido informado um predec.
SENÃO:
PARA CADA elemento de ascendId:
SE: getDescendant=nil: setDescendant(novoOID)
SENÃO: addDescendant(novoOID); //se exceder descendentes retorna erro
ascendant.value:= ascendId;
configuration:= false;
status:= "W";
tvOID:=new OIDt(varE, varC, 1);
SENÃO: ERRO //conjunto de ascendentes inválido}

Métodos

```
Object TemporalVersion (in string entityName)
{SE verifyEntityName(entityName): OQL SELECT: na classe instanciada,
   SE há algum objeto.getTVOID.getEntityName=entityName:ERRO
                          //se entidade existente na classe, então deveria ser informado predecessor
   SENÃO:
    OQL SELECT: nos metadados, SE classe instanciada é subclasse na  hierarquia de extensão:
       OQL SELECT: nos metadados, obtém classe ascendente;
       OQL SELECT: na classe ascendente, SE há algum objeto com entidade=entityName:
           SE existe um objeto.belongs=nil: varX:= objeto
           SENÃO: varX:= objeto.getVersionedObjectId;
           SE varX.getDescendant=nil: varX.setDescendant(novoOID);
           SENÃO: varX.addDescendant(novoOID);     //se exceder descendentes retorna erro
          ascendant.value:=varX;
       SENÃO: ERRO          //se é subclasse na hierarquia de extensão precisa de ascendente
    configuration:= false;
    status:= "W";
    varE:= getEntityId(entityName) ;
    varC:= getClassId ;
    tvOID:=new OIDt(varE, varC, 1);
SENÃO: ERRO        //entidade inexistente}
```

```
Object TemporalVersion (in short entityId, in short classId, in short versionId)
{OQL SELECT: na classe classId, SE existe objeto com entidade entityId e versão versionId: ERRO
                                                                    //já existe essa versão de objeto
SENÃO:
  OQL SELECT: nos metadados e em TVEntity, SE existe classe classId e entidade entityId:
    OQL SELECT: nos metadados, SE classId é subclasse na hierarquia de extensão:
        OQL SELECT: nos metadados, obtém classe ascendente;
        OQL SELECT: na classe ascendente, SE há algum objeto com entidade=entityId:
            SE objeto.belongs=nil: varX:= objeto
            SENÃO: varX:= objeto.getVersionedObjectId;
            SE varX.getDescendant=nil: varX.setDescendant(novoOID)
            SENÃO: varX.addDescendant(novoOID) ;        //se exceder descendentes retorna erro
            ascendant.value:=varX;
        SENÃO: ERRO          //se é subclasse precisa de ascendente
    configuration:= false;
    status:= "W";
    //início - cria tvOID
    varE:= entityId;
    varC:= classId;
    varV:= versionId;
    SE varV = 0:         //objeto versionado
      OQL SELECT: SE existem 2 versões na classe varC com  entidade varE que são as versões 1 e 2, e
      essas ainda não estão associadas a um obj. versionado:    //só insere na 1ª derivação
          tvOID:=new OIDt(varE, varC, varV);
      SENÃO: ERRO
    SENÃO:
       SE versionId<>nil:
          SE varV = findVersion(varE, varC): tvOID:=new OIDt(varE, varC, varV);
          SENÃO: ERRO         //número de versão impróprio
       SENÃO: ERRO         //não foi informado um número de versão
    //fim - cria tvOID
  SENÃO: ERRO     //não existe entidade e classe}
```

Métodos

Object TemporalVersion (**in set**<Object> predecId, **in set**<Object> ascendId, **in boolean** config);
{SE verifyAscendId (ascendId):
SE predecId<>nil e todos os elementos de predecId estão contidos na mesma classe e pertencem a mesma entidade:
PARA CADA elemento de ascendId:
SE: getDescendant=nil: setDescendant(novoOID)
SENÃO: addDescendant(novoOID) ; //se exceder descendentes retorna erro
ascendant.value:= ascendId;
PARA CADA elemento de predecId:
SE: getSuccessor=nil: setSuccessor (novoOID)
SENÃO: addSuccessor (novoOID);
predecessor.value:= predecId;
configuration:= config;
status:= "W";
varX:= um dos elementos de predecId;
varE:= varX.getTVOID.getEntityNr;
varC:= varX.getTVOID.getClassNr ;
varV=findVersion(varE, varC);
tvOID:=new OIDt(varE, varC, varV);
SENÃO: ERRO //predecessores impróprios ou não foram informados
SENÃO: ERRO //conjunto de ascendentes inválido}

---

boolean addAscendant (**in Object** ascendId)
{varC:=ascendId.getClassName;
SE objeto.getAscendant.length>1: varA:= 'n'
SENÃO: varA:= '1' ;
SE varA = objeto.getCorrespondenceAsc(varC): ERRO
SENÃO: setTemporalAttribute(ascendant.value, getAscendant+ascendId); }

---

boolean addDescendant (**in Object** descendId)
{varC:=descendId.getClassName;
SE objeto.getDescendant.length>1: varA:= 'n'
SENÃO: varA:= '1' ;
SE varA = objeto.getCorrespondenceDesc(varC): ERRO
SENÃO: setTemporalAttribute(descendant.value, getDescendant+descendId); }

---

set<Object> getCompletObject ( )
{varX := objeto;
OQL SELECT: nos metadados, obtém SE objeto possui relacionamento de agregação e ascendentes:
SE possui agregações ou ascendentes:
OQL SELECT: na classe, obtém objetos referenciados em relacionamentos de agregação, se os possui, e objetos referenciados como ascendentes, se os possui;
PARA CADA objeto referenciado: objetoReferenciado.getCompleteObject;
varX:=varX+objetos referenciados;
retorna varX;}

---

boolean verifyEntityName (**in string** entityName)
{OQL SELECT: na classe TVEntity, SE existe entityName: retorna *true*
SENÃO: retorna *false*;}

---

short findVersion (**in short** entityId, **in short** classId)
{OQL SELECT: na classe classId, SE não existe objeto com entidade = entityId: retorna 1
SENÃO: SE objeto.belongs <> nil: retorna objeto.belongs.getNextVersionNumber;
SENÃO: retorna 2;}

---

string getClassName ( )
{OQL SELECT: nos metadados em tv_class, retorna o nome da classe cujo id=objeto.tvOID.getClassNr;
}

---

string getClassId ( )
{OQL SELECT: nos metadados em tv_class, retorna o identificador da classe;}

---

short getEntityId (**in string** entityName)
{OQL SELECT: na classe TVEntity, retorna getId do objeto que possuir getName= "entityName";}

OIDt getTVOid ( )
{objeto.tvOID;}

Métodos

```
void delete (in boolean allReferences)
{SE isDeleteAllowed (allReferences) =true
   SE objeto.getTVOID.getVersionNr = 0 :     //obj. versionado, então devem ser apagadas todas as suas
                                                                                                    versões
     OQL SELECT: na classe, obtém todas as versões
       PARA CADA versão:
         SE versão.isDeleteAllowed(allReferences) = false: ERRO; //não é possível excluir alguma versão
       PARA CADA versão (da última p/ 1ª na árvore de derivação):
          versão.delete(allReferences) ;
   //exclui normalmente
   SE objeto.getAscendant<>nil:
     PARA CADA objeto ascendente:
       SE ascendente.getDescendant.length=1: ascendente.setDescendant(nil)
       SENÃO: ascendente.removeDescendant(objeto) ;
   SE objeto.getPredecessor<>nil:
     PARA CADA objeto predecessor:
       SE predecessor.getSuccessor.length=1: predecessor.setSuccessor(nil)
       SENÃO: predecessor.removeSuccessor(objeto) ;
   objeto.setTemporalAttribute(status,'d') ;
   SE allReferences = true:
     PARA CADA objeto relacionado: apaga a referência (definida por rel. de associação) a ele;
   //atualiza o controle do objeto versionado
   SE objeto.belongs<>nil:
     objeto.belongs.updateConfigurationCount;
     objeto.belongs.updateVersionCount;
     SE objeto.belongs.getCurrentVersion=objeto:
       varAt:= objeto.belongs.currentVersion.getCurrent.getTempLabel.getVTimei-1;
       varI:= objeto.belongs.currentVersion.getValueAt(varAt);
       varLabel:= varI.getTempLabel;
       ENQUANTO varLabel.getTTimef <>nil FAÇA:    //percorre o histórico do atributo currentVersion
                                                  //até encontrar uma ex-versão corrente que não tenha sido excluída
          varAt:= varLabel.getVTimei-1;
          varI:= objeto.belongs.currentVersion.getValueAt(varAt);
          varLabel:= varI.getTempLabel;
       objeto.belongs.setCurrentVersion(varI.getValue);
     SE objeto.belongs.getLastVersion=objeto:
       varAt:= objeto.belongs.lastVersion.getCurrent.getTempLabel.getVTimei-1;
       varI:= objeto.belongs.lastVersion.getValueAt(varAt);
       varLabel:= varI.getTempLabel;
       ENQUANTO varLabel.getTTimef <>nil FAÇA:
                  //percorre o histórico do atributo lastVersion até encontrar uma ex-versão corrente
                  // que não tenha sido excluída
          varAt:= varLabel.getVTimei-1;
          varI:= objeto.belongs.lastVersion.getValueAt(varAt);
          varLabel:= varI.getTempLabel;
       objeto.belongs.setLastVersion(varI.getValue) ;
   //atualiza alive
   super.delete;
   SE objeto.belongs=nil OU objeto.belongs.versionCount=0: objeto.belongs.delete;     //apaga o controle
 SENÃO: ERRO;     //não é permitido realizar a exclusão}
```

```
void deleteObjectTree (in boolean allReferences)
{SE isDeleteTreeAllowed (allReferences) =true:
    SE getDescendant <> nil              //possui descendentes
       PARA CADA objeto descendente:
          SE descendente.isDeleteTreeAllowed (allReferences) =false: ERRO;
       PARA CADA objeto descendente: descendente.deleteObjectTree (allReferences);
    SENÃO: delete(allReferences);
 SENÃO: ERRO; }
```

Métodos

Object getVersionedObjectId( )
{OQL SELECT: na classe, SE existe objeto com tvOID que possui entidade e classe iguais ao do objeto e
versão 0: retorna o objeto;
SENÃO: retorna nil;}

Object derive (**in set<**Object**>** versionId)
{SE versionId.lenght=0: ERRO //não foi informado um predecessor
SENÃO:
SE algum objeto em versionId for configuração: //tem que pegar predecessor
PARA CADA objeto em versionId:
SE objeto.isConfiguration:
versionId:=versionId – objeto;
versionId:= versionId + objeto.getPredecessor;
derive(versionId);
SENÃO: // nenhum objeto é configuração
varV:=set(nil);
OQL SELECT: nos metadados, SE subclasse na hierarquia de extensão:
varA:=set(nil);
PARA CADA objeto em versionId:
SE objeto.getAscendant NOT IN varA: varA:= varA+objeto.getAscendant;
PARA CADA objeto em varA:
SE objeto.getVersionedObjectId NOT IN varV: varV:= varV+objeto.getVersionedObjectId;
varY:= new TemporalVersion(versionId, varV, false); //cria 2ª versão
SE versionId.length=1 E versionId.getElement.belongs=nil: //primeira derivação
versao:= o elemento único de versionId;
varE:= versao.getEntityNr;
varC:= versao.getClassNr;
SE getStatus= 'w': setTemporalAttribute(status,'s');
//cria controle do objeto versionado;
varX:=new VersionedObjectControl (objeto, varY);
versao.belongs:=varX;
varY.belongs:=varX;
varZ:= new TemporalVersion (varE,varC,0); //obj. versionado
varZ.belongs:=varX;
varX.contains:= (versao,varY,varZ);
objeto.setSuccessor(varY);
SENÃO: //derivação qualquer
varD:= algum elemento de versionId;
varY.belongs:= varD.belongs;
PARA CADA objeto de versionId: addSuccessor (varY);
//atualiza o controle do obj. versionado pelo relacionamento belongs
SE varY.belongs.userCurrentFlag= *false*: varY.belongs.setTemporalAttribute(currentVersion, varY);
varY.belongs.setTemporalAttribute(lastVersion, varY);
varY.belongs.updateVersionCount;
varY.belongs.updateNextVersionNumber;}

Object getVersionedObjectControl ( )
{retorna o valor de objeto.belongs;}

boolean verifyAscendId (**in set<Object>** ascendId)
{OQL SELECT: nos metadados, obtém classes ascendentes;
varX:= OQL SELECT: obtém objetos das classes ascendentes;
SE os elementos de ascendId estão contidos em varX e todos esses elem. pertencem a mesma entidade:
SE ascendId possui mais de um elemento: verifica nos metadados SE está de acordo com a
correspondênica definida pela herança por extensão: retorna *true*
SENÃO: retorna *false*;
SENÃO: retorna *true*
SENÃO: retorna *false*}

Métodos

Object derive (**in set**<Object> versionId, **in set**<Object> ascendId, **in boolean** config)
{SE versionId.lenght=0: ERRO //não foi informado um predecessor
SENÃO:
SE algum objeto em versionId for configuração: //tem que pegar predecessor
PARA CADA objeto em versionId:
SE objeto.isConfiguration:
versionId:= versionId – objeto;
versionId:= versionId + objeto.getPredecessor;
derive(versionId);
SENÃO: // nenhum objeto é configuração
varY:= new TemporalVersion(versionId, ascendId, false); //cria 2ª versão
SE versionId.length=1 E versionId.getElement.belongs=nil: //primeira derivação
versao:= o elemento único de versionId;
varE:= versao.getEntityNr;
varC:= versao.getClassNr;
SE getStatus= 'w': setTemporalAttribute(status,'s');
//cria controle do objeto versionado;
varX:=new VersionedObjectControl (objeto, varY);
versao.belongs:=varX;
varY.belongs:=varX;
varZ:= new TemporalVersion (varE,varC,0); //obj. versionado
varZ.belongs:=varX;
varX.contains:= (versao,varY,varZ);
objeto.setSuccessor(varY);
SENÃO: //derivação qualquer
varD:= algum elemento de versionId;
varY.belongs:= varD.belongs;
PARA CADA objeto de versionId: addSuccessor (varY);
//atualiza o controle do obj. versionado pelo relacionamento belongs
SE varY.belongs.userCurrentFlag= *false*: varY.belongs.setTemporalAttribute(currentVersion, varY);
varY.belongs.setTemporalAttribute(lastVersion, varY);
varY.belongs.updateVersionCount;
varY.belongs.updateNextVersionNumber;}

void setAscendant (**in Object** ascendId) {setTemporalAttribute(ascendant.value, ascendId);}
void addSucessor (**in Object** succId) {setTemporalAttribute(successor.value, getSuccessor+succId);}
set<Object> getDescendant ( ) {retorna o valor de descendant.value;}
set<Object> getDescendant (**in boolean** all) {SE all= *true*: OQL SELECT: retorna os seus descendentes de todos os níveis da hierarquia de extensão SENÃO: getDescendant;}
set<Object> getPredecessor ( ) {retorna o valor de predecessor.value;}
set<Object> getSucessor (**in boolean** onlyConfigured) {SE onlyConfigured = *false*: OQL SELECT: retorna o valor de successor.value; SE onlyConfigured *true*: OQL SELECT: retorna o valor de successor.value cujo valor do atributo configuration é *true*;}
set<Object> getAscendant ( ) {retorna o valor de ascendant.value;}
set<Object> getAscendant (**in boolean** all) {SE all= *true*: OQL SELECT: retorna os seus ascendentes de todos os níveis da hierarquia de extensão SENÃO: getAscendant;}
boolean removeAscendant (**in Object** ascendId) {SE getAscendant.lenght=1: retorna *false*; SENÃO: setTemporalAttribute(ascendant.value, getAscendant - ascendId);}
void removeDescendant (**in Object** descendId) {setTemporalAttribute(descendant.value, getDescendant - descendId);}

Métodos
void removeSuccessor (**in Object** succId) {setTemporalAttribute(successor.value, getSuccessor - succId);}
void setDescendant (**in Object** descendId) {setTemporalAttribute(descendant.value, descendId);}
boolean setSucessor (**in set<Object>** succId) {setTemporalAttribute(successor.value, succId);}

void promote (**in boolean** allAscendant, **in boolean** allReferenced)
{SE getStatus= 'w':
SE allAscendant *true*:
OQL SELECT: obtém objetos ascendentes
PARA CADA objeto ascendente:
SE objeto.getStatus= 'w': objeto.promote(true, false);
SENÃO: PARA CADA objeto ascendente: SE objeto.getStatus= 'w': ERRO;
SE allReferences *true*:
OQL SELECT: obtém objetos das classes relacionadas
PARA CADA objeto:
SE getStatus= 'w': objeto.promote(true, false);
setTemporalAttribute(status,'s');
SE getStatus( )= 's':
SE allAscendant *true*:
OQL SELECT: obtém objetos ascendentes
PARA CADA objeto ascendente:
SE objeto.getStatus= 'w' OU objeto.getStatus= 's':
SE objeto.getStatus= 'w': objeto.setTemporalAttribute(status,'s');
objeto.promote(true, false);
SENÃO: PARA CADA objeto ascendente:
SE objeto.getStatus= 'w' OU objeto.getStatus= 's': ERRO;
SE allReferences *true*:
OQL SELECT: obtém objetos das classes relacionadas
PARA CADA objeto:
SE getStatus= 'w' OU getStatus= 's':
SE objeto.getStatus= 'w': objeto.setTemporalAttribute(status,'s');
objeto.promote(true, false);
setTemporalAttribute(status,'c');}

boolean isDeleteAllowed (**in boolean** allReferences) {SE getStatus= 'c' OU getStatus= 'd': retorna *false*; SE getDescendant diferente de vazio: retorna *false*; SE getSuccessor diferente de vazio: retorna *false*; SE isConfiguration: retorna *false*; OQL SELECT: nos metadados, obtém SE objeto faz parte de relacionamento de agregação (como componente): retorna *false*; SE allReferences= *true*: retorna *true*; SENÃO: OQL SELECT: obtém SE existe referência (rel. de associação) para ou a partir do objeto: retorna *false* SENÃO: *true*;}
boolean isDeleteTreeAllowed (**in boolean** allReferences) {SE getStatus= 'c' OU getStatus= 'd': retorna *false*; SE getSuccessor diferente de vazio: retorna *false*; SE isConfiguration: retorna *false*; OQL SELECT: nos metadados, obtém SE objeto faz parte de relacionamento de agregação (como componente): retorna *false*; SE allReferences = *false*: OQL SELECT: obtém SE existe referência (rel. de associação) para ou a partir do objeto:retorna *false*; SE getDescendant diferente de vazio: PARA CADA objeto descendente: SE isDeleteTreeAllowed(true) = *false*: retorna *false*; SENÃO: retorna *true*;}

Métodos
char getStatus ( ) {OQL SELECT: retorna o valor do atributo status;}
void setStatus (**in char** newStatus) {setTemporalAttribute(status, newStatus);}
boolean isConfiguration ( ) {OQL SELECT: retorna o valor do atributo configuration;}
string getCorrespondence (**in string** className) {OQL SELECT: nos metadados, retorna a correspondência da classe com a classe className;}
string getCorrespondenceAsc (**in string** ascendClassName) {OQL SELECT: nos metadados, retorna a correspondência da classe com sua ascendente ascendClassName, apenas a cardinalidade da ascendente;}
string getCorrespondenceDesc (**in string** descendClassName) {OQL SELECT: nos metadados, retorna a correspondência da classe com sua ascendente descendClassName, apenas a cardinalidade da descendente; }

Bibliografia

[BAR 2001] BARRY & ASSOCIATES, Inc. **ODMG Compliance**. Disponível em: <http://www.barryandassociates.com/odmg-compliance.html>. Acesso em: 11 set. 2001.

[BAR 98] BARRY, Doug. **ODMG 2.0**: A Standard for Object Storage. 1998. Disponível em: <http://www.odmg.org>. Acesso em: 08 jun. 2000.

[BEL 96] BELLOSTA, M.; WREMBEL, R.; JOMIER, G. Management of Schema Versions and Versions of Schema Instance in a Multiversion Database. 1996. Disponível em: <http://citeseer.nj.nec.com/ 385427.html>. Acesso em: 14 jan. 2001.

[BER 97] BERTINO, E. et al. **Approaches to Handling Temporal data in Object-Oriented Databases**. Milano: Dipartimento di Scienze dell'Informazione Universit' a di Milano, 1997. (Technical Report 192-97).

[BER 98] BERTINO, E. et al. Extending the ODMG Object Model with Time. In: EUROPEAN CONFERENCE ON OBJECT-ORIENTED PROGRAMING, ECOOP, 1998, Brussels. **Proceedings**... [S.l.:s.n.], 1998. p. 41-66. (Lecture Notes in Computer Science, n.1445).

[BIL 90] BILIRIS, A. Modeling Design Object Relationship in PEGASUS. In: INTERNATIONAL CONFERENCE DATA ENGINEERING – ICDE, USA, 1990. **Proceedings**... [S.l.]: IEEE Computer Society, 1990. p. 228-236.

[BUS 94a] BUSSE, R.; FRANKHAUSER, P.; NEUHOLD, E. Federated Schemata in ODMG. In: INTERNATIONAL EAST/WEST DATABASE WORKSHOP, 2., 1994, Klagenfurt, Austria. **Proceedings**... [S.l.:s.n.],1994. p. 356-379.

[BUS 94b] BUSSE, R. et al. IRO-DB: An object-oriented approach towards federated and interoperable DBMS. In: INTERNATIONAL WORKSHOP ON ADVANCES IN DATABASES AND INFORMATION SYSTEM, 1994, Moscow. **Proceedings**... [S.l.]: Russian Academy of Sciences, 1994.

[CAT 2000] CATTELL, R. G. G. et al**. The Object Data Standard:** ODMG 3.0. San Francisco: Morgan Kaufmann, 2000. 280p.

[CEL 90] CELARRY, W.; JOMIER, G. Consistency of Versions on Object-Oriented Databases. In: VLDB CONFERENCE, 16., 1990, Brisbane, Australia. **Proceedings**... [S.l.:s.n.], 1990. p. 432-441.

[CHA 97] CHAVDA, M.; WOOD, P. Towards an ODMG-Compliant Visual Object Query Language. In: VLDB CONFERENCE, 23., 1997, Athens,

Greece. **Proceedings**... [S.l.]: Morgan Kaufmann, 1997. p. 456-465.

[CLA 98a] CLAYPOOL, K.; HIN, J.; RUNDENSTEINER, E. **SERF**: Schema Evolution through an Extensible, Re-usable and Flexible Framework. Massachusetts: Worcester Polytechnic Institute, 1998. (Technical Report WPI-CS-TR-98-9).

[CLA 98b] CLAYPOOL, K.; HIN, J.; RUNDENSTEINER, E. **OQL SERF**: an ODMG Implementation of the Template-Based Schema Evolution Framework. Massachusetts: Worcester Polytechnic Institute, 1998. (Technical Report WPI-CS-TR-98-14).

[CLA 99] CLAYPOOL, K.; RUNDENSTEINER, E.; HEINEMAN, T**. Extending Schema Evolution to Handle Object Models with Relationships**. Massachusetts: Worcester Polytechnic Institute, 1999. (Technical Report WPI-CS-TR-99-15).

[COO 97] COOPER, Richard. **Object Databases**: An ODMG Approach. Boston: International Thomson Computer Press, 1997. 499p.

[DOU 96] DOUCET, A. et al. Integrity Constraints in Multiversion Databases. In: BRITISH NATIONAL CONFERENCE ON DATABASES, 14., 1996, Edinburgh, Scotland. **Proceedings**... Edinburgh: Springer-Verlag, 1996. p. 56-73. (Lecture Notes in Computer Science, n. 1094).

[DOU 97] DOUCET, A; MONTIES, S. Versions of integrity constraints in multiversion databases. In: DEXA, 1997, Toulouse, France. **Proceedings**... [S.l.:s.n.], 1997. p. 252-261. (Lecture Notes in Computer Science, n. 1308).

[DUM 98] DUMAS, M.; FAUVET, M.-C.; SCHOLL, P.-C. Handling temporal grouping and pattern-matching queries in a temporal object model. In: INTERNATIONAL CONFERENCE ON INFORMATION AND KNOWLEDGE MANAGEMENT, 7., 1998, Bethesda. **Proceedings**... [S.l.]:ACM Press, 1998. p. 424-431.

[DUM 99a] DUMAS, M.; FAUVET, M.-C.; SCHOLL, P.-C. **TEMPOS**: A Temporal Database Model Seamlessly Extending ODMG. Grenoble: University of Grenoble, 1999. 45p. (Research report 1013-I-LSR-IMAG).

[DUM 99b] DUMAS, M.; FAUVET, M.-C.; SCHOLL, P.-C.. A Representation independent temporal extension of ODMG's Object Query Language. In: FRENCH CONFERENCE ON ADVANCED DATABASES, BDA, 1999, Bordeaux, France. **Proceedings**... [S.l.:s.n.],1999.

[DUM 99c] DUMAS, M.; FAUVET, M.-C.; SCHOLL, P.-C. Updates and application migration support in an ODMG temporal extension. In: ER - INTERNATIONAL WORKSHOP ON EVOLUTION AND CHANGE IN DATA MANAGEMENT, ECDM, 1., 1999, Paris. **Proceedings**...

[S.l.]: Springer-Verlag, 1999. p. 24-35. (Lecture Notes in Computer Science, n. 1727).

[DUM 2001] DUMAS, M.; FAUVET, M.-C.; SCHOLL, P.-C. **TEMPOS**: A Platform for Developing Temporal Applications on top of Object DBMS. Grenoble: University of Grenoble, 2001. 39 p. (Techinical Report TR-53).

[FEG 98] FEGARAS, L.; ELMASRI, R. A Temporal Object Query Language. In: TIME - INTERNATIONAL WORKSHOP ON TEMPORAL REPRESENTATION AND REASONING, 5., 1998. **Proceedings**... [S.l.:s.n.], 1998. p.51-59.

[FRA 2000] FRANCONI, E.; GRANDI, F.; MANDREOLI, F. A Semantic Approach for Schema Evolution and Versioning in Object-Oriented Databases. In: INTERNATIONAL WORKSHOP IN DESCRIPTION LOGICS, 2000, Aachen, Germany. **Proceedings**... [S.l.:s.n.], 2000. p. 99-112.

[FUR 98] FURLAN, J. D. **Modelagem de Objetos através da UML**. São Paulo: Makron Books, 1998. 329p.

[GAL 99] GALANTE, Renata de Matos. **O Conceito de Versão Aplicado a Mecanismos de Evolução de Esquemas**. 1999. p.12-17. Trabalho Individual I (Mestrado em Ciência da Computação) – Instituto de Informática, Universidade Ferderal do Rio Grande do Sul, Porto Alegre.

[GAN 94] GANÇARSKI, S.; JOMIER, G. Managing Entity Versions within their Contexts: A Formal Approach. In: DEXA, 1994, Athens, Greece. **Proceedings**... Berlin: Springer-Verlag, 1994. p. 400-409. (Lecture Notes in Computer Science, n. 856).

[GAN 99] GANÇARSKI, S. Database versions to represent bitemporal databases. In: DEXA, 1999, Florence. **Proceedings**... [S.l.]: Springer,1999. p. 832-941. (Lecture Notes in Computer Science, n. 1677).

[GAR 97] GARDARIN, G.; FINANCE, B.; FANKHAUSER, P. Federating Object-Oriented and Relational Databases: The IRO-DB Experience. In: IFCIS INTL. CONFERENCE ON COOPERATIVE INFORMATION SYSTEMS - CoopIS, 2., 1997, Kiawah Island, South Carolina. **Proceedings**... Los Alamitos: IEEE Computer Society, 1997. p. 2-13.

[GOL 95] GOLENDZINER, Lia Goldstein; SANTOS, Clesio Saraiva dos. **Um modelo de versões para banco de dados orientado a objetos**. 1995. 147p. Tese de Doutorado (Doutorado em Ciência da Computação) – Instituto de Informática, Universidade Ferderal do Rio Grande do Sul, Porto Alegre.

[GRA 2002] GRANDI, F.; MANDREOLI, F. **A Formal Model for Temporal Schema Versioning in Object-Oriented Databases**. Bologna: [s.n.],

2002. (Technical Report TR-68). Disponível em: <ftp://ftp-db/deis.unibo.it/pub/fabio/TR/TR-68.pdf>. Acesso em: 15 jan. 2002.

[GRA 99] GRANDI, F.; MANDREOLI, F. ODMG language extensions for generalized schema versioning support. In: INTERNATIONAL WORKSHOP ON EVOLUTION AND CHANGE IN DATA MANAGEMENT, ECDM, 1., 1999, Paris. **Proceedings**... [S.l.:s.n.], 1999. p. 36-47.

[GRA 2000] GRANDI, F.; MANDREOLI, F.; SCALAS, M. A Generalized Modeling Framework for Schema Versioning Support. In: AUSTRALASIAN DATABASE CONFERENCE, 11., 2000, Canberra. **Proceedings**... [S.l.:s.n.], 2000. p. 33-44.

[JOR 98] JORDAN, D. In: **C++ Object Databases**: Programming with the ODMG Standard. Reading: Addison-Wesley, 1998. 456p.

[KAK 96a] KAKOUDAKIS, I.; THEODOULIDIS, B. **The TAU Temporal Object Model**. Manchester: University of Manchester (UMIST), 1996. 39p. (Technical Report TR-96-4, TimeLab).

[KAK 96b] KAKOUDAKIS, I.; THEODOULIDIS, B. **The TAU Temporal Object Definition Language**. Manchester: University of Manchester (UMIST), 1996. 28p. (Technical Report TR-96-6, TimeLab).

[KAK 96c] KAKOUDAKIS, I.; THEODOULIDIS, B. **The TAU Time Model**. Manchester: University of Manchester (UMIST), 1996. 24p. (Technical Report TR-96-5, TimeLab).

[KAK 96d] KAKOUDAKIS, I.; THEODOULIDIS, B. **TAU System Architecture and Design**. Manchester: University of Manchester (UMIST), 1996. 15p. (Technical Report TR-96-9, TimeLab).

[KHO 94] KHOSAHFIAN, S. **Banco de Dados Orientado a Objeto**. Rio de Janeiro: Infobook, 1994. 353p.

[MOR 2001a] MORO, M. M. **Modelo Temporal de Versões**. 2001. 120p. Dissertação de Mestrado (Mestrado em Ciência da Computação) – Instituto de Informática, Universidade Ferderal do Rio Grande do Sul, Porto Alegre.

[MOR 2001b] MORO, Mirella Moura; GELATTI, Paôla Cristina; GOMES, Cláudio Hessel Peixoto; ROSSETTI, Lialda Lúcia Fernandes; ZAUPA, Aglaê Pereira; EDELWEISS, Nina; SANTOS, Clésio Saraiva dos. **Linguagem de Consultas para o Modelo Temporal de Versões**. Porto Alegre: PPGC da UFRGS, 2001. 92 p. (Relatório de Pesquisa, 308).

[ODM 2001] ODMG Compliance. Disponível em: <http://www.odmg.org/odmg/compliancelist.htm>. Acesso em: 11 set. 2001.

[OGA 99] OGATA, H.; RODRIGUEZ, L.; YANO, Y. An Access Mechanism for a

Temporal Versioned Object-Oriented Database. **IEICE Transactions on Information and Systems**, [S.l.], v. E82-D, n. 1, p. 128-135, Jan. 1999.

[OGA 2001] OGATA, H.; RODRIGUEZ, L.; YANO, Y. A Temporal Versioned Object-Oriented Data Schema Model. **Computer and Mathematics with Applications**: An International Journal, [S.l.], v. 41, p. 177-192, 2001.

[ORI 2001] ORIENT TECHNOLOGIES. **Orient ODBMS Main User Guide**. 2001. Disponível em: <http://www.orientechnologies.com/docs/MainGuide.htm>. Acesso em: 06 ago. 2001.

[RIE 99] RIEDEL, H. Outdating Outdated Objects. In: ECOOP WORKSHOP ON OBJECT-ORIENTED DATABASES, 1., 1999, Lisboa. **Proceedings**... [S.l.:s.n.], 1999. p. 73-83.

[SAC 95] SACHWEH, S.; SCHÄFER, W. Version Management for Tightly Integrated Software Engineering Environments. In: INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING ENVIRONMENTS, 7., 1995, Leiden, The Netherlands. **Proceedings**... [S.l.]: IEEE Computer Society Press, 1995. p. 21-31.

[SAM 2000] SAMPAIO, P. Design and Implementation of a Deductive Query Language for ODMG Compliant Object Databases. In: EDBT; PHD WORKSHOPS, 2000, Konstanz, Germany. **Proceedings**... [S.l.:s.n.], 2000.

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

[TAN 93] TANSEL, C. G. et al. **Temporal Databases**: theory, design and implementation. Redwood City: Benjamin/Cummings, 1993.

[TES 97] TESCH, T.; WÄSCH, J. Global Nested Transaction Management for ODMG-Compliant Multi-Database Systems. In: INTERNATIONAL CONFERENCE ON INFORMATION AND KNOWLEDGE MANAGEMENT, 6., 1997, Las Vegas. **Proceedings**... [S.l.]: ACM, 1997. p. 67-74.

[TOO 99] TOOBIS - Temporal Object-Oriented Databases within Information Systems. [S.l.:s.n.], 1999. p.12-34. (Final Project Report T12R.3, Espirit Project 20671).

[TOR 2001] TORRES, M.; SAMOS, J. Generation of External Schemas in ODMG Databases. In: IDEAS, 2001, Grenoble. **Proceedings**... [S.l.]: IEEE Computer Society Press, 2001. p. 89-98.